USING FEATURE ENGINEERING AND DATA MINING OF ACCELEROMETER DATA TO DETERMINE LINKS TO METABOLIC DISEASES

Adam William Barron
160212899

School of Computing
BSc Computer Science w/ Industrial Placement
7th May 2020

Supervisor
Dr Jaume Bacardit

Word Count
14993

Abstract

Through data mining, machine learning algorithms can be used to highlight and extract patterns that would otherwise never be recognised by humans. Accelerometer data is a prime example of data that cannot be read in its raw form and is collected in huge volumes. The UK Biobank study included hundreds of thousands of participants and a week's worth of data from wearables. This project details the process of preparing and converting a subset of this raw accelerometer data into meaningful categories or features to be used as input for a machine learning model. This dataset is prepared and cleaned in order to supply a variety of classifiers constructed in Python, with the aim of determining the health status of a given patient based off their corresponding accelerometer data. After designing, training, and validating this model using various machine learning techniques in order to ensure maximum confidence in its output, the results are visualised and evaluated to determine if there is a link between physical activity and having a metabolic disease. To explain any link found, the best features are extracted and analysed using existing literature. Although the classifiers do not score highly enough to suggest a significant relationship, the results, in combination with existing research, suggest a clear link between low physical activity as well as a poor sleep pattern and metabolic diseases.

Declaration

I declare that this dissertation represents my work except where otherwise stated.

Dedication and Acknowledgements

I would like to thank my supervisor, Dr Jaume Bacardit, for his guidance, patience, and expertise during the project.

I would also like to thank Fabiana Lombardi for being my constant source of inspiration and hope during the COVID-19 lockdown. Thank you for transforming an unprecedented and often scary situation completely beyond our control into an experience I will treasure and remember for a lifetime.

Table of Contents

Chapter 1 – Introduction 1
1.1 Motivation and Rationale 1
1.2 Project Aim and Objectives 1
1.3 Planning and Schedule 2
1.4 Software Development Methodology 3
1.5 Report Structure and Layout 3
Chapter 2 – Background Research 5
2.1 Feature Engineering Methods 5
2.1.1 Oxford Method 5
2.1.2 GGIR 5
2.1.3 Deep Learning Methods 6
2.2 Model Implementation Technologies 6
2.2.1 Python 6
2.3 Supervised Learning 7
2.4 Classifiers 7
2.4.1 Logistic Regression 8
2.4.2 Linear Support Vector Machine (Linear SVM) 9
2.4.3 Random Forest 10
2.4.4 Nearest Neighbour 12
2.5 Model Validation 13
2.5.1 Cross-validation 13
2.5.2 Grid Search 14
2.5.3 Recursive Feature Elimination (RFE) 14
2.6 Performance Metrics 15
2.6.1 ROC AUC 15
2.6.2 F1 Score 16
2.7 Class Imbalance 17
2.8 Analyses of the UK Biobank Study 17
2.9 Metabolic Diseases 17
Chapter 3 – Data Preparation 19
3.1 Choice of Feature Engineering Method 19
3.2 Data Manipulation 19
3.3 Class Imbalance and Subsetting 19
Chapter 4 – Model Implementation 21

4.1 Methodology 21
4.2 Overview 21
4.3 Data Cleaning 22
4.3.1 Missing (NaN) Values 22
4.3.2 Skew 23
4.4 Pipeline 24
4.5 Visualisation 25
4.6 Parameter Selection 25
4.6.1 Logistic Regression Classifier 25
4.6.2 Linear Support Vector Classifier (Linear SVC) 25
4.6.3 Random Forest Classifier 26
4.6.4 Nearest Neighbours Classifier 26
Chapter 5 – Results and Evaluation 27
5.1 Cross-Validation Results 27
5.1.1 Logistic Regression Classifier 27
5.1.2 Linear Support Vector Classifier (Linear SVC) 28
5.1.3 Random Forest Classifier 30
5.1.4 Nearest Neighbour Classifier 31
5.2 Performance Comparison 32
5.3 Feature Ranking 34
5.4 Clinical Significance 35
Chapter 6 – Conclusion 37
6.1 Project Aim and Objectives 37
6.2 Planning and Schedule 38
6.3 Future Work 39
References 41
Appendices 45
Appendix A – Project Gantt Chart 45
Original Project Plan 45
Appendix B – GGIR Script 46
BiobankDissertation.shell.GGIR 46
Appendix C – Java Helper Programs 47
DropEmptyClassLabels.java 47
GetAccelerometerIDs.java 48
FindCSVIntersection.java 50
GenerateSubset.java 52

SimplifyClassLabels.java 53
CollectSubset.java 54
Appendix D – Python Model 56
clean_data.py 56
model.py 57
Appendix E – Statistical Tables 61
Skewness of features 61
Feature Ranking 65

Table of Figures

Figure 1.1 - Work Plan 2
Figure 2.1 - GGIR Workflow (Migueles, 2019) 6
Figure 2.2 - Choosing the right estimator (scikit-learn documentation, 2019a) 8
Figure 2.3 - Sigmoid Activation Function (Subasi, 2019) 8
Figure 2.4 - Hyperplanes (Gandhi, 2017) 9
Figure 2.5 - Support Vector Margins (Gandhi, 2017) 10
Figure 2.6 - Random Forest Making a Prediction (Yiu, 2019) 11
Figure 2.7 - Feature Randomness (Yiu, 2019) 11
Figure 2.8 - Nearest Neighbours Classification (Naviani, 2018) 12
Figure 2.9 - K-Fold Visualisation (scikit-learn developers, 2019h) 13
Figure 2.10 – Stratified K-Fold Visualisation (scikit-learn developers, 2019h) 14
Figure 2.11 - ROC Curve (scikit-learn developers, 2019f) 15
Figure 2.12 - Multiclass ROC Curve (scikit-learn developers, 2019f) 16
Figure 4.1 - part5_personsummary_WW_L30M100V400_T5A5.csv 21
Figure 4.2 - A skewed feature (Pandas Profiling) 23
Figure 4.3 - A normally distributed feature (Pandas Profiling) 23
Figure 5.1 - Logistic Regression ROC Curve 28
Figure 5.2 - Linear SVC ROC Curve 29
Figure 5.3 - Random Forest ROC Curve 31
Figure 5.4 - Nearest Neighbour ROC Curve 32
Figure 5.5 - Classifier Accuracy Comparison 33
Figure 5.6 - Classifier Runtime Comparison 33
Figure 6.1 - Project Progress 38

Chapter 1 – Introduction

1.1 Motivation and Rationale

The UK Biobank is conducting an ongoing, large-scale clinical study. With conventional studies of this scale, it is necessary to rely on self-reporting and risk "self-reporting bias" (Althubaiti, 2016). This phenomenon causes data to not be fully accurate and therefore of less value to the researchers. When given the option, participants will often report what they think the person conducting the study wants to hear. Self-reporting is sometimes resolved by bringing participants in-house to conduct the study. However, this presents its own problems. Conducting a study at a hospital, for example, requires participants to arrange their own transport, and without spending large sums on accommodation, a study can only realistically last a few hours. This means that large sums of data, like that needed by the UK Biobank, cannot be gathered in this manner. In addition, people tend to behave differently when observed. This is known in psychology as the Hawthorne effect. Furthermore, it has been noted that this effect is almost impossible to control for or mitigate (McCambridge, 2014). To resolve these two key problems, Biobank study participants were asked to wear accelerometers. Through this method, data is guaranteed to be complete and representative of how participants live their everyday lives. They are not being directly observed, and the data is not self-reported. In addition, the relatively low cost of an accelerometer makes collecting large volumes of data possible. This study involved over 100,000 participants and 7 days of continuous wear of the accelerometers (Doherty, 2017). Subsets of this data are available via a paid application process. A research group at Newcastle University already has an application in progress, specifically looking at the data with regards to certain metabolic diseases. As a result, I will be effectively 'piggybacking' on to this application and investigating the same area for my project.

The large amount of data generated from this study is key to potentially significant discovery with data mining through machine learning. Data mining is defined as "the process of discovering patterns in data" (Witten, 2011). However, data mining as a technique has gained popularity recently due to the sheer increase in the amount of data being stored. It is estimated that the amount stored in the world's databases "doubles every 20 months" (Witten, 2011). This huge volume of data is practically impossible to analyse for a human; therefore, data mining offers the best potential to highlight a pattern or trend that might otherwise go undiscovered. This process, however, is not automatic, through carefully preparing the data to extract features I know will be useful for establishing a medical connection (feature engineering), I will be able to build a machine learning pipeline. This pipeline will then need to be carefully trained with an appropriately selected set of test data and validated, before being upscaled for the full data set (Bacardit, 2019). The key motivation for my project lies in the potential clinical significance held within this full data set. After analysing the results, I will be able to establish if there is any link between a person's physical activity and these metabolic diseases. If a link can be found, my results could be used to inform future health advice to patients and possibly prevent these diseases. This would, in turn, improve quality of life and save money that would otherwise be spent treating them.

1.2 Project Aim and Objectives

Aim: Assess techniques to extract features from accelerometer data and use data mining to determine if these features can be linked to the descriptors of metabolic diseases.

1) Identify and understand suitable feature engineering techniques for accelerometer data.

Accelerometer data has been processed repeatedly in many different scenarios. I will need to fully analyse and evaluate different techniques available and reach a decision on the best method for my project.

2) Establish validation protocols to determine if the descriptors are linked to diseases.

Every machine learning model needs carefully selected validation protocols so that the data scientist can have confidence in it. My project will be no different, before I can analyse the model on the full data set available, I will need to verify formally that it can reach correct conclusions on known data.

3) Analyse, by training and validating the machine learning model, whether the features extracted can establish a link to the diseases.

After I have a machine learning model I am confident in, I can use the full data set as input. By analysing the output from these experiments, I will need to reach a conclusion supported by the data on whether there is a link between lifestyle choices expressed in the input and having, or being at risk of, metabolic diseases.

4) Interpret if and why the best features identified hold clinical significance.

If a clear link is established between the features extracted and metabolic diseases, I will need to research expert medical knowledge to produce a concrete explanation behind the causation (or lack thereof) behind any correlation.

1.3 Planning and Schedule

Each of the objectives in the last section follows on from the previous and must be completed in the order presented to complete the project. As a result, a thorough plan with sensible contingencies was vital to ensure the project was delivered on time. An illustration of this plan can be seen below.

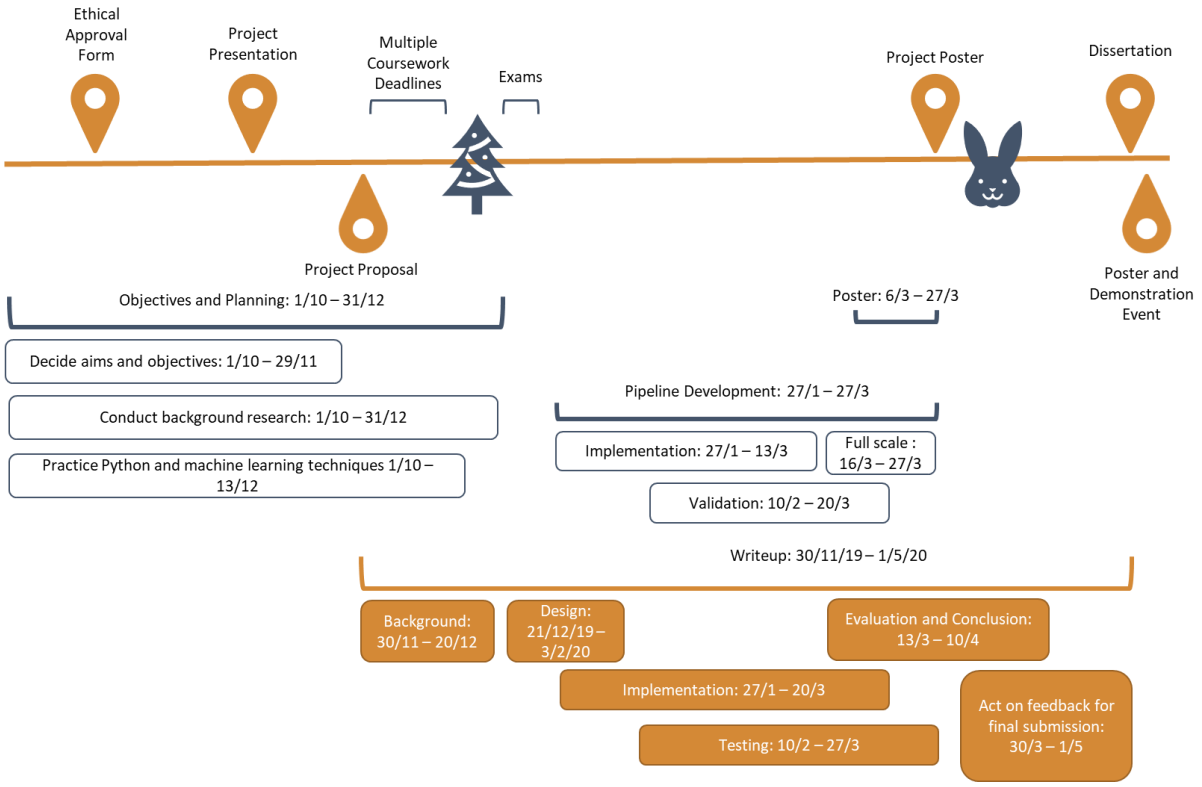


Figure 1.1 - Work Plan

This gives an overview of my plan for both semesters, set out against hard deadlines. When allocating time for each section, I have tried to include around one week on top of what I think will be necessary, as a contingency plan. All deadlines will be influenced by other coursework, and therefore I felt that it would be necessary to adapt and include flexibility from the start to prepare for this. A full Gantt chart

documenting this plan can be found in appendix A. In addition, I added all these tasks to my project Trello board, to keep track of progress. This Kanban board was shared with my supervisor.

1.4 Software Development Methodology

I selected an agile methodology for the software development section of this project, I felt that this lent itself better to working closely with my supervisor and offered maximum flexibility to tweak the machine learning model based on initial results to ensure I could have maximum confidence in it, as mentioned in my second objective. This methodology is reflected in my plan; the design, implementation, and validation of the pipeline overlap, as do the corresponding sections of the write-up. Throughout development, progress on my write up continued in parallel, documenting the design, implementation, and testing as it progressed. Once the validation section began, I was looking at validation protocols for the pipeline. Implementation continued alongside, working iteratively to ensure maximum confidence in the model. All source code produced during the project is stored in a private Github repository. This also allowed me to easily track the history of my development and rollback code if needed during the project. I was additionally protected against hardware failure, as I could be confident that a backup was available. Due to the confidential nature of the Biobank dataset, and it only being accessible via an application process, the data itself nor features extracted from it could be published on Github.

1.5 Report Structure and Layout

In chapter two, this report discusses the background reading I completed and the most relevant findings from this process. These findings will help outline the existing work in the field, i.e., current research surrounding links between physical activity and metabolic diseases as well as existing analyses of the UK Biobank dataset. A short section will also be dedicated to explanations of the most relevant machine learning terms and concepts for the project. In addition, I briefly explain the key technologies used during this project to prepare the dataset and construct the machine learning model used to analyse it. Following this, the project can be thought of as taking two parts. First, in chapter three, I explain the steps I took to prepare and manipulate the dataset to produce a valid input for the model, elaborating on my thought process and why key decisions were taken. Secondly, in chapter four, I go on to discuss the process of creating the model itself, the key technical choices I made during development and justify why I feel that I can be highly confident in its output. After this, chapter five presents and critically evaluates and analyses said output, discussing potential conclusions that could be made from the results of the data mining and exploring potential clinical significance held within these conclusions. Finally, chapter six closes the document with a reflection on the project. I discuss what went well, whether the project was a success relative to the original aims and objectives, what I would do differently should I attempt this project again, and what additional work could be completed in future to extend it. A full bibliography in addition to a range of appendices can be found at the end of the document.

Chapter 2 – Background Research

This section will explain in detail the key elements of data science necessary to understand the methodology behind this project as well as existing biological research relevant to accelerometer-based clinical studies and potential causes of various metabolic diseases.

2.1 Feature Engineering Methods

As briefly discussed earlier, data mining is the process of discovering patterns in data and is especially useful for large datasets like the one provided by the UK Biobank. However, in its raw form, the UK Biobank dataset is simply a selection of accelerometer readings that need to be analysed and translated before they can be used as input for a machine learning model. This is essentially the art of feature engineering. (Witten, 2011) Specific to the Biobank data, each patient's data is stored in its raw form as a CWA file. These files contain a series of readings made up of a timestamp and values from the x, y, and z axes of the accelerometer. These readings need to be converted into a 'human-readable' form such as time spent sleeping, time spent doing physical activity. I considered two different methods specifically designed for accelerometer data during this project, described below. The identification and consideration of these two methods was essential to the completion of the first project objective.

2.1.1 Oxford Method

The Big Data Institute at Oxford University developed a method for measuring physical activity from the UK Biobank dataset. (Doherty, 2018) It is capable of classifying accelerometer data into categories for analysis such as sleeping, sedentary, or physical activity. This tool works can be incorporated into custom python scripts and can be used for visualising data. The data is first calibrated according to local gravity, and values exceeding the sensor's dynamic range are flagged. Valid data is then resampled to 100Hz to prevent differential effects from slightly different frequencies during activity classification. The x, y and z axis values are combined to produce a norm acceleration value before machine noise and gravity is removed from the readings. These readings are then analysed to classify activity using a two-stage machine learning model consisting of balanced random forests and hidden markov models. A csv file is then generated for each participant. (Willets, 2018; Doherty, 2017)

2.1.2 GGIR

GGIR is an R-package specifically designed to process raw multi-day accelerometer data for physical data research, where raw refers to data expressed as acceleration values. These signals are processed to detect abnormally high values, periods of non-wear and calculate average dynamic acceleration using a range of metrics. This information is then used to describe the data per individual or per day of measurement. GGIR is notable for the large amounts of optional parameters that can be specified, allowing a large amount of flexibility in the analysis produced. GGIR runs in five parts, summarised below. (Migueles, 2019; van Hees, 2013)

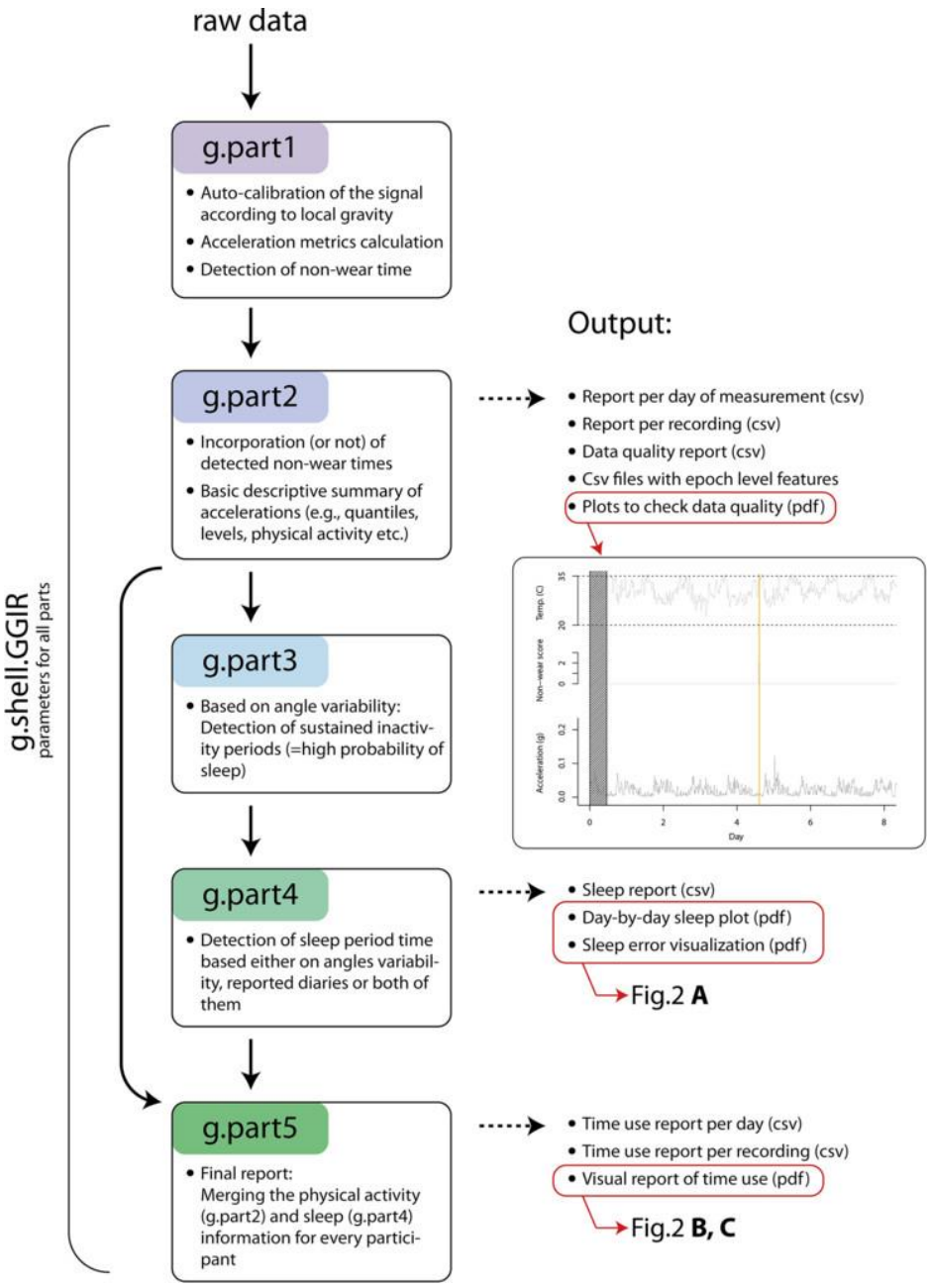


Figure 2.1 - GGIR Workflow (Migueles, 2019)

2.1.3 Deep Learning Methods

It is worth noting the process of feature engineering can be avoided entirely using deep learning. Such methods have their own implicit feature engineering implementation; however, they require extremely large amounts of data to learn from (Bacardit, 2020c). As a result, they are not analysed in this project.

2.2 Model Implementation Technologies

2.2.1 Python

Python is the most used language for machine learning. This is largely for two reasons. First, due to it being a high-level language, it allows the user to focus more on addressing the problem than the syntax required to implement the solution. In addition, Python has an excellent ecosystem of libraries and frameworks designed for machine learning and artificial intelligence. These two things combined

have created a wide community around using the language for these problems. (Beklemysheva, 2020) This means a great deal of support is available, making it easier for me to complete my implementation. Several of the most used libraries are also open source and free, and therefore more likely to be well documented and supported. As a result, Python is the ideal language for this problem and will make my implementation as effective as possible.

2.2.1.1 Scikit-learn

Scikit-learn is one such machine learning library for Python, as discussed above. It is a library that integrates a wide range of machine learning algorithms that can be used for both supervised and unsupervised learning. All these algorithms are then able to be integrated into a Python application. (Pedregosa, 2011) Using scikit-learn, I can test the effectiveness of numerous machine learning algorithms on my data set and evaluate the results to create the most effective model possible. Scikit-learn is also specifically designed to be accessible to non-experts, so I would not have to spend excessive time learning to develop with scikit-learn or grappling with a complex API. Its large community and thorough documentation would make solving problems as easy as possible in my implementation.

2.2.1.2 Pandas Profiling

Pandas Profiling is a python library designed to perform exploratory data analysis. This analysis is presented in a report generated automatically by the package. (Brugman, 2020) Specific to this project, Pandas Profiling can be used to investigate potential problems in the features extracted that could impact the performance of a machine learning model. Certain features with missing values or high standard deviation can be highlighted automatically, allowing appropriate action to be taken. Identifying such features requiring attention manually would be an almost impossible task for such a large dataset.

2.3 Supervised Learning

This project is concerned primarily with using attributes, as extracted by feature engineering as described above, and identifying patterns between them and whether a participant has some type of metabolic disease. In the field of machine learning, this is known as a classification problem, solved using classification learning. A learning scheme is presented with a set of already classified examples from which it is expected to learn a way of classifying unseen examples. Classification learning is often referred to as supervised learning. This is because the system operates ‘under supervision’ by being given the actual outcome of each training example. (Witten 2011)

Given some input and output variables, the model attempts to learn the mapping between these inputs and their output. The goal is to “approximate the mapping function so well that when you have new input data that you can predict the output variables for that data.” (Shuka, 2019; Bacardit, 2019). These input variables will take the form of the features extracted for each participant using feature engineering. The output variable is what is referred to as the participant’s class label. A class label is a simple number corresponding to a specific ID indicating what combination of metabolic diseases the participant has. For example, class 0 is a healthy person, while classes 1-3 represent various unhealthy participants. These labels are explained in more detail in section 3.3.

2.4 Classifiers

To best address the problem of extracting meaningful data (data mining) from the biobank study, this project will analyse and evaluate the performance of multiple different classifiers. A classifier is simply a machine learning model designed to solve a classification problem (Raschka, 2020). A wide variety of estimators are provided by the scikit-learn library, to narrow down the most appropriate estimators

for a given project a 'cheat sheet' is provided. The section of the diagram relevant to classification problems is shown below.

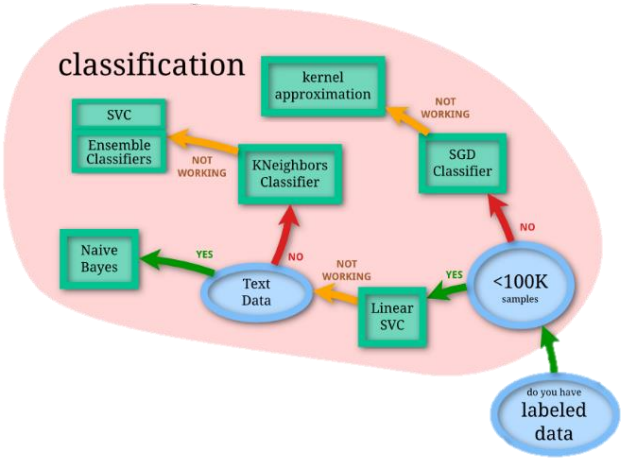


Figure 2.2 - Choosing the right estimator (scikit-learn documentation, 2019a)

Although the biobank is larger than 100,000 samples, the scope of this project only covers a smaller subset. Additionally, all features extracted from accelerometer data will be numeric in nature (time spent in certain states etc.). Therefore, based on this diagram, linear support vector (Linear SVC) and nearest neighbour (KNeighbors Classifier) were selected. In addition to these, due to their ubiquity, random forest and logistic regression were also selected. These classifiers are explained in more detail below.

2.4.1 Logistic Regression

The logistic regression classifier is one of the most popular methods of classification. It is especially famous in the finance industry, where it is used to predict if someone will default on debt. It is unique in the sense that the name indicates it is designed for regression problems rather than classification, however, this estimator uses regression equations to produce binary output. It is worth noting that, despite producing a binary output, this estimator can still be used for multi-class problems by producing one classifier for each class available, comparing a given class with all others. The logistic regression classifier uses a sigmoidal activation function, seen below.

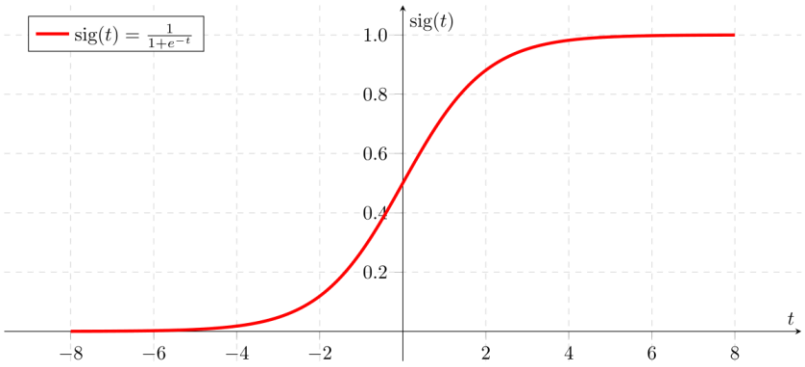


Figure 2.3 - Sigmoid Activation Function (Subasi, 2019)

Such an activation function is useful when predicting some class. Due to being concerned with probability, which is conventionally measured between 0 and 1, it is necessary to compress real-world generated data into this range. The continuous output values of some linear function are compressed

using the sigmoidal model and assigned a class label based on its position, anything above 0.5 is given one label, and a data point below 0.5 is assigned the other class. (Subasi, 2019)

Two different formulations are offered for logistic regression. The optimal choice is based on whether the number of samples outnumber the features in the dataset (scikit-learn developers, 2019d). Due to the number of samples in the biobank subset exceeding the number of features, the classifier is set to solve the primal optimisation problem rather than its dual. Numerous different solvers can be used when fitting a logistic regression classifier, however, two of these (sag and saga) are designed for larger datasets like the biobank subset (scikit-learn user guide, 2019c). In addition to performing well on large datasets, sag has been found to converge faster for some high dimensional data. The sag solver only supports l2 (penalised logistic regression), the cost function for which is given below.

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log\left(\exp\left(-y_i (X_i^T w + c)\right) + 1\right)$$

The saga solver supports Elastic-Net regularisation, which is a combination of l2 and l1 (regularised logistic regression), the cost function for Elastic-Net is given below.

$$\min_{w,c} \frac{1-\rho}{2} w^T w + \rho \parallel w \parallel_1 + C \sum_{i=1}^{n} \log\left(\exp\left(-y_i (X_i^T w + c)\right) + 1\right)$$

where ρ controls the strength of l1 regularisation vs. l2 regularisation (scikit-learn user guide, 2019b).

2.4.2 Linear Support Vector Machine (Linear SVM)

The basic objective of a linear support vector is to find the optimal 'hyperplane' that separates the data according to its class labels. This hyperplane is created in n-dimensional space, where n is the number of features in the data set. An example of this in 2D and 3D space is seen below.

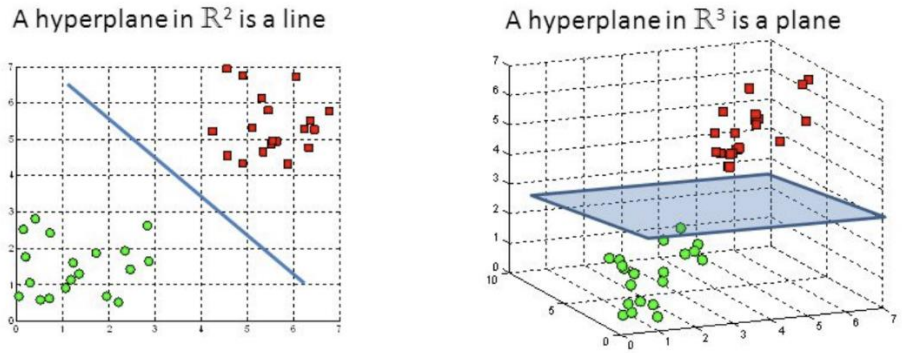


Figure 2.4 - Hyperplanes (Gandhi, 2017)

As can be seen from the diagrams, there are multiple possible locations for the hyperplane. The optimal position is chosen based on support vectors. Support vectors are data points that are closest to the hyperplane and therefore influence its exact position and orientation.

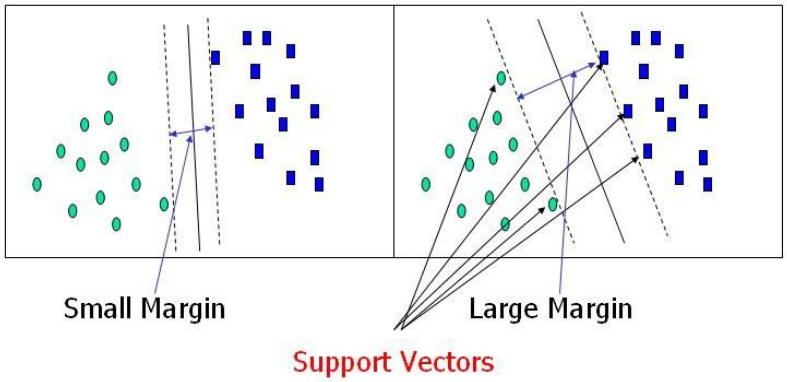


Figure 2.5 - Support Vector Margins (Gandhi, 2017)

A linear support vector tries to achieve the largest margin (distance between the data points of both classes) possible based on these support vectors to maximise the chance that future data points are classified correctly. (Gandhi, 2017)

It is more difficult to visualise this process in more than three dimensions, as is the case with the biobank dataset, which contains hundreds of features. With many models, this complexity translates to poorer model performance, a phenomenon that is known as the 'curse of dimensionality'. Most notably, many models tend to suffer from overfitting when dimensionality is high. This problem can be mitigated somewhat by feature selection algorithms, which are used in this project. However, SVMs are less vulnerable to this than some other models due to their use of regularisation parameters to prevent this issue. (Cawley, 2007)

Like logistic regression, linear SVCs can also choose whether to solve the dual or primal optimisation problem. For the same reason as with the logistic regression classifier, this classifier was also set to solve the primal problem, given below: (scikit-learn user guide, 2019b)

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, \ldots, n$$

2.4.3 Random Forest

Random forest classifiers are summarised by Yiu [2019], they rely on a committee made up of several individual trees to make a classification decision. Each tree is constructed with elements of randomness to ensure they are uncorrelated, resulting in a group prediction more accurate than any individual.

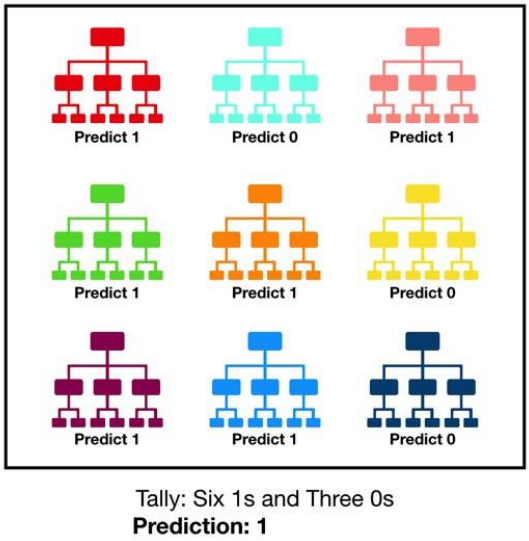


Figure 2.6 - Random Forest Making a Prediction (Yiu, 2019)

A single member of a random forest is a decision tree. A decision tree classifies a set of observations by dividing them based on some feature, like a flowchart, to create as many distinct groups as possible. Each of the individual trees in a random forest makes a prediction for what a data point should be classified as, then the actual prediction made by the model is chosen via a simple majority vote. The use of a majority vote allows the model to minimise individual errors made by trees. By doing this, the group of trees is more likely to be correct. It is worth noting that unlike a conventional random forest, the implementation used by scikit-learn averages the probabilistic prediction of each individual tree, rather than having each tree vote for a single class.

To achieve good results using this method, it is important that the predictions made by trees are uncorrelated, as if the same errors are made by a majority of trees the overall prediction will also be wrong. To achieve this uncorrelated affect, the model relies on feature randomness and bagging. Feature randomness highlights the key difference between a single decision tree and a random forest.

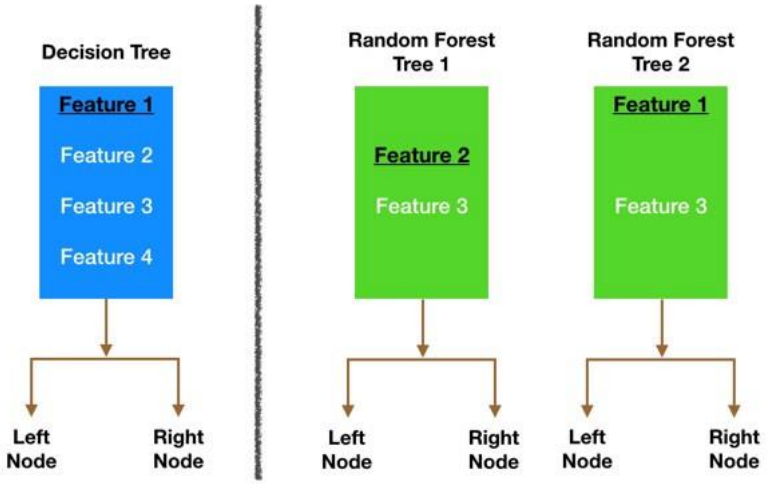


Figure 2.7 - Feature Randomness (Yiu, 2019)

A standalone decision tree has access to all available features to separate the data, but with a random forest, each tree can only use a random subset of the available features. As a result, trees in the forest have increased variation resulting in lower correlation between trees. It makes sense that giving each

tree a subset of the training data may also help reduce correlation, however, a smaller training set can result in poor model fitting. To facilitate variation without compromising on the volume of data, bagging is used. With this method, each tree randomly chooses n values from a dataset of size n with replacement. This results in a training set for each tree that is the same size as the original, but slightly different in content to maximise variation and further facilitate low correlation between trees. (Yiu, 2019)

2.4.4 Nearest Neighbour

Nearest neighbour classification is another common and arguably the simplest of the popular classification methods. The nearest neighbour method is unique amongst the classifiers used for this project in that it is considered a 'lazy learning' algorithm. This means that no data is used to train a model and instead, instances of the training data are simply stored. To classify a new data instance, the classification of the new data point's nearest neighbours is looked at to determine the label for the new data. Like the linear support vector method, the data is plotted on an n-dimensional graph, where n is the number of features in the dataset. The nearest neighbour method can also be used for multi-class problems. A simple example of the process on a binary classification problem using three nearest neighbours is shown below.

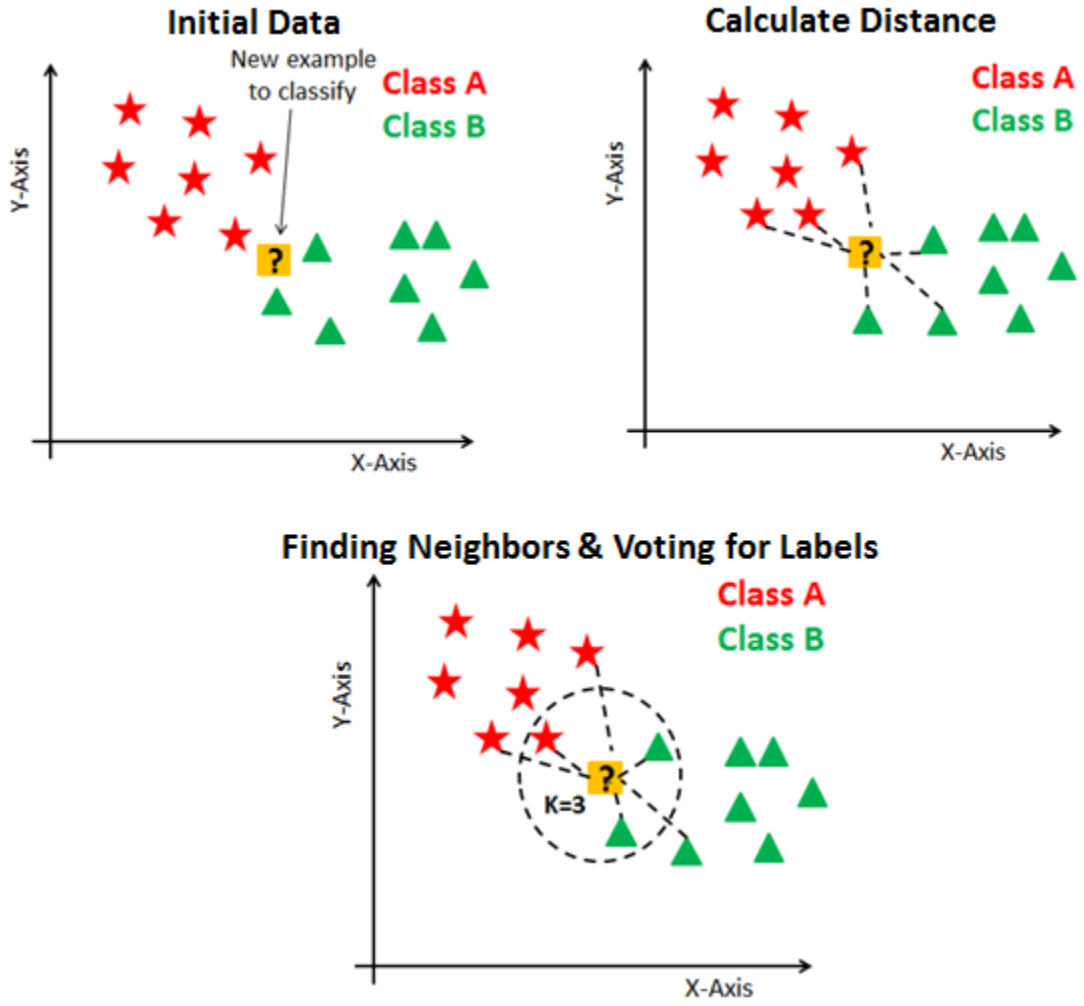


Figure 2.8 - Nearest Neighbours Classification (Naviani, 2018)

The nearest neighbour method grows more complicated with more features present. Unlike SVMs, the nearest neighbour method of classification is very susceptible to the 'curse of dimensionality', the

algorithm performs much better with a lower number of features and is vulnerable to overfitting (discussed in the next section) when many features are present. Due to the biobank dataset having many features, this is a concern, however, this can be mitigated with different methods of calculating distance between neighbours or using a reduced feature set. In addition, the problem is compounded when the number of features exceeds the number of samples (Naviani, 2018), an issue not present when analysing the Biobank dataset.

2.5 Model Validation

One of the most important aspects of a machine learning model is knowing that one can have confidence in the results it produces. This is the primary concern of the second project objective. To test a model, the dataset must be partitioned into both a training (used to create the model) and a test set (used to validate the model). It is important that these two sets do not overlap in any way. This is because, when testing the model, its performance needs to be evaluated against previously unseen data points. (Bacardit, 2020a) The performance of the model can then be computed using various metrics against both datasets. If the metric is considerably higher for the training set than for the test set, overfitting has occurred. Overfitting is defined as "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably." (OxfordDictionaries, 2020) Simply, in machine learning terms, the method has modelled the training set rather than the problem itself.

2.5.1 Cross-validation

A commonly used and robust estimator for machine learning models is cross-validation, or more commonly, k-fold cross-validation, where k is some integer. This validation strategy divides the dataset into k strata or folds. The performance of the model is computed against some metrics a total of k times, with each fold taking turns to be the test set, while all other folds take the role of a combined training set. The overall performance of the model is then calculated by taking an average of the performance using each of the k test and training sets. This is computationally more expensive than a simple partition of data into a test and training set but is much more likely to give an accurate report of the model's performance. (Brownlee, 2018)

A visualisation of k-fold cross-validation, with the test set highlighted red and the training set in blue, can be seen below.

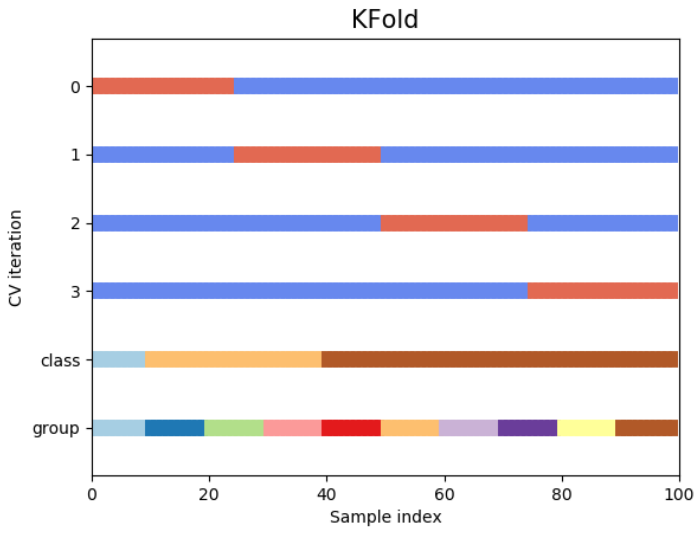


Figure 2.9 - K-Fold Visualisation (scikit-learn developers, 2019h)

It is important to note that standard k-fold cross-validation does not consider the distribution of each class across the training and test sets. As a result, it is often beneficial to utilise stratified k-fold cross-validation instead. This variation simply ensures that each fold has the same class distribution as the overall dataset (Bacardit, 2020a), and is therefore more representative ensuring a more reliable performance estimation. A visualisation of stratified k-fold cross-validation considering class labels can be seen below.

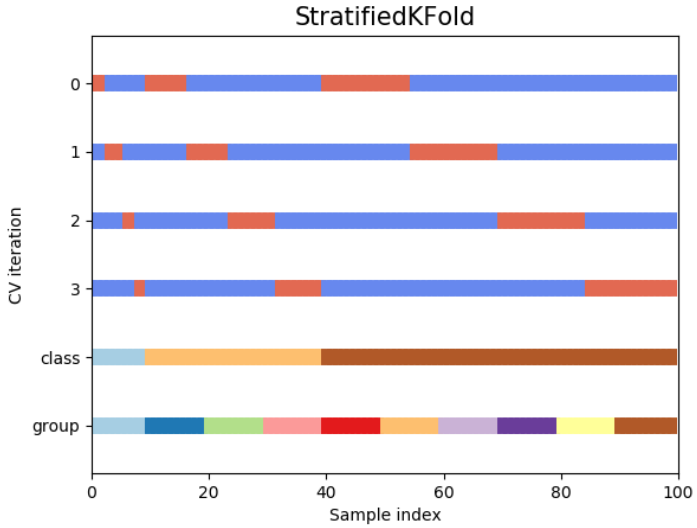


Figure 2.10 – Stratified K-Fold Visualisation (scikit-learn developers, 2019h)

2.5.2 Grid Search

With many machine learning models, there are a selection of parameters that are not learned directly during training. These parameters are known as hyper-parameters. It is important to have confidence that these parameters are set to optimal values to maximise the model's performance. The hyper-parameters of a model can be tuned using grid search. This is an exhaustive search of a given range of parameters, spread across one or more grids as appropriate. Every combination of the specified parameters and their values is tested. The performance of the model is computed using some metric against each of these combinations. Grid search can be combined with k-fold cross-validation, described above, to determine the optimal parameters for each fold. The most selected parameters can then be used for maximum confidence in the model's performance. This can be thought of as nested cross-validation. Each fold is split up again into training and test sets to ensure that the partitions used for performance estimation are not reused for determining the optimal parameters for the model.

2.5.3 Recursive Feature Elimination (RFE)

Once the data for a model is processed, each attribute or column is known as a feature. Some of the features extracted through feature engineering may not be useful and may in fact make the model perform worse if it is included in the data. (scikit-learn documentation, 2019b) For this reason, some form of feature ranking and selection is useful. As the name suggests, RFE is the method of recursively considering smaller sets of features until the desired number of features is found. The model is first trained on the full set of features and the importance of each feature for making correct classifications is measured. The features are ranked using this importance metric and the least useful are removed first during the RFE process. This process then repeats until the desired number of features is reached. This technique allows the filtering out of irrelevant or redundant features, and can help reduce training times, simplify the model, and reduce overfitting. (Radečić, 2019) Additionally, the ranking of the

features can be extracted based on the training process of a classifier to consider why the best features hold significance. This is the key focus of this project's final objective.

2.6 Performance Metrics

There are different metrics for measuring the performance of a model on a classification problem. The two most useful metrics that this project is concerned with are the area under the receiver operating characteristic curve (ROC AUC) from prediction scores, and the weighted f1 score.

2.6.1 ROC AUC

In its simplest terms, ROC AUC "tells us how much the model is capable of distinguishing between classes". (Narkhede, 2018) In terms of this project, this metric can tell us how good the model is at correctly classifying different patients according to which, if any, metabolic diseases they have. The larger the area under the curve, the better the model is at distinguishing. It is effectively a measure of the probability a correct classification choice is made for new data, therefore, if the area is 1, the model makes the perfect prediction every time. At 0.5, the model is basically random and has no real separation capacity. If the value were 0, the model would be inverting the results, i.e. giving the incorrect answer each time. This curve is created by plotting the true positive rate of the model against the false positive rate at various threshold sensitivity and specificity settings. Therefore, each individual point on the curve represents a possible trade-off between sensitivity and specificity.

An example ROC curve can be seen below:

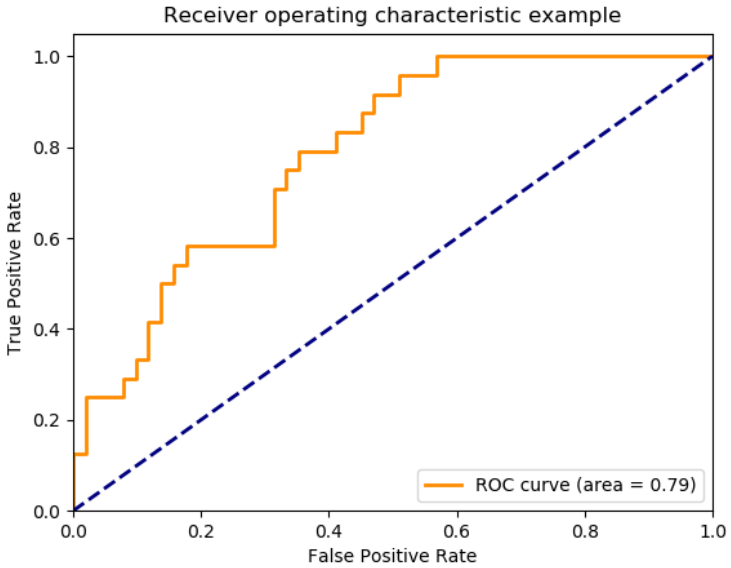


Figure 2.11 - ROC Curve (scikit-learn developers, 2019f)

In the case of the biobank data set, there are multiple classes. As a result of multiple options for a model to potentially classify data as, the method for using a ROC Curve is slightly different. Each class needs to have an individual curve for its true positives and false positives, and then an average can be calculated. An example of a multiclass ROC curve can be seen below:

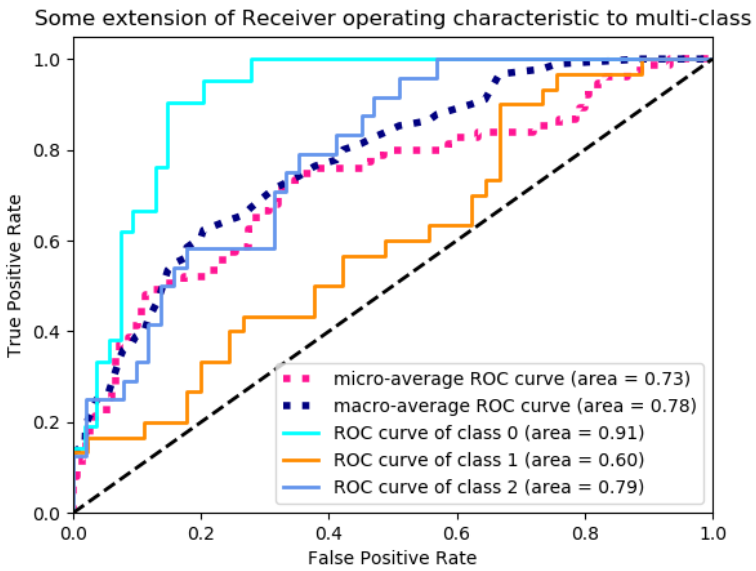


Figure 2.12 - Multiclass ROC Curve (scikit-learn developers, 2019f)

2.6.2 F1 Score

The second key metric for classification problems is the F1 score. This is the harmonic mean of the model's precision and recall and quantifies the balance between the two metrics. The precision of a model can be thought of as how exact the classifier is, whereas the recall is a measure of how complete the classifier is. (Brownlee, 2019) It is often useful to balance the two when evaluating the performance of a model, as a low precision but high recall can indicate a large number of false positives, whereas a high precision and low recall can indicate a large number of false negatives. As a result, precision and recall on their own are not sufficient metrics for establishing the performance of a machine learning model. The F1 score of a model is calculated as follows, as explained by Bacardit [2020a]:

The precision of a model is given as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Where TP is the number of true positives and FP is the number of false positives.

The recall of a model is given as follows:

$$\text{Recall} = \frac{TP}{P}$$

Where P is the number of examples of the positive class.

Therefore, the F1 score is given as follows:

$$\text{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

In this project, the weighted f1 average is used. This method accounts for any potential label imbalance when calculating the score by weighting a label's average according to the number of true instances of that label. (scikit-learn developers, 2019a)

2.7 Class Imbalance

A well-documented and widely understood problem in classification is working with imbalanced datasets, or the class imbalance problem. This occurs when there is a significant difference between the number of examples for different classes of the problem. The crux of this issue is that standard models tend to be biased towards the higher frequency classes. This leads to a higher misclassification rate for classes that are less frequent when training the model. (López, 2013) There are however three major ways of attempting to correct this issue.

The first of these is during the sampling of the data. The training instances supplied to the model can be changed so that the classes are more evenly distributed. This should allow classifiers to perform in a manner like usual classification. (Batista, 2004) In the case of the biobank data set, due to its large size, this would be best accomplished by grouping together all non-healthy participants into one class. Another viable option for this project would be to utilise cost-sensitive learning. This would aim to correct errors at an algorithmic level by specifying to the model the class weights present in the training set. Using this, errors in the minority classes can be penalised more, helping reduce the misclassification rate for these classes. (Zadrozny, 2003) A third option involves modifying the base learning models used so they are more attuned to the class imbalance issues. (Zadrozny, 2003) However, modifying base learning methods in this manner is beyond the scope of this project, and would be the most time consuming.

2.8 Analyses of the UK Biobank Study

Several studies have already been carried out using the UK Biobank data. An existing study of the same data used for this project, Willets [2018] was a very useful starting point for identifying potential research techniques and establishing what existing useful conclusions have already been extracted from the data. Willets' project is concerned with sleep rather than metabolic diseases, but I felt some of the data analysis and preparation methods used were transferrable to my own project. As a result, I found it useful to critically evaluate the machine learning techniques and conclusions found within the report. This study trained a balanced random forest for classifying and recognising activities, this model is now distributed by the Biobank for use in other projects as the Oxford method, as discussed in section 2.1.1.

Outside of the domain of accelerometer data, Bycroft [2018] discussed the potential of the biobank data for studying the genomes of the participants. The authors conclude that even the interim release of the data used for this study has enabled a huge amount of investigation into links between disease and various lifestyle factors. This gives confidence that there is clinical significance within the data, which can, in turn, be extracted through data mining or other techniques. This paper also indicates the large amount of biological data included from the participants. This justifies that medical conclusions could be reached based on patterns extracted by my machine learning model, even if they are beyond the scope of my expertise. Therefore, the data I present as the result of this project could then be utilised by another to draw more thorough conclusions in the field of medical science.

2.9 Metabolic Diseases

Due to my background as a computer scientist, and not as a biologist, I am hesitant to draw any medical claims based on the results of my project unless they can be very well supported with other literature. However, to evaluate the success of my model and begin to draw conclusions as to the

significance of any patterns discovered, it was necessary to familiarise myself with current research regarding metabolic diseases and any link to lifestyle. For example, Bankoski [2011] concludes that there is a "borderline significant" link between intensity during sedentary time and metabolic syndrome. This study was similar in that it relied on accelerometer data and many participants. However, this study was limited to those over 60. The biobank dataset I am analysing is more diverse in terms of its participants, so I may be able to give confidence to a conclusion that encompasses more of the population. Additionally, in another accelerometer study, Sisson [2010] finds that as the daily step count of the study participants increased, the prevalence of metabolic diseases decreased. This seems to indicate that there is some link between physical activity and the likelihood of having a metabolic disease. Specifically, patients were more at risk when activity levels are low. It has also been well documented that sleep curtailment contributes to a range of diseases, including metabolic syndrome. (Cappuccio, 2017; Kim, 2018) Sleeping patterns are another element contained within accelerometer datasets that can be analysed through data mining.

Chapter 3 – Data Preparation

3.1 Choice of Feature Engineering Method

As outlined in section 2.1, there are two well-established methods to extract features from the Biobank dataset. GGIR, an R package, and the Oxford method, a Python script. I decided to use GGIR for my feature engineering over the Oxford method for a few reasons. Most importantly, R scripts can be set to run in parallel across multiple cores easily, this was essential to ensure that features were extracted in a reasonable time frame to allow the rest of the project to continue. Also, when setting up an R script for GGIR, it is possible to specify an entire directory for analysis to be run on all CWA files within, meaning I would only need to run the R script once. This is very different from the Oxford method, which would require multiple runs (one per CWA file) to be scripted. Additionally, GGIR offers a large range of other optional parameters that can be specified to better fine-tune the analysis of the data and its output. However, due to being more general (GGIR is designed for accelerometer data in general, not just the Biobank set), it was more complicated to set up and run than the Oxford method. GGIR requires a script to be written, specifying these parameters. However, as an R package designed to be used widely within the data science community, there was a wide range of support available should I encounter any problems. The settings used for the main GGIR run of this project can be found in appendix B.

3.2 Data Manipulation

When stored locally in its raw form, the UK Biobank data is around 9TB when compressed. Due to the sheer volume of data prior to any processing, it would have been impossible to manage the data on my own machine or a university machine. As a result of this, all data preparation was done remotely on the university's 'cruncher3' machine. As well as having a large amount of storage to hold the data, this workstation has powerful GPUs to enabling the performance of simulations and manipulation of the data in a reasonable time frame. Due to the large scale of the data, all manipulations were accomplished using a range of Java programs, the source code for which can be found in appendix C of this document. Java was used for this section of the project simply because it is the language I am most familiar with and I therefore felt it would be the easiest way to complete the task as fast as possible.

The key attribute of each patient for this problem is whether they have the various diseases this project is concerned with. However, as the Biobank dataset is quite general, not all participants have any sort of class label associated with them. As a result of this, they cannot be used for supervised learning. (Bacardit, 2019) By dropping all participants with no class label (DropEmptyClassLabels.java) the dataset was effectively cut in half, reducing the dataset from 502,645 records to 293,270. Additionally, it was important to note that not all the patients with a class label would necessarily have an associated accelerometer data file. Due to this, it was necessary to search through the data, extracting the ID numbers of those patients with accelerometer files (GetAccelerometerIDs.java). In total, there are 103,713 accelerometer files in the biobank dataset. To identify usable patients for this project, the overlap between these two sets was identified (FindCSVIntersection.java). The intersection between those with class labels and those with accelerometer data yielded 57,782 records. In total, out of the 502,645 starting records, 57,782 of those would be usable for this project.

3.3 Class Imbalance and Subsetting

In preparation for training my model, the class labels were simplified somewhat. The biobank data assigns one of four numerical classes to each participant. Class 0 represents a healthy patient, class 1 indicates cardio-vascular disease (CVD), class 2 indicates type-2 diabetes and class 3 is CVD as well as

diabetes. A breakdown of their distribution among all the patients with accelerometer data can be seen below.

Class	Frequency
0	28,962
1	26,070
2	789
3	1961
Total	**57,782**

It can be seen from the table that classes 2 and 3 are considerably under-represented in the data compared to classes 0 and 1. This leads to the class imbalance problem, explained in section 2.7. Although this could have been resolved at an algorithmic level in my model's python implementation using cost-sensitive learning, I felt that in the interests of time it would be simplest to create a binary version of the class labels (SimplifyClassLabels.java) by grouping the class labels that represent an unhealthy participant into class 1. The new distribution of class labels after this simplification can be seen below.

Class	Frequency
0	28,962
1	28,820
Total	**57,782**

This table shows that the labels are now almost exactly even, and there are still useful patterns that could potentially be learned through data mining, they are just less specific. Due to a blanket unhealthy class, any conclusions reached cannot be tied to specific metabolic diseases.

Due to the time constraints of the project, and the large amounts of time and processing power required to process the raw data. I was unable to perform analysis on the full dataset. To get a sense for how long running GGIR would take, I performed a test run on a single folder of accelerometer data, containing 1,000 files. This run used the same script that I would eventually use for the main feature engineering. 1,000 files took around 9 hours to process. Extrapolating this estimate to all 58,000 files gives a runtime for GGIR of 522 hours or nearly 22 days. Additionally, as mentioned earlier, this data processing was taking place on a shared high-power machine, therefore as well as taking a long time I would effectively be blocking other users from completing their own work. As a result of lacking time, I decided to instead select a subset of around 10% of the records. This subset was selected using purely random sampling; I assigned each record a probability of being selected of 0.1. (GenerateSubset.java) Due to the relatively even number of healthy and unhealthy patients once the class labels were grouped, I felt it was not necessary to complicate this process. This resulted in a subset of 5780 records. From this point, I needed to isolate the corresponding compressed accelerometer file for each of these participants. Once the appropriate file paths were identified and the files were copied to a separate directory (CollectSubset.java), they could then be unzipped, and the GGIR script could be run on them to produce the relevant output csv files.

Chapter 4 – Model Implementation

4.1 Methodology

Due to the large amount of time required to process the Biobank dataset, I decided to build my model In Python using one of the scikit-learn provided sample datasets instead. As mentioned above, the key attribute or class of the Biobank dataset is whether each patient has any of the various diseases relevant to the project. As these values are discrete, this dataset is a classification problem. (Bacardit, 2019) Therefore, by selecting a sample dataset that contained both known inputs and discrete outputs, I could build the experiment up using data that was processed in an identical manner to the real data. With this method, I would simply need to swap out the sample data set for the actual Biobank data once I was confident in my model. This allowed me to work on my model implementation while the data was being processed using GGIR on cruncher3. The provided dataset used during creation and to test the model was the breast cancer dataset. This dataset was chosen as, like the biobank dataset, it also a binary classification problem.

4.2 Overview

The python implementation for this project uses the simplified class labels and the output provided by GGIR following chapter 3, specifically, the csv file providing a breakdown of activity per patient. In this file, each patient is represented by a row, and each of the extracted features from GGIR is represented as a column. This data forms the training and tests used to create and validate the model. This dataset contained a total of 4630 samples (rows) and 215 features (columns). A small snapshot of this file can be seen below.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	filename	id_pla	startday	calendardate	Nvaliddays	Nvaliddays	Nvaliddays	Ndaysleep	Ncleaning	Nsleeplog_	Nacc_avail	acc_onset_	acc_wake_
2	1000180_9(	1000180	Friday	11/04/2014	7	2	5	0	0	0	6	23.893056	31.269213
3	1001785_9(	1001785	Friday	22/05/2015	7	2	5	0	0	0	7	22.615278	29.74623
4	1003945_9(	1003945	Friday	20/11/2015	5	2	3	0	0	0	5	24.310556	30.668056
5	1004203_9(	1004203	Wednesda	29/04/2015	5	2	3	0	0	0	5	22.554167	31.203889
6	1004674_9(	1004674	Wednesda	05/11/2014	7	2	5	0	0	0	7	24.34246	32.652976
7	1004755_9(	1004755	Saturday	09/05/2015	7	2	5	0	0	0	7	23.022024	28.940079
8	1006496_9(	1006496	Wednesda	03/09/2014	7	2	5	0	0	0	7	28.168056	33.987897
9	1006680_9(	1006680	Saturday	15/06/2013	6	2	4	0	0	0	6	23.943981	30.631019
10	1007249_9(	1007249	Saturday	29/03/2014	7	2	5	0	0	0	6	23.784491	31.734954
11	1007270_9(	1007270	Saturday	13/09/2014	7	2	5	0	0	0	7	22.753175	30.714087
12	1007387_9(	1007387	Wednesda	17/06/2015	7	2	5	0	0	0	6	24.765972	31.178935
13	1007432_9(	1007432	Wednesda	22/01/2014	6	2	4	0	0	0	6	23.575	32.221991
14	1008260_9(	1008260	Thursday	05/06/2014	7	2	5	0	0	0	7	22.571032	30.876587
15	1009284_9(	1009284	Thursday	02/07/2015	7	2	5	0	0	0	7	22.278571	30.732143
16	1009552_9(	1009552	Tuesday	26/08/2014	5	1	4	0	0	0	5	23.731389	29.165833
17	1009637_9(	1009637	Friday	06/12/2013	7	2	5	0	0	0	7	24.849603	31.7625
18	1010975_9(	1010975	Saturday	16/08/2014	7	2	5	0	0	0	7	22.298214	30.502976
19	1011101_9(	1011101	Saturday	05/07/2014	7	2	5	0	0	0	6	27.121759	31.913194
20	1011576_9(	1011576	Tuesday	11/06/2013	2	0	2	0	0	0	2	25.315972	32.900694

Figure 4.1 - part5_personsummary_WW_L30M100V400_T5A5.csv

The python implementation takes the form of two separate files, the source code for which can be found in appendix D of this document. First, clean_data.py further pre-processes and prepares the data for use. It includes various output statements and produces profile reports on the dataset using Pandas Profiling to direct decisions about further pre-processing. When this program concludes it exports a new, cleaned csv file for use with the model, this functionality was made separate as it would only need to be run once. The core of the model is provided by model.py, which is designed to be run multiple times. It can be seen from the code that multiple classifier declarations are commented out. This is because the model is only designed to fit one type of classifier at a time, but multiple different types of classifier were trained to compare their performance. Likewise, corresponding parameter grids used for grid search cross-validation are similarly commented out. By commenting and

uncommenting different classifiers and their parameter grids, the programmer can fit, and view the results produced by, different classifiers.

To ensure confidence in the model, all fitting and tuning is done using cross-validation. This experiment uses the standard five-fold cross validation. This is commonly regarded as a rule of thumb, as it gives 20% of the data for use as test data which is generally representative (Bacardit, 2019). As explained in section 2.5.1, 5-fold cross-validation will produce 5 distinct training and test sets through slicing the data in different ways. Due to there being only two almost exactly evenly weighed class labels following the simplification (unhealthy and healthy), stratified k-fold cross-validation was not used. A significant class imbalance in a fold was statistically extremely unlikely, so the additional computational expense was not worth it.

The cross-validation loop processes different estimators as a scikit-learn pipeline and performs data manipulation, feature selection, and fitting of a classifier according to each data fold. After a classifier is trained, its predictive power against the fold's corresponding test set is measured. To select parameters during the grid search and measure model performance, the cross-validation loop runs separately against two metrics, the weighted f1 score and roc_auc. These metrics are explained in more detail in section 2.5. As well as giving different measures of performance, the use of different metrics means that different optimal parameter selections may be found for the same fold.

As the cross-validation loop runs, the scores generated by these metrics in addition to the runtime for training each classifier and the features selected by RFE are recorded. These scores and runtimes are then averaged and outputted in addition to a graph showing the ROC curve for the classifier so they can be critically compared and evaluated. These results for each metric were essential to facilitate the discussion necessary to complete my third objective. The recording of which features were selected was used to create feature ranking to select the most significant features for investigation in the fourth objective.

4.3 Data Cleaning

Before the dataset could be used to train the model, some additional pre-processing needed to be performed. First, the selection of features for each patient in the subset and the corresponding labels were stored in separate csv files. As a result of this, it was necessary to perform a join on both csv files to ensure that the corresponding class label for each patient was included. By inspection, the dataset contained some columns (features) that were basic metadata for each patient and therefore irrelevant to the health conditions of a patient and/or were unique values per patient. 'filename' and 'id_pla' represent the name of the file the features were extracted from and the ID number of the patient, respectively. The ID number was a necessary field for merging with the class labels but afterwards could be removed. 'startday' and 'calendardate' represented the weekday and calendar date the accelerometer first started taking readings. These four columns were removed from the dataset and the class labels added before any other processing was performed. Additionally, by producing a profile report on the data it became apparent that the dataset contained some columns that were constant, meaning all values in the column was the same. These columns could also be removed as they offer no contribution towards predictive power and are therefore irrelevant.

4.3.1 Missing (NaN) Values

The dataset contained some null values, represented programmatically as NaN. The quality of a model can be adversely affected with large amounts of missing values and therefore needed to be dealt with. (Witten, 2011) In particular, scikit-learn algorithms as used in this project cannot fit models with missing values. I decided to employ two different approaches depending on the amount of missing values for each feature. When a column had greater than 10% of its values missing, it was dropped

from the dataset. This was justified due to the large number of features present in the dataset and the large number of samples. Should a feature be missing more than 10% of its data, at least 500 samples must be missing a value for the feature in question. At this point, it would be unreasonable to estimate what this large number of missing values would be based off the existing data.

This resulted in the removal of three columns: 'ACC_nightwak_VIG400_mg_pla ', 'ACC_day_SIB_mg' and 'ACC_nightwak_VIG400_mg_wei'. The two columns containing 'nightwak' refer to patients night walking while wearing the accelerometer, these columns can only have values if a given patient went on a night walk at some point over the seven day period. 'ACC_day_SIB_mg' refers to sustained inactivity bouts which van Hees [2019] describes as "the periods of time during which the accelerometer does not rotate at all for at least 5 or 10 minutes. This could be daytime sleep, or the monitor not being worn for a very short period of time." It is therefore likely in this case that simply not enough patients recorded any form of daytime sleep.

In total, a further 37 columns contained NaN values, but not enough to trigger the 10% threshold. Upon inspection, all these features were numeric and continuous in nature. Therefore, it was possible to use imputation to replace the missing data. There are numerous different forms of imputation. One such strategy is to replace any missing data with the most common (modal) value; however, this strategy is most used with discrete values and not continuous data like the biobank dataset. This left a choice between imputing using the mean or median value of a given column. (Badr, 2019) To prevent potentially erroneous or otherwise exceptional values from influencing the imputed values too much, I opted to use the median value of the column rather than the mean.

4.3.2 Skew

Many warnings were present in the profile report for the dataset indicating skewed columns. A set of data is considered skewed when most of its values are far away from the statistical average (mean or median). Skewed columns have a negative impact on the ability of a machine learning model to make good predictions (Vasudev, 2017). Skewness can be visualised by graphing the values a feature takes and their frequency. An example of a skewed feature compared to more normally distributed feature present in the biobank dataset can be seen below.

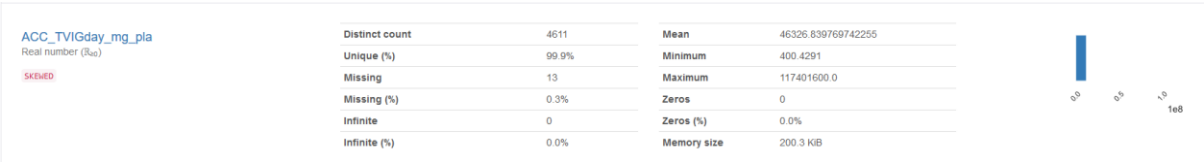


Figure 4.2 - A skewed feature (Pandas Profiling)

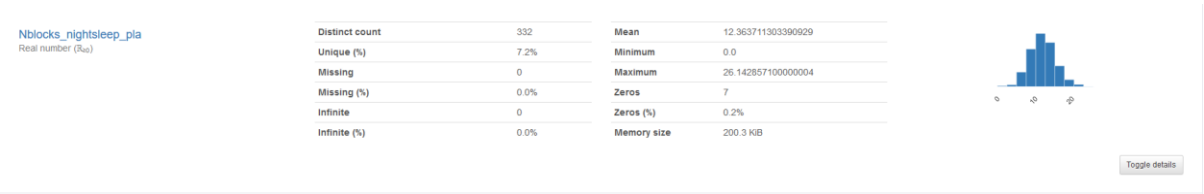


Figure 4.3 - A normally distributed feature (Pandas Profiling)

Although the data would be scaled before being used to fit the model to mitigate this issue, the most skewed columns were removed in advance. This is because if a column is very skewed with a large standard deviation, scaling the data is not enough to prevent the model having poor predictive power. When the features of the dataset are sorted by their absolute skewness, there is a large gap in values after 21.5, as the next value is 36.6. This also reflects the cut-off point in the profile report, where

columns are flagged as highly skewed. Due to this, all remaining features in the dataset with a positive or negative skew of greater than 30 were removed from the dataset. This resulted in the removal of an additional 40 columns, still leaving 152 features remaining. Due to the large number of features, a table of skewness values is not presented here but can be found in appendix E of this document.

4.4 Pipeline

In machine learning, a pipeline is something designed to automate workflow. (Shashanka, 2019). With scikit-learn, pipelines are used to create composite estimators, consisting of transformers combined with classifiers to produce a robust model. In the case of this project, as described in chapter 2, there are multiple important steps to apply to the final model to ensure maximum confidence in it. From a software engineering perspective, the use of a pipeline makes implementing each of these steps in a consistent manner as easy as possible, all the estimators can be trained in one simple call.

Additionally, the use of a pipeline brings many benefits: The model is made safer by ensuring that the same samples are used in training the transformers and estimators, preventing the accidental leaking of data from the test set into the trained model. It also makes the whole model more computationally efficient by avoiding repeated computation through caching results obtained in previous steps of the pipeline and a parameter grid can be used to tune parameters over all the pipeline components at once. These benefits are especially important for this project due to the use of grid search cross-validation. The use of grid search for parameter tuning combined with multiple training and test sets (cross validation) means that it is vital that these steps are applied to each pipeline component in a uniform method to ensure comparisons made between results are valid. Through this method, the model can obtain the best results possible and as a result, conclusions can be made from its ability to predict the class label of a new, unseen patient.

The pipeline constructed for this project takes four parts, all of which take place within the cross-validation loop. First, the data is scaled, this can be thought of as a continuation of accounting for skewed features as discussed in the previous section. This further scaling is performed using scikit-learn's RobustScaler. This component scales the data to standardise it. The RobustScaler was used instead of the StandardScaler because it is more resistant to outliers in the dataset. RobustScaler removes the median value as opposed to the mean and scales the data according to the inter-quartile range (IQR) rather than unit variance. This generally yields better results as it ignores the often-negative influence of outliers on the mean and variance of a feature set (scikit-learn user guide, 2019f). Next, as discussed in the above section, missing data values are imputed according to the median of a given column. It is important to note that this median value to use for imputation is only calculated on the training set. This is to ensure that exclusively the training set is used to affect the behaviour of the model. If there are missing values in the test dataset, these are imputed using the median calculated earlier based on the training set. At this point, the data set is prepared for being inputted into the model.

With a now-complete dataset, the process of recursive feature elimination (RFE) begins. As discussed in chapter 2, there are often features that are not useful for training a model. As outlined in the previous section, some obviously redundant features such as ID numbers were removed from the dataset before this point, however recursive feature elimination is still an important step to perform. It is necessary to reduce overfitting, improve accuracy and reduce train times for the overall estimator. (Brownlee, 2014) The biobank dataset after manual cleaning still contains over 150 features. This is a prohibitively large feature set that needs to be reduced, the exact number of features to select by RFE is tuned through the parameter grid for the pipeline to determine the best selection for maximum accuracy and performance. The step parameter for RFE is specified as 0.1, causing the estimator to

eliminate 10% of the features at a time, this is used rather than the conventional one at a time method to speed up the process for the large feature-set. The features chosen by each classifier for each fold and metric are recorded and counted to determine the most useful features in the dataset. All classifiers perform their own RFE apart from the nearest neighbour classifier which does not support this functionality due to not exposing any feature importance statistics. A linear SVC is used instead to perform RFE before the data is fed to the nearest neighbour model. The final step of the pipeline is then to fit the classifier. Each classifier has different hyper-parameters that are tuned by grid search cross-validation according to the contents of the parameter grid for the pipeline.

4.5 Visualisation

In addition to basic output for each metric used to test the model, recording the average and best performance both per fold and overall, and feature ranking, model.py also outputs a graph illustrating the ROC curve for the chosen classifier. As discussed in section 2.6.1, the larger the area under the curve, the higher the quality of the classifier output. However, the graph shows the curve for each fold with standard deviation in addition to an overall average. This gives an indication of the variance between folds and can indicate if the quality of the classifier is affected by changes in the training data and how different the generated splits from k-fold cross-validation are from one another. (scikit-learn developers, 2019g)

4.6 Parameter Selection

The model is developed in such a way as to test multiple different scikit-learn estimators. As previously stated, each classifier was tested using a range of parameters to find the optimal configuration through grid search cross-validation. However, a range of parameters to test must still be specified. Many classifiers contain such a wide range of tuneable hyperparameters it would be impractical to perform grid search on them all due to time constraints. For this reason, only the most impactful parameters are trained. At the start of the experiment, the biobank subset is made up of 152 features. As there is no general rule of thumb regarding the number of features to choose via RFE, unless otherwise stated, all classifiers were tested with between 25 and 125 features. A full list of the parameter grids used in grid search cross-validation can be found in the source code for model.py in appendix D.

4.6.1 Logistic Regression Classifier

The logistic regression classifier offers a large range of tuneable hyperparameters. Due to being designed for large datasets, the sag and saga solvers were tested. The saga solver allows a ratio of l2 to l1 regularisation, this is reflected in the l1 ratio parameter in the grid, set to 0.5 for testing. l1 regularisation is not tested due to it being designed for sparse datasets. Due to imputation to remove blank values in a prior step, the biobank dataset cannot be considered sparse. Logistic regression utilises a regularisation parameter, C to increase numerical stability (scikit-learn user guide, 2019c), and a tolerance setting. Smaller values of C represent stronger regularisation, and the classifier tests a range of values for C between 0.1 and 100. This tolerance setting is used for the stopping criteria, values that are too strict can cause the model to exceed the maximum number of iterations available. (Bacardit, 2020b) For this reason, the tolerance values tested are no lower than 0.001. Furthermore, to ensure model convergence, the maximum iterations allowed was set to 3000 rather than the default 100.

4.6.2 Linear Support Vector Classifier (Linear SVC)

The linear SVC also offers the choice between two different penalty options, l1 and l2, again, only the default l2 is used. Like the logistic regression classifier, linear SVCs rely on a regularisation parameter, C, and a tolerance setting (scikit-learn developers, 2019c). This classifier tests C values between 0.01

and 100, while tolerance values are tested at 0.01 and 0.001. To ensure model convergence, the maximum iterations allowed was increased from 1000 to 30000.

4.6.3 Random Forest Classifier

The two key parameters to tune for this classifier are the number of trees in the forest and the number of features available to each tree. As this experiment is a classification problem, the number of features is set to the recommended square root of the total features. A greater number of estimators (trees) leads to increased runtimes but better results (up to a certain point). To try and balance this, the classifier is tested with a range of trees from the default 100 to 300 (scikit-learn user guide, 2019a). By default, trees are left to fully develop until all features are fully separated, in practice, this consumes a large amount of time and memory. For this reason, the trees are tested with two parameters that limit the growth of trees, managing the maximum depth and the minimum samples needed to form a leaf node (scikit-learn developers, 2019e). Maximum depth ranges from 2 to 6, and minimum leaf samples from 2 to 8.

4.6.4 Nearest Neighbours Classifier

Due to being highly susceptible to the 'curse of dimensionality' as discussed in section 2.4.4, the nearest neighbour classifier is trained on a lower range of features between 10 and 50. The number of neighbours is tested between 3 and 9. These numbers are all odd as due to the experiment dealing with a binary classification problem, tie votes can be avoided. This is common practice for such a classification problem. (Naviani, 2018) The other key parameter for nearest neighbour classification, weights, governs how much influence each point in the neighbourhood has on the overall classification decision. The default uniform assigns equal weight to each point in the neighbourhood while the distance option places greater influence on those neighbours closest to the new point to classify (scikit-learn developers, 2019b). Both options are tested in the parameter grid.

Chapter 5 – Results and Evaluation

5.1 Cross-Validation Results

This section presents the results obtained from the experiments outlined above. The performance of each classifier, for each metric, for each fold as obtained via grid search is given in addition to the optimal set of parameters for that estimator and the total time taken to train the classifier through the cross-validation loop. The most important metric is the average performance across folds on the test sets, as this indicates the model's ability to correctly classify new, previously unseen data.

Each classifier's ROC curve is also plotted, to facilitate brief discussion about the effect of potential differences between training folds on performance and to better visualise the effectiveness of each model.

5.1.1 Logistic Regression Classifier

Metric		**Weighted F1**					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.647971	0.642948	0.640731	0.645577	0.633534	**0.642152**
	Test	0.646819	0.642551	0.631872	0.633771	0.648093	**0.640621**
Parameters	**No. Features**	100	75	100	100	75	**100**
	Solver	sag	saga	saga	saga	sag	**saga**
	Penalty	l2	elasticnet	elasticnet	elasticnet	l2	**elasticnet**
	C	100	10	100	100	1	**100**
	Tol	0.01	0.001	0.001	0.01	0.01	**0.01**

Metric		**ROC_AUC**					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.692861	0.690733	0.697701	0.697006	0.684221	**0.692505**
	Test	0.641497	0.628578	0.632651	0.637815	0.663875	**0.640883**
Parameters	**No. Features**	100	75	75	75	75	**75**
	Solver	sag	saga	sag	sag	saga	**sag**
	Penalty	l2	elasticnet	l2	l2	elasticnet	**l2**
	C	1	10	0.1	0.1	100	**0.1**
	Tol	0.001	0.01	0.001	0.001	0.001	**0.001**

Total Training Time (s)	**8565.65411**

As can be seen from the tables, logistic regression achieves consistent results of around 64% accuracy across all five cross-validation folds for both metrics. However, when optimising for ROC AUC, the model performs around 5% better on the training set than the test set, indicating a potential mild case of overfitting. However, this does not occur when optimising for f1 score. The total cross-validation loop took 143 minutes, this is a fast training time especially considering the large number of possible parameter configurations that needed to be tested via grid search.

Optimal parameter configurations differ slightly depending on which metric the model is optimised for. For weighted F1 score, the saga solver is slightly preferred with Elastic Net penalty and generally,

more features are preferred. In contrast, to maximise ROC AUC fewer features are preferred and the sag solver with l2 penalty is preferred. Regularisation parameter (C) and tolerance values also differ greatly between the two metrics. When optimising for F1 score, lower regulation (higher C values) and lower tolerance (larger value) is preferred, in contrast to achieve high ROC AUC scores high regulation and higher tolerance is preferred.

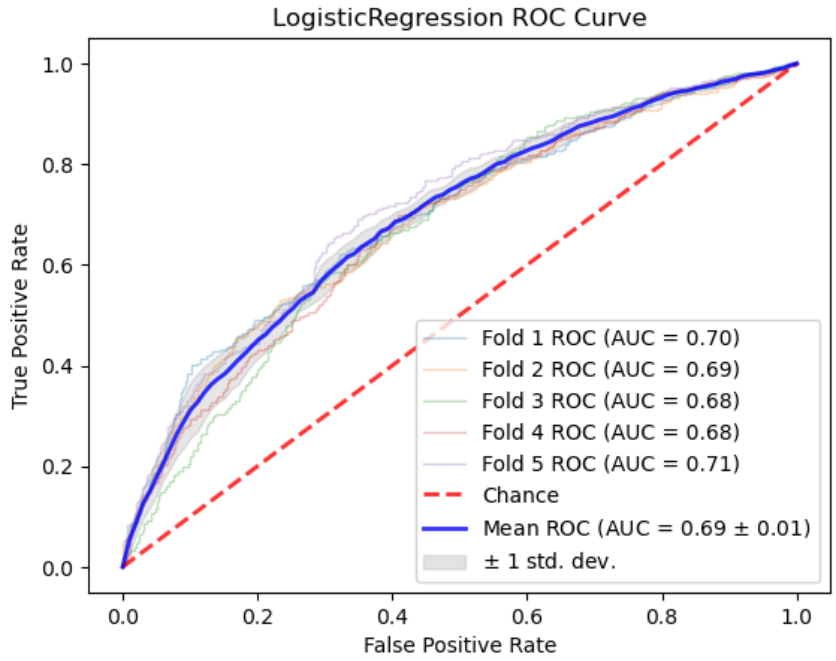


Figure 5.1 - Logistic Regression ROC Curve

It can be seen from the ROC curve itself that there is little variation between model performance on different folds, indicated by the low standard deviation of 0.01. This shows little to no effect due to different training data, giving confidence that the model would continue to be consistent in performance when performing predictions with new, unseen data.

5.1.2 Linear Support Vector Classifier (Linear SVC)

Metric		**Weighted F1**					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.642698	0.63944	0.64251	0.644342	0.645222	**0.642843**
	Test	0.625284	0.658714	0.643745	0.627083	0.627214	**0.636408**
Parameters	**No. Features**	100	125	100	100	125	**100**
	C	0.01	0.1	0.1	0.01	1	**0.01**
	Tol	0.01	0.01	0.01	0.001	0.01	**0.01**

Metric		ROC_AUC					
Fold		1	2	3	4	5	Average
Best Estimator	Training	0.688301	0.686427	0.693948	0.694575	0.69244	**0.691138**
	Test	0.624883	0.657819	0.647225	0.631893	0.635575	**0.639479**
Parameters	No. Features	100	125	125	100	125	**125**
	C	0.01	0.01	0.1	0.01	0.1	**0.01**
	Tol	0.001	0.001	0.001	0.01	0.001	**0.001**

Total Training Time (s)	9633.11161

The support vector classifier achieves solid results, proving to be correct approximately 64% of the time. It is again noticeable that for ROC AUC the model performs roughly 5% better on the training set, possibly indicating a mild case of overfitting. The total cross validation loop took 160 minutes; however, it is worth noting the relatively small number of hyper-parameters that needed to be tuned compared to other classifiers.

In terms of optimal parameter configurations, the classifier prefers large numbers of features when optimising for both ROC and F1 score. Across both metrics, high regularisation values (low C) are preferred, however when optimising for F1 some larger C values are chosen. Tolerance values were also generally lower when optimising for ROC compared to F1. Overall, both metrics lead to very similar optimal parameters.

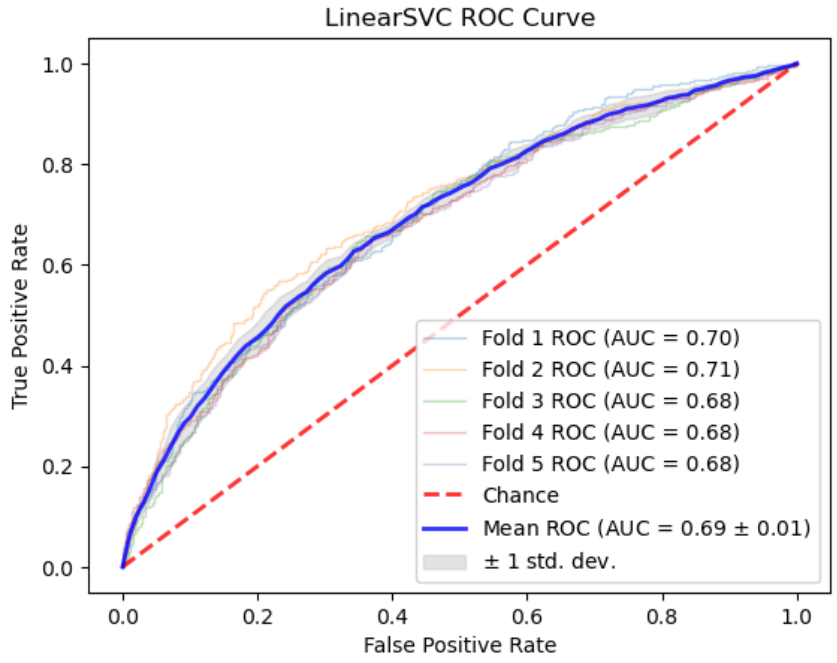


Figure 5.2 - Linear SVC ROC Curve

The ROC curve for the linear SVC indicates little variation in terms of performance on different data folds, achieving a very low standard deviation of 0.01. This gives confidence that the model would

maintain its performance on future data points and is relatively unaffected by differences between fold data.

5.1.3 Random Forest Classifier

Metric		Weighted F1					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.638596	0.637924	0.637704	0.637273	0.63152	**0.636603**
	Test	0.636059	0.61116	0.62208	0.645788	0.644705	**0.631958**
Parameters	**No. Features**	50	25	50	125	25	**50**
	Max Depth	6	6	6	6	6	**6**
	Min Samples Leaf	4	4	8	2	8	**4**
	No. Trees	200	100	200	100	200	**200**

Metric		ROC_AUC					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.68452	0.688154	0.682405	0.684165	0.678954	**0.683639**
	Test	0.642124	0.601629	0.633883	0.662816	0.638286	**0.635748**
Parameters	**No. Features**	100	25	50	50	25	**25**
	Max Depth	6	6	6	6	6	**6**
	Min Samples Leaf	4	2	4	2	2	**2**
	No. Trees	200	300	300	300	300	**300**

Total Training Time (s)	**49795.10315**

The random forest classifier achieves results only slightly worse than previous classifiers, proving correct roughly 63% of the time. As with other classifiers, when optimising for ROC AUC, the model performs around 5% better on the training set than the test set, indicating a mild case of overfitting. In stark contrast to other classifiers however, the total cross validation loop took a substantial 830 minutes, or 13.83 hours.

The optimal maximum depth for both metrics across all folds is the maximum option of 6. This is unsurprising as a greater maximum depth allows the trees to develop more fully and therefore achieve better results. However, this parameter was capped to reduce the already significant runtime of the training process. The optimal values for the other parameters differ based on the metric the model is optimising for. F1 score prefers more features than ROC AUC, but still does not prefer a significant number apart from one fold which used 125 features. F1 also prefers fewer trees (estimators) and a larger minimum samples per leaf than ROC AUC.

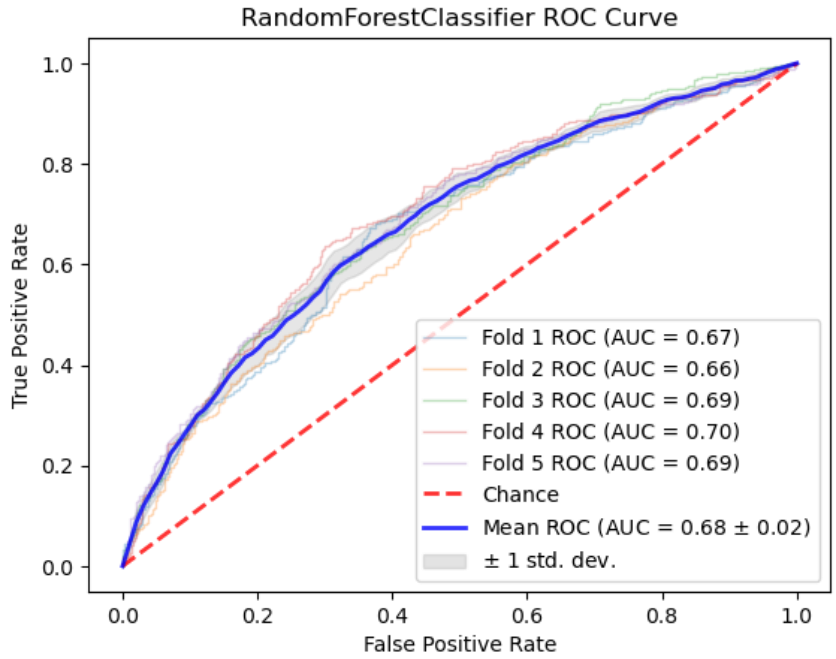


Figure 5.3 - Random Forest ROC Curve

From the graph, the random forest has a more significant variation on performance between folds compared to other classifiers. This may indicate it is affected somewhat between variations in the training sets, alternatively, the random nature of the forest may be the cause.

5.1.4 Nearest Neighbour Classifier

Metric		**Weighted F1**					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.579694	0.57818	0.574473	0.586806	0.576949	**0.57922**
	Test	0.577835	0.577081	0.560519	0.57879	0.584284	**0.575702**
Parameters	**No. Features**	20	40	40	50	30	**40**
	No. Neighbours	9	7	9	9	9	**9**
	Weights	uniform	uniform	distance	uniform	distance	**uniform**

Metric		**ROC_AUC**					
Fold		**1**	**2**	**3**	**4**	**5**	**Average**
Best Estimator	**Training**	0.600586	0.596835	0.594328	0.61342	0.605846	**0.602203**
	Test	0.588969	0.586483	0.582918	0.549265	0.573249	**0.576177**
Parameters	**No. Features**	50	50	30	40	50	**50**
	No. Neighbours	9	9	9	9	9	**9**
	Weights	uniform	uniform	distance	distance	distance	**distance**

Total Training Time (s)	**8097.944164**

The nearest neighbour classifier performs poorly on the biobank dataset, only achieving correct classifications on the test sets roughly 57% of the time. It can be seen from the ROC AUC table that the nearest neighbour classifier tends to overfit, indicated by noticeably higher training set scores than test set scores. This is unsurprising due to the model's susceptibility to the curse of dimensionality as explained in section 2.4.4. The overfitting is especially noticeable when the model chooses more features in the cross-validation loop. However, the classifier trains quickly, finishing the cross-validation loop in 135 minutes.

A higher number of neighbours is preferred overwhelmingly by both metrics on almost all folds. ROC AUC prefers a higher number of features than F1, potentially exploring why the model does not overfit when optimising for ROC AUC, fold 5 for F1 even manages to achieve a higher accuracy on the test set than the training set. The method of assigning weights to neighbours varies greatly between folds and metrics.

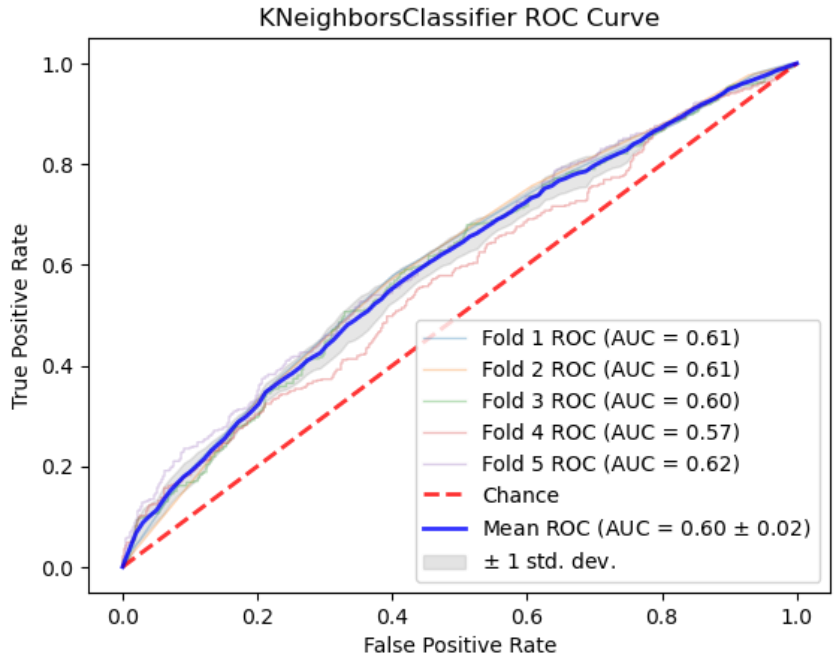


Figure 5.4 - Nearest Neighbour ROC Curve

It is noticeable from the curve that fold 4 performs noticeably worse than the other folds, potentially indicating that the model suffers from changes in the test data. Despite this, the overall standard deviation remains low at 0.02. Overall, the classifier appears consistent.

5.2 Performance Comparison

The following graphs compares the performance of each classifier based on the average of its best performances on each fold's test set, as well as the total time taken for each classifier to complete the cross-validation loop.

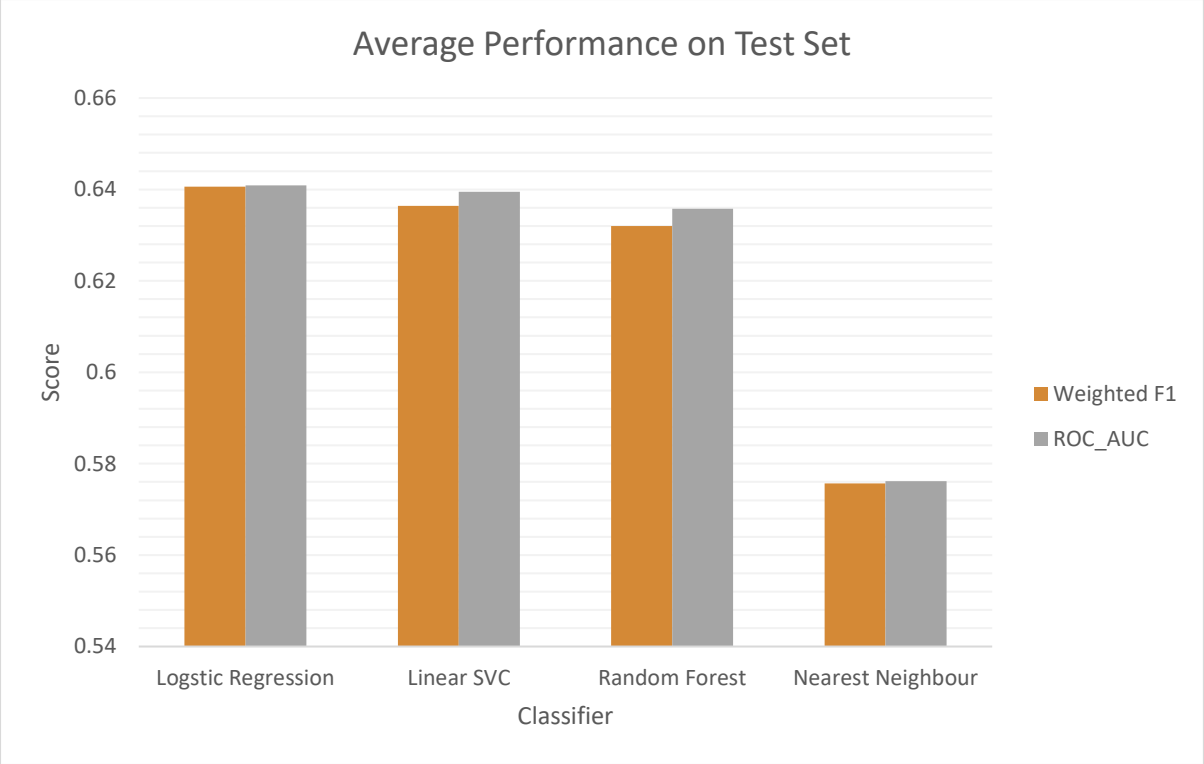


Figure 5.5 - Classifier Accuracy Comparison

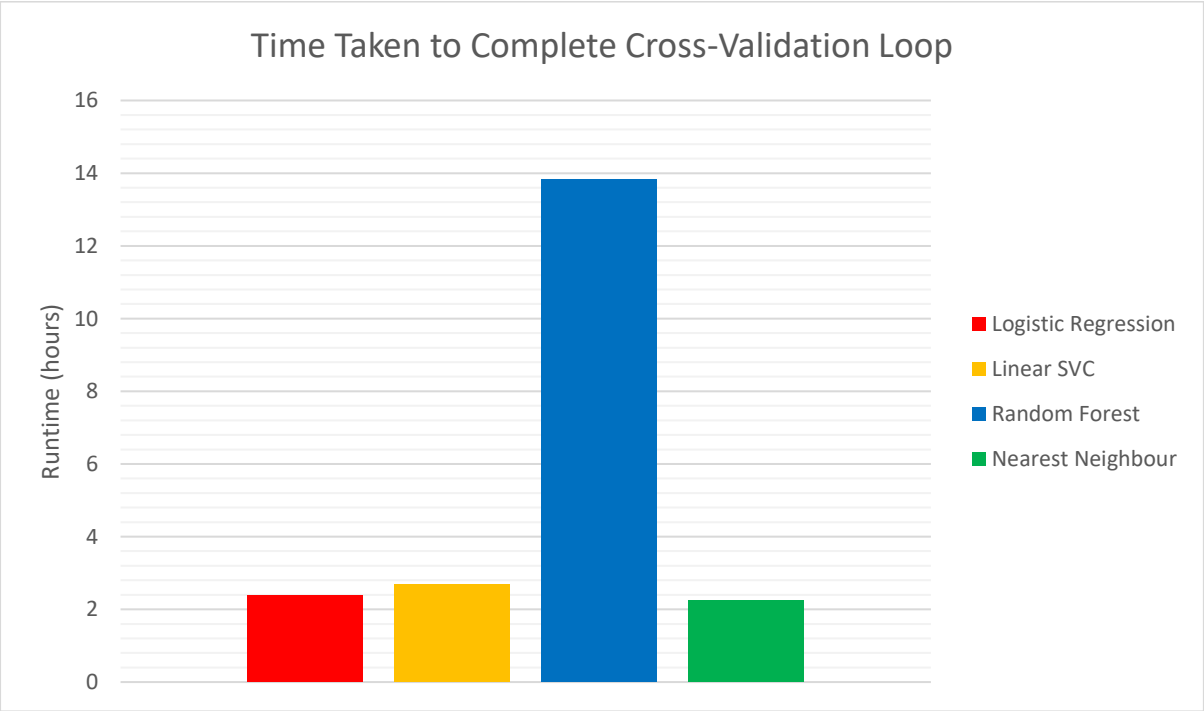


Figure 5.6 - Classifier Runtime Comparison

It can be seen from the graphs that, overall, logistic regression is the best method for classification of the biobank dataset. It is notable not only for scoring the highest on both metrics, but also for the fact that the two scores are very similar. All other classifiers score noticeably higher on ROC AUC than F1. It also achieves the second-lowest runtime despite its large parameter grid.

The nearest neighbour classifier scored most poorly, on average being 6% worse than the other three classifiers tested. To compensate for this somewhat it does take the least amount of time to train. This poor performance is likely to be due to the curse of dimensionality. The poor performance of the classifier even with a reduced feature set chosen implies that the problem modelled by the biobank subset is too complex or not obvious enough to be properly modelled by a low number of key features.

The random forest classifier took more than six times longer to train than any other classifiers and only managed to achieve better results than the nearest neighbour classifier. As a result of this I feel that despite not achieving the worst result it is overall the worst classifier for the Biobank dataset.

5.3 Feature Ranking

As mentioned in chapter 4, the number of times each feature was selected for use through RFE in an estimator that achieved the highest score for a fold was recorded. The full list of selected features can be found in appendix E. The features most often selected are likely to have the largest impact on the model making a successful prediction and are therefore the most likely to hold clinical significance. These most important features are the key focus of the fourth and final project objective. The table below shows the features that were selected at least 75% of the time (minimum 30 out of 40 possible times) and their meanings as explained by van Hees [2019].

Any features using a weighted average weight weekend days 2/5 relative to the contribution of weekdays.

Feature	Frequency	Meaning
dur_TSIBday_min_wei	39	The weighted average of the total time spent in minutes in sustained inactivity bouts during the day.
ACC_LIGB_D1T30_100_mg_wei	37	The weighted average of the acceleration during light activity bouts lasting at least one minute defined between 30 and 100mg.
Nbouts_MVPA_D1T100_wei	36	The weighted average of the number of bouts per day of moderate or vigorous activity lasting at least one minute defined as 100mg or higher.
sleeplog_onset_wei	34	The weighted average start time of the sleep period.
sleeplog_onset_pla	34	The plain average start time of the sleep period.
Nblocks_MVPA_D1T100_wei	34	The weighted average of the number of blocks per day of moderate or vigorous activity lasting at least one minute defined as 100mg or higher.
dur_day_min_wei	33	The weighted average length of the daytime period.
Nblocks_MVPA_D1T100_pla	32	The plain average of the number of blocks per day of moderate or vigorous activity lasting at least one minute defined as 100mg or higher.
dur_TLIGday_min_wei	32	The weighted average of the total time spent in minutes in light activity bouts per day.
dur_TLIGday_min_pla	32	The plain average of the total time spent in minutes in light activity bouts per day.
dur_INB_D10T30_min_wei	32	The weighted average of the time spent in minutes per day in inactivity bouts lasting at least one minute with a threshold of 30mg.

Feature	Frequency	Meaning
ACC_TMODday_mg_pla	32	The plain average of the acceleration during all daytime moderate activity bouts.
dur_day_LIG30_100_min_pla	31	The plain average of the time in minutes spent in daytime light activity bouts lasting at least one minute defined between 30 and 100mg.
ACC_TLIGday_mg_pla	31	The plain average of the acceleration during all daytime light activity bouts.
ACC_LIGB_D1T30_100_mg_pla	31	The plain average of the acceleration during light activity bouts lasting at least one minute defined between 30 and 100mg.
dur_TMODday_min_wei	30	The weighted average of the time spent in minutes in all daytime moderate activity bouts.
dur_day_OIN30_min_pla	30	The plain average of the time spent in minutes in other inactivity during the day with a threshold of 30mg not part of inactivity bouts.

5.4 Clinical Significance

The third and fourth project objectives, as well as the overall aim, depend on analysis of these results determining the presence of potential clinical significance held within the biobank dataset. Consistent scores of 0.64 across classifiers on test sets translates to the model being correct when classifying new data approximately 64% of the time. Although not a strong enough result to suggest a significant relationship, it is likely from this figure that physical activity levels and sleeping patterns contribute to the development of various metabolic diseases.

Although the model does not expose specifics, based on the existing research in the field as outlined in section 2.9, it is likely that poor activity levels increase risk. This is evidenced by many of the most important features as deduced from RFE involving physical activity in some way. The single most important feature, selected 39 times out of a possible 40, concerned the amount of time spent inactive. This was closely followed by average acceleration during light activity, and the total number of moderate or vigorous activity bouts. These features alone are enough to indicate that simply sending less time sedentary can affect the likelihood of developing a metabolic disease, echoing the findings of Bankoski [2011]. Several other features commonly selected concern only light activity, suggesting that even a mild increase in activity levels is enough to influence certain diseases. An intense exercise regime is likely unnecessary and even something as simple as walking more often could make a difference, this supports the conclusions reached by Sisson [2010].

The fourth and fifth most important features concern the average time that a participant went to sleep. Although not surprising that the sleep patterns of patients were useful when classifying the data when considering the research of Cappuccio [2017] and Kim [2018], it is notable that the time a person went to sleep (34 of 40 folds) was favoured over the average amount of sleep a person received (25 of 40 folds). This is an area that may warrant further investigation to establish if the time an individual goes to bed is more important than the total amount of sleep they get for minimising the risk of developing a metabolic disease and the scientific explanation for this.

It is important to understand that the problem is more complex to model and cannot be fully understood just using these few key features. This is reflected by the generally high number of features selected by the best classifiers, and the poor performance of the nearest neighbour classifier, which is known to struggle with large feature sets. Despite not supporting a significant link between

lifestyle choices and metabolic diseases, the results of this project's experiments could be used to advise patients that are otherwise healthy whether or not they may be on course to developing some form of metabolic disease based on their physical activity and sleep patterns, and if consider improving their physical activity levels or adjusting their sleep pattern accordingly.

Chapter 6 – Conclusion

This chapter offers a reflection on the project relative to the original aims and objectives and discusses the schedule the project was completed to compared to the original plan. Additionally, it considers future work that could be carried out to further strengthen or add additional context to the conclusions already reached.

6.1 Project Aim and Objectives

Aim: Assess techniques to extract features from accelerometer data and use data mining to determine if these features can be linked to the descriptors of metabolic diseases.

Overall, I feel the project has been a success, and achieved its overall aim. Appropriate feature engineering techniques were identified, and one was used to extract features from a substantial subset of the accelerometer data. From these features, a machine learning pipeline was constructed to perform data mining and produce statistical and graphical output for analysis. Using appropriate contemporary research in the field of metabolic diseases and accelerometer data, measured conclusions were drawn, and hypotheses presented to explain the links between the features and metabolic disease descriptors.

1) Identify and understand suitable feature engineering techniques for accelerometer data.

The two most prominent feature engineering methods for accelerometer data as identified in this report are the R package GGIR and the python-based method developed by Oxford University. The project focuses on GGIR particularly because R is a common statistical language used in data science and is easy to parallelise across the high-powered machine used for data processing. The Oxford method in comparison is Python based and specifically designed to interpret the Biobank dataset, making it a natural candidate. Due to time constraints, beyond the initial justification provided in chapter 3, it was impossible to critically evaluate and compare these two methods of extracting features from the Biobank data. Ultimately, the only way to quantify the effectiveness of both methods would be to compare the results achieved when training a classifier on the features extracted. Overall, I feel that although this objective was met in terms of identification and understanding of the methods, the justification behind choosing to use GGIR over the Oxford method could have been more solid and ideally backed by some data.

2) Establish validation protocols to determine if the descriptors are linked to diseases.

As documented in chapter 4 and explained theoretically in chapter 2, the final machine learning model includes a range of methods designed to ensure maximum confidence in its output. Between a cross-validation loop, the use of a scikit-learn pipeline designed to avoid common software engineering pitfalls and a thorough grid-search I feel extremely confident that the model constructed produced the best quality output possible. Additionally, multiple classifiers were researched and tested to discover the best underlying implementation for obtaining the best accuracy on the test data.

All four classifiers described had their performance evaluated according to two separate metrics most used to measure performance on classification problems. This was done to investigate numerous different aspects of what makes an effective classifier, including precision, recall and false positive/negative rate. All these metrics can be extremely important and should be considered, especially in a medical setting where an incorrect diagnosis can cost valuable money or time.

3) Analyse, by training and validating the machine learning model, whether the features extracted can establish a link to the diseases.

As discussed above, I believe that the training and validation process was carried out in a thorough fashion, ensuring that any output data could be interpreted with confidence when used as the basis for any conclusion, however firm. The 64% correctness of the best classifier is enough to indicate some

link between the extracted features and metabolic diseases. I am confident this conclusion is reasonable and well evidenced given the results obtained by the experiment.

4) Interpret if and why the best features identified hold clinical significance.

Using feature ranking, I was able to identify the best features used by the model to influence its predictions. The identification of fitness and sleep related features as the most important is supported by current research in the field. While the notion of the time sleep begins being highly influential is not as well documented in existing literature. I feel it is an interesting finding that requires further investigation and consideration. Overall, I am happy this objective was met.

6.2 Planning and Schedule

The actual progress of my project saw some marked deviations from the original plan presented in section 1.3. In addition to my own decisions, the outbreak of COVID-19 and the subsequent suspension of face-to-face teaching in March saw several hard deadlines moved back to compensate for the challenging circumstances I would now have to complete the project under. The diagram below documents these deviations.

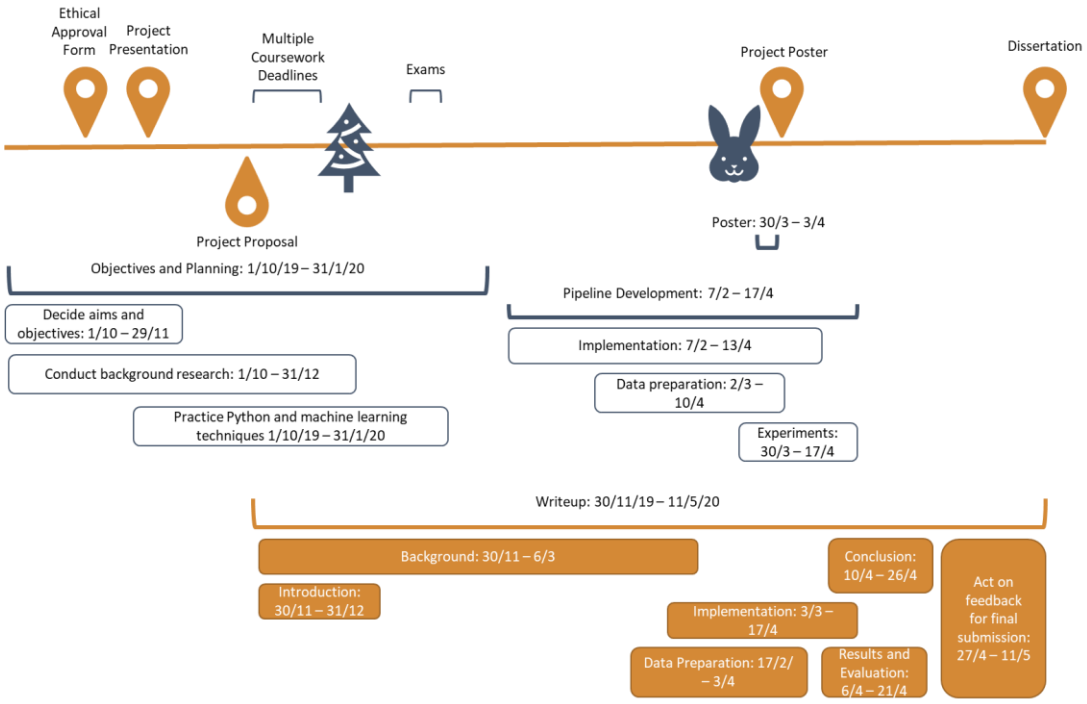


Figure 6.1 - Project Progress

Before Christmas, the large workload of semester one meant that I was unable to complete much of the project. As a result, some practical learning of Python and machine learning techniques had to continue after the exam period before I could begin the implementation properly. The background writeup section was expanded throughout this period and beyond as I added explanations of new concepts as they were incorporated into the model.

It became apparent early on in development that the project would not have an explicit design and testing section. The original plan was designed with a conventional software engineering project following the agile model, with iterative design, implementation, and testing phases in mind. Due to my lack of experience with machine learning, I was unaware of how to plan in any other way. During the project, I found that the design of the experiment is more easily expressed in the description of the implementation, and that concepts such as class diagrams were not applicable to this kind of

project. Instead, the preparation of the data set, the design of the experiments and the results extracted are expressed chronologically, explaining what would otherwise take the form of the design, implementation, and testing sections in a conventional project.

The most time-consuming part of the project was the data preparation section described in chapter 3 and the data cleaning in section 4.3. This came as a surprise to me when doing the project and is reflected in my lack of planning around this area. If I was to do this project again, one of the objectives would concern the preparation of data to reflect the substantial amount of work and consideration required in addition to allocating additional time in my project plan to account for this. The length of time required to extract the features was much larger than I first imagined. To this end, this step would have been prepared as a priority, so that work could continue in parallel on the python implementation of the model rather than the project being bottlenecked waiting for the feature engineering portion to conclude. Beginning to construct my model without beginning this section was a mistake and cost a large amount of time. Problems encountered during the final implementation could have been solved much earlier if the final dataset had been ready. Several classifiers would not converge without increasing their number of iterations, and the data required further cleaning via additional python code to eliminate heavily skewed or constant features. These problems were not present in the clean datasets used to practice on, and as such I was not prepared for them. This also necessitated the completion of additional background research to decide the best way to solve them and provide context to the reader.

The poster took much less time than expected, however, due to delays in implementation, I was unable to produce results in time to display on the poster as originally planned. Once the model was complete, it was revised to add new, more clear output, including the ROC curves and complete feature ranking. These aspects were essential to the completion of the third and fourth project objectives. The experiments themselves took around a week to run fully, during this time, some aspects of this chapter were completed, such as reflecting on earlier objectives, in addition to tidying up earlier sections.

6.3 Future Work

It was necessary to make simplifications of the dataset to ensure that the project was delivered on time. The reduction of the number of class labels from four to two changed the problem considerably, my main hope for the future is to analyse the dataset with respect to the full range of class labels. It is possible that meaningful patterns may be contained within the data that are specific to a combination of metabolic diseases rather than just the distinction between healthy and unhealthy. Additionally, the methodology behind designing and evaluating the performance of a multi-class classification experiment (as touched on in chapter 2) is very different and would be an exciting challenge.

Although likely impractical, it would also be interesting to perform an analysis of all patients with class labels and accelerometer data. The large increase in the volume of data would likely necessitate the research and use of different classifiers designed for larger datasets and would allow me to place more confidence in my extracted results. Unfortunately, the lack of access to a high-powered computing cluster capable of extracting features from a dataset this large in a reasonable time frame renders this impossible to do.

Furthermore, referring to my evaluation of the first project objective, I would like to analyse the performance of the various classifiers on features extracted by the Oxford method or a combination of GGIR and the Oxford method. Additionally, these results could be compared to those achieved using deep learning to bypass explicit feature engineering entirely.

References

Althubaiti A. (2016). Information bias in health research: definition, pitfalls, and adjustment methods. *Journal of multidisciplinary healthcare*, *9*, pp.211–217.

AX3 User Guide (2013). Revision 1.2. Axivity.

Bacardit, J. (2019). CSC3423 – Biologically-inspired Computing L04 – Genetic Algorithms for Machine Learning.

Bacardit, J. (2020a). Introduction to Machine Learning Part 2: Evaluation protocols. [Online] Available at: https://github.com/jaumebp/ML-tutorial/blob/master/slides/ML-2.pptx [Accessed: 18/03/2020]

Bacardit, J. (2020b). jaume.bacardit@newcastle.ac.uk. Re: Biobank Dissertation. 9th April 2020.

Bacardit, J. (2020c). Jaume.bacardit@newcastle.ac.uk. Re: Biobank Dissertation Draft. 5th May 2020.

Badr, W. (2019) 6 Different Ways to Compensate for Missing Values In a Dataset (Data Imputation with examples). [Online] Available at: https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779 [Accessed: 04/04/2020]

Bankoski, A., Harris, T., McClain, J. *et al.* (2011). Sedentary Activity Associated With Metabolic Syndrome Independent of Physical Activity. *Diabetes Care Feb 2011.*

Batista, G.E.P.A., Prati, R.C., Monard, M.C. (2004). A study of the behaviour of several methods for balancing machine learning training data. SIGKDD Explorations 6 (1) pp.20-29. doi: 10.1145/1007730.1007735

Beklemysheva, A. (2020). Why Use Python for AI and Machine Learning?. [Online] Available at: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/ [Accessed: 07/02/2020]

Brownlee, J. (2014). Feature Selection to Improve Accuracy and Decrease Training Time. [Online] Available at: https://machinelearningmastery.com/feature-selection-to-improve-accuracy-and-decrease-training-time/ [Accessed: 08/04/2020]

Brownlee, J. (2018). A Gentle Introduction to k-fold Cross-Validation. [Online] Available at: https://machinelearningmastery.com/k-fold-cross-validation/ [Accessed: 25/03/2020]

Brownlee, J. (2019). Classification Accuracy is Not Enough: More Performance Measures You Can Use. [Online] Available at: https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/ [Accessed: 19/03/2020]

Brugman, S. (2020). Pandas Profiling. [Online] Available at: https://pandas-profiling.github.io/pandas-profiling/docs/ [Accessed: 09/04/2020]

Bycroft, C., Freeman, C., Petkova, D. *et al.* (2018). The UK Biobank resource with deep phenotyping and genomic data. *Nature 562***.** pp.203–209

Cappuccio, F. P., & Miller, M. A. (2017). Sleep and Cardio-Metabolic Disease. *Current cardiology reports*, 19(11), 110. https://doi.org/10.1007/s11886-017-0916-0

Cawley, G.C., Talbot, N.L.C. (2007). Preventing Over-Fitting during Model Selection via Bayesian Regularisation of the Hyper-Parameters. *Journal of Machine Learning Research 8.* pp.841-861

Doherty, A. (2018*).* Biobank Accelerometer Analysis Extracting meaningful health information for large scale accelerometer datasets. [Online] Available at: https://biobankaccanalysis.readthedocs.io/en/latest/ [Accessed: 04/01/2020]

Doherty, A., Jackson, D., Hammerla, N. *et al.* (2017). Large Scale Population Assessment of Physical Activity Using Wrist Worn Accelerometers: The UK Biobank Study.

Gandhi, R. (2017). Support Vector Machine — Introduction to Machine Learning Algorithms. *Towards Data Science.* [Online] Available at: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 [Accessed: 11/04/2020]

Kim, C.E., Shin, S., Lee, H. *et al*. (2018). Association between sleep duration and metabolic syndrome: a cross-sectional study. *BMC Public Health 18*, 720. https://doi.org/10.1186/s12889-018-5557-8

López, V., Fernandez, A., Garcia, S. *et al.* (2013). An Insight into Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics. Information Sciences 250. pp.113-141. doi: 10.1016/j.ins.2013.07.007

McCambridge, J., Witton, J., Elbourne, D. (2014). Systematic Review of the Hawthorne effect: New concepts are needed to study research participation effects*. Journal of Clinical Epidemiology.* pp.267-277.

Migueles JH, Rowlands AV, *et al*. (2019). GGIR: A Research Community-Driven Open Source R Package for Generating Physical Activity and Sleep Outcomes From Multi-Day Raw Accelerometer Data. *Journal for the Measurement of Physical Behaviour. 2(3).* doi: 10.1123/jmpb.2018-0063.

Narkhede, S. (2018). Understanding AUC - ROC Curve. [Online] Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 [Accessed: 19/03/2020]

Naviani, A. (2018). KNN Classification using Scikit-learn. *Datacamp.* [Online] Available at: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn [Accessed: 11/04/2020]

Openmovement (2018). *Open Movement Conversion Program*. [Online] Available at: https://github.com/digitalinteraction/openmovement/tree/master/Software/AX3/omconvert [Accessed: 04/03/2020]

OxfordDictionaries (2020). Meaning of overfitting in English. [Online] Available at: https://www.lexico.com/definition/overfitting [Accessed: 18/03/2020]

Pedregosa *et al*. (2011). Scikit-learn: Machine Learning in Python*. JMLR 12*. pp.2825-2830.

Radečić, D. (2019). Feature Selection in Python – Recursive Feature Elimination. [Online] Available at: https://towardsdatascience.com/feature-selection-in-python-recursive-feature-elimination-19f1c39b8d15 [Accessed: 25/03/2020]

Raschka, S. (2020). Machine Learning FAQ. [Online] Available at: https://sebastianraschka.com/faq/docs/difference_classifier_model.html [Accessed: 11/04/2020]

Sabia S, van Hees VT, Shipley MJ *et al.* (2014). Association between questionnaire- and accelerometer-assessed physical activity: the role of sociodemographic factors. *Am J Epidemiol*. 2014 Mar 15;179(6):781-90. doi: 10.1093/aje/kwt330. Epub Feb 4. PMID: 24500862

scikit-learn developers. (2019a). f1 score documentation. [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html [Accessed: 15/04/2020]

scikit-learn developers. (2019b). KNeighborsClassifier documentation. [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html [Accessed: 13/04/2020]

scikit-learn developers. (2019c). LinearSVC documentation. [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html [Accessed: 13/04/2020]

scikit-learn developers. (2019d). LogisticRegression documentation. [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed: 13/04/2020]

scikit-learn developers. (2019e). RandomForestClassifier documentation. [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed: 13/04/2020]

scikit-learn developers. (2019f). Receiver Operating Characteristic (ROC). [Online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html [Accessed: 19/03/2020]

scikit-learn developers. (2019g). Receiver Operating Characteristic (ROC) with cross validation. [Online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html [Accessed: 11/04/2020]

scikit-learn developers. (2019h). Visualizing cross-validation behavior in scikit-learn. [Online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html [Accessed: 25/03/2020]

scikit-learn documentation. (2019a). Choosing the right estimator. [Online] Available at: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html [Accessed: 04/03/2020]

scikit-learn documentation. (2019b). Recursive feature elimination with cross-validation. [Online] Available at: https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html [Accessed: 25/03/2020]

scikit-learn user guide. (2019a). 1.11.2 Forests of randomized trees. [Online] Available at: https://scikit-learn.org/stable/modules/ensemble.html#forest [Accessed: 13/04/2020]

scikit-learn user guide. (2019b). 1.4.7. Mathematical formulation. [Online] Available at: https://scikit-learn.org/stable/modules/svm.html#mathematical-formulation [Accessed: 13/04/2020]

scikit-learn user guide. (2019c). 1.1.11. Logistic regression. [Online] Available at: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression [Accessed: 13/04/2020]

scikit-learn user guide. (2019d). 3.2. Tuning the hyper-parameters of an estimator. [Online] Available at: https://scikit-learn.org/stable/modules/grid_search.html [Accessed: 25/03/2020]

scikit-learn user guide. (2019e). 6.1. Pipelines and composite estimators. [Online] Available at: https://scikit-learn.org/stable/modules/compose.html [Accessed: 07/04/2020]

scikit-learn user guide. (2019f). 6.3 Preprocessing data. [Online] Available at: https://scikit-learn.org/stable/modules/preprocessing.html [Accessed: 07/04/2020]

Shashanka, M. (2019). What is a Pipeline in Machine Learning? How to create one? [Online] Available at: https://medium.com/analytics-vidhya/what-is-a-pipeline-in-machine-learning-how-to-create-one-bda91d0ceaca [Accessed: 07/04/2020]

Shukla, S. (2019). What is Regression and Classification in Machine Learning? [Online] Available at: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ [Accessed 06/03/2020]

Sisson, S., Camhi, S., Church, T. *et al*. (2010). Accelerometer-Determined Steps/Day and Metabolic Syndrome. *American Journal of Preventive Medicine Vol 38, Issue 6, June 2010.* pp.575-582.

Subasi, C. (2019). Logistic Regression Classifier. *Towards Data Science.* [Online] Available at: https://towardsdatascience.com/logistic-regression-classifier-8583e0c3cf9 [Accessed: 11/04/2020]

van Hees, VT., Fang, Z., Zhao, JH. *et al.* (2019). GGIR Package Manual. [Online] Available at: https://cran.r-project.org/web/packages/GGIR/GGIR.pdf [Accessed 04/04/2020]

van Hees, VT., Gorzelniak, L. *et al*. (2013). Separating Movement and Gravity Components in an Acceleration Signal and Implications for the Assessment of Human Daily Physical Activity. PLoS ONE 8(4)

Vasudev, R. (2017). How to deal with Skewed Dataset in Machine Learning? *Becoming Human: Artificial Intelligence Magazine* [Online] Available at: https://becominghuman.ai/how-to-deal-with-skewed-dataset-in-machine-learning-afd2928011cc [Accessed: 09/04/2020]

Willetts, M., Hollowell, S., Aslett, L. *et al.* (2018). Statistical machine learning of sleep and physical activity phenotypes from sensor data in 96,220 UK Biobank participants. *Sci Rep* 8.

Witten, I., Frank, E., Hall, M. (2011). Data mining – practical machine learning tools and techniques (3rd Ed). Elsevier. pp.3-187

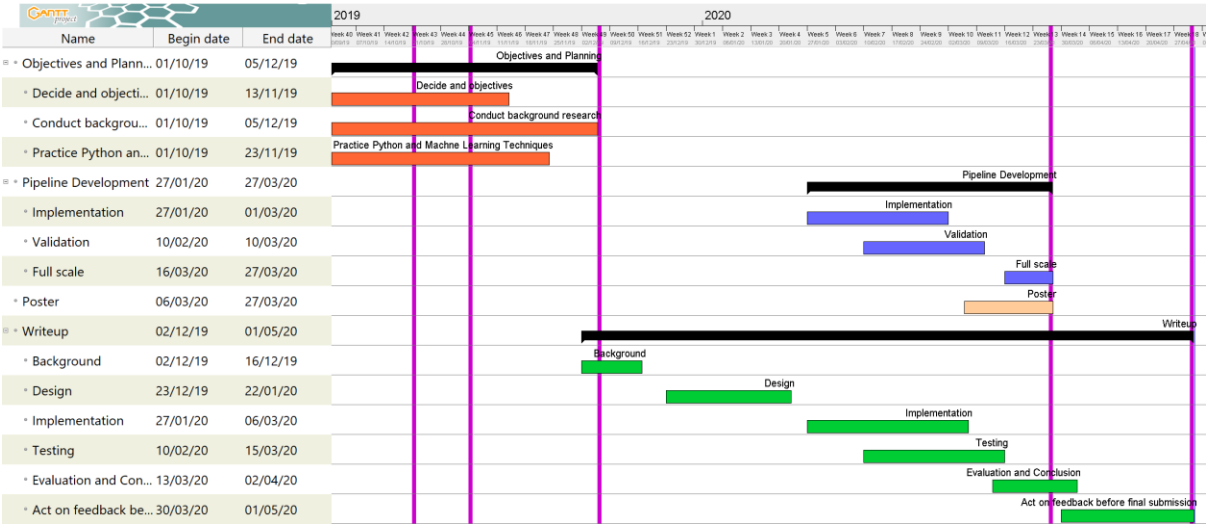
Yiu, T. (2019). Understanding Random Forest. *Towards Data Science.* [Online] Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed: 11/04/2020]

Zadrozny, B., Langford, J., Abe, N. (2003). Cost–sensitive learning by cost–proportionate example weighting. *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03).* pp. 435–442

Appendices

Appendix A – Project Gantt Chart

Original Project Plan



Appendix B – GGIR Script

BiobankDissertation.shell.GGIR

```
library(GGIR)
g.shell.GGIR(
             mode=c(1,2,3,4,5),
             datadir="/data1/biobank/b6021289/datadir",
             outputdir="/data1/biobank/b6021289/outputdir",
             do.report=c(2,4,5),
             #======================
             # Part 2
             #======================
             strategy = 1,
             hrs.del.start = 0,          hrs.del.end = 0,
             maxdur = 9,                 includedaycrit = 16,
             qwindow=c(0,24),
             mvpathreshold =c(100),
             bout.metric = 4,
             excludefirstlast = FALSE,
             includenightcrit = 16,
             #======================
             # Part 3 + 4
             #======================
             def.noc.sleep = 1,
             outliers.only = TRUE,
             criterror = 4,
             do.visual = TRUE,
             #======================
             # Part 5
             #======================
             threshold.lig = c(30), threshold.mod = c(100),  threshold.vig
= c(400),
             boutcriter = 0.8,      boutcriter.in = 0.9,     boutcriter.lig
 = 0.8,
             boutcriter.mvpa = 0.8, boutdur.in = c(1,10,30), boutdur.lig =
c(1,10),
             boutdur.mvpa = c(1),
             #======================
             # Visual report
             #======================
             timewindow = c("WW"),
             visualreport=TRUE)
```

Appendix C – Java Helper Programs

DropEmptyClassLabels.java

```
import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * @author Adam Barron
 * @author 160212899
 * Drop any patients with no class label.
 */
public class DropEmptyClassLabels {

    public static void main(String[] args) {
        if (args.length != 2)
            throw new IllegalArgumentException("Usage: <input_file>
<output_file>");
        String inputFile = args[0];
        String outputFile = args[1];

        BufferedReader br = null;
        BufferedWriter bw = null;
        // Look for patients with no class labels
        Pattern pat = Pattern.compile(", ");
        Matcher mat = pat.matcher("");
        try {
            br = new BufferedReader(new FileReader(inputFile));
            bw = new BufferedWriter(new FileWriter(outputFile));
            String currentLine;
            while ((currentLine = br.readLine()) != null) {
                mat.reset(currentLine);
                if (!(mat.find())) {
                    bw.write(currentLine + "\n");
                }
            }
            bw.close();
            br.close();
        } catch (FileNotFoundException e) {
            System.out.println("File " + inputFile + " not found");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

GetAccelerometerIDs.java

```
import java.io.*;

/**
 * @author Adam Barron
 * @author 160212899
 * Produce a CSV file of IDs for all patients with accelerometer data.
 * Can take 2 args to run over data1 and data2 into one csv.
 */
public class GetAccelerometerIDs {

    private static BufferedWriter bw;

    public static void main(String[] args) {
        if (args.length < 2)
            throw new IllegalArgumentException("Usage: <output_file>
<filepath_1> <optional_filepath>");
        String outputFile = args[0];
        File[] files;
        try {
            bw = new BufferedWriter(new FileWriter(outputFile));
        } catch (IOException e) {
            e.printStackTrace();
        }

        String filePath = args[1];
        files = new File(filePath).listFiles();
        try {
            appendIDs(files);
        } catch (IOException e) {
            e.printStackTrace();
        }

        if (args[2] != null) {
            filePath = args[2];
            files = new File(filePath).listFiles();
            try {
                appendIDs(files);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        try {
            bw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void appendIDs(File[] files) throws IOException {
        for (File f : files) {
            if (f.isDirectory()) {
                // Recurse over folders
                appendIDs(f.listFiles());
            } else {
                // Split on "_" to get ID component of file name
```

```
                String[] filename = f.getName().split("_");
                String id = filename[0];
                bw.write(id + "\n");
            }
        }
    }
}
```

FindCSVIntersection.java

```
import java.io.*;
import java.util.*;

/**
 * @author Adam Barron
 * @author 160212899
 * Produces a new CSV file with class labels of those patients with
accelerometer data.
 */
public class FindCSVIntersection {

    public static void main(String[] args) {
        if (args.length != 3)
            throw new IllegalArgumentException("Usage: <class_labels>
<accelerometer_ids> <output_csv");

        String classLabelInput = args[0];
        String idsInput = args[1];
        String outFile = args[2];

        TreeMap<Integer, Integer> classLabels = null;
        ArrayList<Integer> ids = null;
        try {
            classLabels = sortClassLabels(classLabelInput);
            ids = sortIds(idsInput);
            outputIntersection(classLabels, ids, outFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static TreeMap<Integer, Integer> sortClassLabels(String
filepath) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(filepath));
        // TreeMaps automatically sort keys
        TreeMap<Integer, Integer> classLabels = new TreeMap<>();

        // Skip header row
        String headerLine = br.readLine();
        String currentLine;
        while ((currentLine = br.readLine()) != null) {
            String lineSplit[] = currentLine.split(",");
            if (lineSplit.length == 2) {
                int[] ints = new int[2];
                ints[0] = Integer.valueOf(lineSplit[0]);
                ints[1] = Integer.valueOf(lineSplit[1]);
                classLabels.put(ints[0], ints[1]);
            }
        }
        br.close();
        return classLabels;
    }

    private static ArrayList<Integer> sortIds(String filepath) throws
IOException {
        BufferedReader br = new BufferedReader(new FileReader(filepath));
```

```
        ArrayList<Integer> ids = new ArrayList<>();

        String currentLine;
        while ((currentLine = br.readLine()) != null) {
            // Ignore non integers
            try {
                Integer i = Integer.valueOf(currentLine);
                ids.add(i);
            } catch (NumberFormatException e) {
                System.out.println("WARNING: Non-integer ID found: " +
currentLine);
            }
        }
        br.close();
        Collections.sort(ids);
        return ids;
    }

    private static void outputIntersection(TreeMap<Integer, Integer>
classLabels, ArrayList<Integer> ids, String outfile) throws IOException {
        BufferedWriter bw = new BufferedWriter(new FileWriter(outfile));

        // Add header row
        bw.write("eid,HEALTHY_CVD_T2D1_T2D2\n");
        for (Integer i : ids) {
            if (classLabels.containsKey(i)) {
                bw.write(i + "," + classLabels.get(i) + "\n");
            }
        }
        bw.close();
    }

}
```

GenerateSubset.java

```java
import java.io.*;
import java.util.Random;

/**
 * @author Adam Barron
 * @author 160212899
 * Generate a subset of the biobank data according to some probability.
 */
public class GenerateSubset {

    public static void main(String[] args) {
        if (args.length != 3)
            throw new IllegalArgumentException("Usage: <class_labels>
<probability> <output_file>");
        String classLabels = args[0];
        double prob = Double.parseDouble(args[1]);
        if (prob >= 1 || prob <= 0)
            throw new IllegalArgumentException("Probability must be between
0 and 1");
        String outFile = args[2];

        try {
            BufferedReader br = new BufferedReader(new
FileReader(classLabels));
            BufferedWriter bw = new BufferedWriter(new
FileWriter(outFile));
            // Copy header line to output
            bw.write(br.readLine() + "\n");
            String currentLine;
            int count = 0;
            while ((currentLine = br.readLine()) != null) {
                double random = new Random().nextDouble();
                if (random < prob) {
                    bw.write(currentLine + "\n");
                    count++;
                }
            }
            bw.close();
            br.close();
            System.out.println("Created subset of size " + count);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

SimplifyClassLabels.java

```java
import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * @author Adam Barron
 * @author 160212899
 * Simplifies class labels. All unhealthy patients (1,2,3) become class 1,
all healthy patients (0) remain the same.
 */
public class SimplifyClassLabels {

    public static void main(String[] args) {
        if (args.length != 1)
            throw new IllegalArgumentException("Usage: <class_labels>");
        String input = args[0];
        // Look for unhealthy patients
        Pattern pat = Pattern.compile(",[123]");
        Matcher mat = pat.matcher("");

        try {
            BufferedReader br = new BufferedReader(new FileReader(input));
            BufferedWriter bw = new BufferedWriter(new
FileWriter("simplified-" + input));
            // Copy header line to output
            bw.write(br.readLine() + "\n");
            String currentLine;
            while ((currentLine = br.readLine()) != null) {
                mat.reset(currentLine);
                // If class 1, 2 or 3
                if (mat.find()) {
                    String lineSplit[] = currentLine.split(",");
                    String id = lineSplit[0];
                    // Change class to 1
                    bw.write(id + ",1" + "\n");
                } else {
                    bw.write(currentLine + "\n");
                }
            }
            bw.close();
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CollectSubset.java

```java
import java.io.*;
import java.util.ArrayList;

/**
 * @author Adam Barron
 * @author 160212899
 * Move all of the accelerometer files for the given subset to a seperate
folder,
 * ready to be unzipped and have GGIR ran on them.
 */
public class CollectSubset {

    private static final File[] DATA_1 = new
File("/data1/biobank/ukbb_1").listFiles();
    private static final File[] DATA_2 = new
File("/data2/biobank/ukbb_2").listFiles();
    private static ArrayList<String> filePaths = new ArrayList<>();

    public static void main(String[] args) {
        if (args.length != 1)
            throw new IllegalArgumentException("Usage: <subset.csv>");
        String subset = args[0];
        ArrayList<String> ids = new ArrayList<>();

        System.out.println("Parsing csv file");
        try {
            BufferedReader br = new BufferedReader(new FileReader(subset));
            // Skip header row
            br.readLine();
            String currentLine;
            while ((currentLine = br.readLine()) != null) {
                String lineSplit[] = currentLine.split(",");
                if (lineSplit.length == 2)
                    ids.add(lineSplit[0]);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        findAccelerometerFiles(ids);

        try {
            File copyScript = createCopyFilesScript(filePaths);
            runScript(copyScript);
        } catch (InterruptedException | IOException e) {
            e.printStackTrace();
        }
    }

    private static void findAccelerometerFiles(ArrayList<String> ids) {
        for (String s : ids) {
            System.out.println("Searching for " + s);
            if (search(DATA_1, s))
                break;
            search(DATA_2, s);
        }
```

```
    }

    private static boolean search(File[] files, String targetId) {
        for (File f : files) {
            if (f.isDirectory()) {
                search(f.listFiles(), targetId);
            } else {
                // Split on "_" to get ID component of file name
                String[] filename = f.getName().split("_");
                String id = filename[0];
                if (id.equals(targetId)) {
                    System.out.println("File found for ID " + targetId);
                    filePaths.add(f.getAbsolutePath());
                    return true;
                }
            }
        }
        return false;
    }

    private static File createCopyFilesScript(ArrayList<String> filePaths)
throws IOException {
        System.out.println("Building script");
        File script = File.createTempFile("copyFiles", "sh");
        Writer streamWriter = new OutputStreamWriter(new
FileOutputStream(script));
        PrintWriter printWriter = new PrintWriter(streamWriter);
        // Make output directory if not present
        printWriter.println("mkdir -p datadir");
        for (String s : filePaths) {
            printWriter.println("cp " + s + " datadir");
        }
        printWriter.close();
        return script;
    }

    private static void runScript(File script) throws IOException,
InterruptedException {
        System.out.println("Building process");
        try {
            ProcessBuilder pb = new ProcessBuilder("bash",
script.toString());
            pb.inheritIO();
            System.out.println("Copying files to /datadir...");
            Process p = pb.start();
            p.waitFor();
            System.out.println("Process completed");
        } finally {
            System.out.println("Deleting temporary file");
            script.delete();
        }
    }
}
```

Appendix D – Python Model

clean_data.py

```
"""clean_data.py - Cleans the biobank dataset ready for fitting
classifiers"""
__author__ = "Adam Barron - 160212899"

import pandas as pd
from pandas_profiling import ProfileReport

# Load features extracted by GGIR
features =
pd.read_csv('resources/part5_personsummary_WW_L30M100V400_T5A5.csv')
# Load class labels
class_labels = pd.read_csv('resources/simplified-subset.csv')
# Merge to add class labels
biobank = features.merge(class_labels, on='id_pla')
# Print data shape
print(biobank.shape)

# Produce data report with Pandas Reporting
report = ProfileReport(biobank, title='Biobank',
html={'style':{'full_width':True}},minimal=True)
report.to_file(output_file='resources/biobank-pandas-profiling-
report.html')

# Drop unneeded metadata columns
biobank.drop(['filename', 'id_pla', 'startday', 'calendardate'], axis=1,
inplace=True)
# Drop constant columns
biobank.drop(biobank.loc[:, biobank.nunique() == 1], axis=1, inplace=True)
# Drop those columns with more than 10% NaN values
biobank.drop(biobank.loc[:, (biobank.isna().sum()) > (biobank.count() *
0.1)], axis=1, inplace=True)

# Sort features by skewness
features = []
skew = []
for col, vals in biobank.iteritems():
    features.append(col)
    skew.append(abs(vals.skew()))
skews_df = pd.DataFrame(list(zip(features, skew)), columns=['Feature',
'Skew'])
print(skews_df.sort_values(by='Skew', ascending=False))
skews_df.to_csv('resources/features_skew.csv')

# Drop highly skewed columns
biobank.drop(biobank.loc[:, (abs(biobank.skew()) > 30)], axis=1,
inplace=True)

# Produce new data report on cleaned data
report = ProfileReport(biobank, title='Biobank-Cleaned',
html={'style':{'full_width':True}}, minimal=True)
report.to_file(output_file='resources/biobank-pandas-profiling-report-
cleaned.html')

# Export cleaned data as new csv
biobank.to_csv('resources/biobank.csv')
```

model.py

```
"""model.py - Machine learning model for analysing the Biobank dataset"""
__author__ = "Adam Barron - 160212899"

import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import time

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn import preprocessing
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import f1_score, roc_auc_score, auc, plot_roc_curve
from sklearn.datasets import load_breast_cancer

# --- Load breast cancer dataset --- #
# X, y = load_breast_cancer(return_X_y=True)
# feature_names = np.array(load_breast_cancer().feature_names)

# --- Load cleaned biobank dataset --- #
biobank = pd.read_csv('resources/biobank.csv')
# Separate features and class labels
feature_names = np.array((biobank.iloc[:, :-1]).columns)
X = biobank.iloc[:, :-1].values
y = biobank.iloc[:, -1].values

# --- Create classifier | comment out as appropriate --- #
classifier = LogisticRegression(max_iter=3000)
# classifier = LinearSVC(max_iter=30000, dual=False)
# classifier = RandomForestClassifier(max_features='sqrt')
# classifier = KNeighborsClassifier()
# --- Set up RFE --- #
rfe = RFE(estimator=classifier, step=0.1)
# KNN cannot perform RFE, used instead
# rfe = RFE(estimator=LinearSVC(max_iter=30000, dual=False), step=0.1)
# --- Create imputer --- #
imputer = SimpleImputer(missing_values=np.nan, strategy='median',
verbose=1)
# --- Create scaler --- #
scaler = preprocessing.RobustScaler()
# --- Construct pipeline --- #
pipeline = Pipeline([("scaler", scaler), ("imputer", imputer), ("rfe",
rfe), ("classifier", classifier)])

# --- Parameter grid for pipeline to search during grid-search cv | comment
out as appropriate --- #
# LogisticRegression
param_grid = [{'rfe__n_features_to_select': [25, 50, 75, 100, 125],
               'classifier__solver': ['sag'],
               'classifier__penalty': ['l2'],
               'classifier__C': [0.1, 1, 10, 100],
```

```
                'classifier__tol': [1e-3, 1e-2]},
               {'rfe__n_features_to_select': [25, 50, 75, 100, 125],
                'classifier__solver': ['saga'],
                'classifier__penalty': ['elasticnet'],
                'classifier__l1_ratio': [0.5],
                'classifier__C': [0.1, 1, 10, 100],
                'classifier__tol': [1e-3, 1e-2]}]
# LinearSVC
# param_grid = [{'rfe__n_features_to_select': [25, 50, 75, 100, 125],
#                'classifier__C': [0.01, 0.1, 1, 10, 100],
#                'classifier__tol': [1e-3, 1e-2]}]
# RandomForestClassifier
# param_grid = [{'rfe__n_features_to_select': [25, 50, 75, 100, 125],
#                'classifier__max_depth': [2, 4, 6],
#                'classifier__min_samples_leaf': [2, 4, 8],
#                'classifier__n_estimators': [100, 200, 300]}]
# KNeighborsClassifier
# param_grid = [{'rfe__n_features_to_select': [10, 20, 30, 40, 50],
#                  'classifier__n_neighbors': [3, 5, 7, 9],
#                  'classifier__weights': ['uniform', 'distance']}]

# Performance metrics
metrics = ['f1_weighted', 'roc_auc']

# Data structures for ROC graph
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)
fig, ax = plt.subplots()

# Model scores
scores_f1 = []
scores_roc = []

# Chosen features
chosen_features = []

# Split data using k-fold
k_fold = KFold(n_splits=5, shuffle=True)
iters = 1
start = time.perf_counter()
for train_idx, test_idx in k_fold.split(X):
    print("Tuning parameters against fold %s of 5" % iters)
    # Create training and test sets for each fold
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]
    # Tune hyper-parameters using grid-search cross-validation

    for metric in metrics:
        print("Tuning parameters against %s" % metric)
        grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=5,
n_jobs=4, scoring='%s' % metric)
        grid_search.fit(X_train, y_train)

        # Best estimator score, parameters and features chosen
        best_estimator = grid_search.best_estimator_
        supports = list(best_estimator.named_steps['rfe'].support_)
        features = feature_names[supports]
```

```
        chosen_features.extend(features)
        print("Chosen features ({0}) for fold {1}: {2}".format(metric,
iters, features))
        print("Best parameters ({0}) for fold {1}: {2}".format(metric,
iters, grid_search.best_params_))
        print("Score ({0}) for fold {1} training set: {2}".format(metric,
iters, grid_search.best_score_))

        # Predict with optimised model
        prediction = best_estimator.predict(X_test)
        # Record score
        if metric in ['f1_weighted']:
            score = f1_score(y_test, prediction, average='weighted')
            print("Score ({0})  for fold {1} test set: {2}".format(metric,
iters, score))
            scores_f1.append(score)
        else:
            score = roc_auc_score(y_test, prediction)
            print("Score ({0}) for fold {1} test set: {2}".format(metric,
iters, score))
            scores_roc.append(score)

    # Record ROC Curve for fold
    viz = plot_roc_curve(grid_search, X_test, y_test, name='Fold {0}
ROC'.format(iters), alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

    iters += 1
# Record time taken
stop = time.perf_counter()

# --- Plot ROC curve for classifier --- #
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Chance',
alpha=.8)
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b', label=r'Mean ROC (AUC = %0.2f $\pm$
%0.2f)' % (mean_auc, std_auc), lw=2,
        alpha=.8)
std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
label=r'$\pm$ 1 std. dev.')
ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05], title='{0} ROC
Curve'.format(classifier.__class__.__name__))
ax.legend(loc="lower right")
# Save graph
plt.savefig('results/{0}_ROC_Curve'.format(classifier.__class__.__name__),
bbox_inches='tight', transparent=True)

# --- Rank features based on how many times they were selected --- #
chosen_features = np.array(chosen_features)
```

```
unique_features, count_features = np.unique(chosen_features,
return_counts=True)
feature_ranking = pd.DataFrame(list(zip(unique_features, count_features)),
                               columns=['Feature',
'Frequency']).sort_values(by='Frequency', ascending=False)
# Save ranking
feature_ranking.to_csv('results/{0}_Feature_Ranking.csv'.format(classifier.
__class__.__name__), index=False)

# Show results
plt.show()

print('\n--------------------Results--------------------')
print("{0}:".format(classifier.__class__.__name__))
print("Feature ranking:\n{0}".format(feature_ranking))
print("F1 score for each test fold: {0}".format(scores_f1))
print("Average F1 score across all test folds:
{0}".format(np.average(scores_f1)))
print("ROC_AUC score for each test fold: {0}".format(scores_roc))
print("Average ROC_AUC score across all test folds:
{0}".format(np.average(scores_roc)))
print("Total time taken to train classifier: {0}".format(stop - start))
print('-----------------------------------------------\n')
```

Appendix E – Statistical Tables

Skewness of features

Feature	Skew
ACC_INB_D30T30_mg_wei	68.02188
ACC_INB_D30T30_mg_pla	68.02187
dur_nightwak_VIG400_min_pla	67.72502
dur_nightwak_VIG400_min_wei	67.72316
ACC_MVPA_D1T100_mg_wei	67.68159
ACC_MVPA_D1T100_mg_pla	67.66362
ACC_INB_D10T30_mg_pla	67.27008
ACC_INB_D10T30_mg_wei	67.26712
ACC_night_mg_wei	65.12137
ACC_night_mg_pla	65.05097
ACC_LIGB_D10T30_100_mg_wei	63.86835
ACC_LIGB_D10T30_100_mg_pla	63.67985
ACC_TVIGday_mg_wei	60.66424
ACC_TSIBday_mg_pla	58.6598
ACC_TSIBday_mg_wei	58.6483
ACC_TVIGday_mg_pla	56.87373
ACC_TINday_min_wei	54.73556
ACC_TINday_min_pla	54.72179
ACC_day_mg_wei	52.47724
ACC_day_VIG400_mg_pla	51.00901
ACC_day_mg_pla	49.01601
L5VALUE_wei	48.88808
L5VALUE_pla	48.88804
quantile_mostactive60min_mg_pla	47.46733
quantile_mostactive60min_mg_wei	47.46244
ACC_INB_D1T30_mg_pla	47.22889
ACC_INB_D1T30_mg_wei	47.22271
ACC_day_VIG400_mg_wei	47.04089
quantile_mostactive30min_mg_pla	46.31346
quantile_mostactive30min_mg_wei	46.30386
dur_TVIGday_min_pla	46.15852
dur_TVIGday_min_wei	46.12029
ACC_nightsleep_mg_wei	43.83839
ACC_nightsleep_mg_pla	43.68991
Nblocks_nightwak_VIG400_wei	43.17857
Nblocks_nightwak_VIG400_pla	42.77307
ACC_nightandday_mg_wei	40.68417
ACC_nightandday_mg_pla	38.74435
M5VALUE_wei	38.68671
M5VALUE_pla	36.55961
Ndaysleeper	21.45461
nonwear_perc_night_wei	14.78535
ACC_LIGB_D1T30_100_mg_pla	13.63355
ACC_LIGB_D1T30_100_mg_wei	13.4604
nonwear_perc_day_pla	12.73273
nonwear_perc_nightandday_pla	12.73265

Feature	Skew
nonwear_hours_wei	12.69982
nonwear_perc_nightandday_wei	12.58869
nonwear_perc_day_wei	12.46499
nonwear_hours_pla	12.17412
nonwear_perc_night_pla	11.57703
dur_day_VIG400_min_pla	11.00991
dur_day_VIG400_min_wei	10.93744
dur_MVPA_D1T100_min_pla	6.971875
dur_MVPA_D1T100_min_wei	6.950678
dur_nightwak_MOD100_400_min_wei	6.078324
dur_nightandday_min_pla	5.998587
window_length_in_hours_pla	5.998587
dur_nightwak_MOD100_400_min_pla	5.916046
acc_wake_wei	5.745706
acc_wake_pla	5.624217
dur_nightandday_min_wei	5.597877
window_length_in_hours_wei	5.597876
dur_LIGB_D10T30_100_min_wei	5.473832
dur_LIGB_D10T30_100_min_pla	5.460993
dur_day_SIB_min_wei	5.177489
dur_day_SIB_min_pla	5.031704
Nblocks_day_VIG400_pla	5.007053
Nblocks_day_VIG400_wei	4.97588
Nblocks_day_SIB_wei	4.397949
dur_nightwak_LIG30_100_min_wei	4.226283
Nblocks_day_SIB_pla	4.159801
dur_nightwak_LIG30_100_min_pla	3.994003
Nbouts_LIGB_D10T30_100_wei	3.983244
Nblocks_TVIGday_pla	3.980968
Nblocks_TVIGday_wei	3.96533
Nbouts_LIGB_D10T30_100_pla	3.956763
Nblocks_LIGB_D10T30_100_wei	3.95102
Nblocks_LIGB_D10T30_100_pla	3.926104
acc_onset_wei	3.739412
acc_onset_pla	3.496132
Nvaliddays_WE	3.261063
Nvaliddays_WD	3.178146
Nvaliddays	2.983815
ACC_nightwak_and_IN30_mg_wei	2.715415
sleep_efficiency_wei	2.52643
ACC_nightwak_and_IN30_mg_pla	2.513879
Nacc_available	2.278037
sleep_efficiency_pla	2.075555
dur_TSIBday_min_wei	1.802302
dur_TSIBday_min_pla	1.745404
Nbouts_MVPA_D1T100_pla	1.738505
Nbouts_MVPA_D1T100_wei	1.736216
Nblocks_MVPA_D1T100_pla	1.73559
Nblocks_MVPA_D1T100_wei	1.732946

Feature	Skew
Nblocks_nightwak_MOD100_400_wei	1.575355
Nblocks_nightwak_MOD100_400_pla	1.536217
N_atleast5minwakenight_wei	1.443946
N_atleast5minwakenight_pla	1.440873
dur_TMODday_min_pla	1.386174
dur_TMODday_min_wei	1.368216
dur_nightwak_and_IN30_min_pla	1.113945
dur_nightwak_and_IN30_min_wei	1.111152
Nblocks_nightwak_LIG30_100_wei	1.096639
ACC_day_MOD100_400_mg_wei	1.06055
ACC_day_MOD100_400_mg_pla	1.058885
Nblocks_TSIBday_wei	1.058223
Nblocks_nightwak_LIG30_100_pla	1.056846
Nblocks_TSIBday_pla	1.05609
Nblocks_nightwak_and_IN30_wei	1.012628
Nblocks_nightwak_and_IN30_pla	0.995196
ACC_nightwak_MOD100_400_mg_wei	0.841527
L5TIME_num_wei	0.810734
sleeplog_wake_wei	0.803392
ACC_nightwak_MOD100_400_mg_pla	0.796288
dur_nightsleep_min_wei	0.773887
dur_nightsleep_min_pla	0.771475
L5TIME_num_pla	0.751268
ACC_TOINday_mg_wei	0.737753
sleeplog_onset_pla	0.732773
ACC_TOINday_mg_pla	0.728319
dur_INB_D30T30_min_wei	0.648494
dur_night_min_wei	0.645606
dur_INB_D30T30_min_pla	0.641457
sleeplog_onset_wei	0.630232
dur_night_min_pla	0.629237
dur_day_min_pla	0.623215
dur_day_min_wei	0.622671
sleeplog_wake_pla	0.569982
ACC_day_LIG30_100_mg_wei	0.562126
dur_LIGB_D1T30_100_min_pla	0.559545
dur_LIGB_D1T30_100_min_wei	0.558836
M5TIME_num_wei	0.550281
ACC_day_LIG30_100_mg_pla	0.548411
dur_day_MOD100_400_min_wei	0.547912
dur_TLIGday_min_wei	0.547276
dur_day_MOD100_400_min_pla	0.545592
M5TIME_num_pla	0.54501
dur_TLIGday_min_pla	0.530804
ACC_TMODday_mg_wei	0.525691
ACC_TMODday_mg_pla	0.524063
Nblocks_day_MOD100_400_wei	0.51903
Nblocks_day_MOD100_400_pla	0.518212
Nblocks_TMODday_wei	0.4996

Feature	Skew
Nblocks_TMODday_pla	0.496473
ACC_TLIGday_mg_pla	0.478556
ACC_TLIGday_mg_wei	0.476194
Nblocks_INB_D1T30_pla	0.468758
Nbouts_INB_D1T30_pla	0.467778
Nblocks_INB_D1T30_wei	0.462953
Nbouts_INB_D1T30_wei	0.461993
Nbouts_LIGB_D1T30_100_pla	0.430098
Nbouts_LIGB_D1T30_100_wei	0.429872
Nblocks_LIGB_D1T30_100_pla	0.429219
Nblocks_LIGB_D1T30_100_wei	0.428992
dur_day_OIN30_min_wei	0.39446
dur_day_OIN30_min_pla	0.393639
Nblocks_day_OIN30_pla	0.382001
Nblocks_day_OIN30_wei	0.381028
dur_INB_D1T30_min_pla	0.379389
dur_INB_D1T30_min_wei	0.378704
Nblocks_INB_D10T30_wei	0.33216
Nblocks_INB_D10T30_pla	0.329649
Nbouts_INB_D10T30_wei	0.317251
Nbouts_INB_D10T30_pla	0.313811
dur_INB_D10T30_min_pla	0.300179
dur_INB_D10T30_min_wei	0.296309
dur_TOINday_min_pla	0.258173
dur_TOINday_min_wei	0.25158
Nblocks_TLIGday_wei	0.18257
Nblocks_TLIGday_pla	0.180274
Nblocks_day_LIG30_100_pla	0.177413
Nblocks_day_LIG30_100_wei	0.173368
Nblocks_nightsleep_pla	0.161502
ACC_day_OIN30_mg_wei	0.156079
ACC_day_OIN30_mg_pla	0.150414
Nblocks_nightsleep_wei	0.13938
dur_TINday_min_wei	0.130633
dur_TINday_min_pla	0.11394
dur_day_LIG30_100_min_pla	0.09655
dur_day_LIG30_100_min_wei	0.094888
ACC_nightwak_LIG30_100_mg_pla	0.092919
Nblocks_INB_D30T30_pla	0.073195
Nblocks_INB_D30T30_wei	0.068621
ACC_nightwak_LIG30_100_mg_wei	0.062075
id_wei	0.041931
Nbouts_INB_D30T30_pla	0.03748
Nblocks_TINday_pla	0.036724
Nblocks_TOINday_pla	0.036296
Nbouts_INB_D30T30_wei	0.03465
Nblocks_TINday_wei	0.034631
Nblocks_TOINday_wei	0.034164

Feature Ranking

Feature	Frequency
dur_TSIBday_min_wei	39
ACC_LIGB_D1T30_100_mg_wei	37
Nbouts_MVPA_D1T100_wei	36
sleeplog_onset_wei	34
sleeplog_onset_pla	34
Nblocks_MVPA_D1T100_wei	34
dur_day_min_wei	33
Nblocks_MVPA_D1T100_pla	32
dur_TLIGday_min_wei	32
dur_TLIGday_min_pla	32
dur_INB_D10T30_min_wei	32
ACC_TMODday_mg_pla	32
dur_day_LIG30_100_min_pla	31
ACC_TLIGday_mg_pla	31
ACC_LIGB_D1T30_100_mg_pla	31
dur_TMODday_min_wei	30
dur_day_OIN30_min_pla	30
ACC_nightwak_LIG30_100_mg_wei	30
Nbouts_MVPA_D1T100_pla	29
Nblocks_INB_D1T30_wei	29
Nbouts_INB_D1T30_pla	28
Nblocks_nightsleep_pla	28
M5TIME_num_wei	28
dur_TINday_min_wei	28
ACC_nightwak_LIG30_100_mg_pla	28
dur_nightwak_and_IN30_min_pla	27
ACC_TMODday_mg_wei	27
Nblocks_nightwak_and_IN30_wei	26
Nblocks_day_MOD100_400_pla	26
dur_TINday_min_pla	26
dur_MVPA_D1T100_min_wei	26
dur_LIGB_D10T30_100_min_wei	26
dur_day_MOD100_400_min_wei	26
ACC_day_LIG30_100_mg_pla	26
Nblocks_day_LIG30_100_wei	25
dur_TOINday_min_pla	25
dur_day_min_pla	25
acc_onset_wei	25
acc_onset_pla	25
Nbouts_INB_D10T30_wei	24
Nblocks_TVIGday_wei	24
Nblocks_TSIBday_wei	24
Nblocks_INB_D10T30_pla	24
dur_TSIBday_min_pla	24

Feature	Frequency
dur_LIGB_D10T30_100_min_pla	24
dur_day_VIG400_min_pla	24
dur_day_MOD100_400_min_pla	24
dur_day_LIG30_100_min_wei	24
Nblocks_LIGB_D1T30_100_pla	23
dur_LIGB_D1T30_100_min_pla	23
dur_INB_D30T30_min_pla	23
acc_wake_pla	23
Nbouts_INB_D1T30_wei	22
Nblocks_LIGB_D10T30_100_pla	22
Nblocks_day_MOD100_400_wei	22
Nblocks_day_LIG30_100_pla	22
N_atleast5minwakenight_wei	22
dur_TOINday_min_wei	22
dur_night_min_wei	22
dur_MVPA_D1T100_min_pla	22
dur_INB_D10T30_min_pla	22
dur_day_VIG400_min_wei	22
Nblocks_nightwak_MOD100_400_pla	21
Nblocks_INB_D30T30_pla	21
Nblocks_INB_D10T30_wei	21
Nblocks_day_SIB_pla	21
dur_TMODday_min_pla	21
dur_nightsleep_min_wei	21
dur_LIGB_D1T30_100_min_wei	21
dur_INB_D1T30_min_wei	21
ACC_TLIGday_mg_wei	21
Nblocks_nightwak_and_IN30_pla	20
Nblocks_INB_D30T30_wei	20
Nblocks_day_VIG400_pla	20
N_atleast5minwakenight_pla	20
dur_INB_D1T30_min_pla	20
acc_wake_wei	20
Nblocks_TSIBday_pla	19
Nblocks_TINday_wei	19
Nblocks_day_SIB_wei	19
M5TIME_num_pla	19
L5TIME_num_wei	19
Nbouts_LIGB_D10T30_100_wei	18
Nblocks_TVIGday_pla	18
dur_nightwak_and_IN30_min_wei	18
dur_INB_D30T30_min_wei	18
ACC_day_OIN30_mg_wei	18
ACC_day_MOD100_400_mg_pla	18
Nblocks_TLIGday_wei	17

Feature	Frequency
Nblocks_nightwak_LIG30_100_wei	17
Nblocks_nightsleep_wei	17
Nblocks_day_OIN30_wei	17
id_wei	17
dur_nightwak_LIG30_100_min_wei	17
dur_nightwak_LIG30_100_min_pla	17
dur_nightsleep_min_pla	17
dur_day_OIN30_min_wei	17
ACC_TOINday_mg_pla	17
Nblocks_TOINday_pla	16
Nblocks_nightwak_LIG30_100_pla	16
dur_nightwak_MOD100_400_min_wei	16
dur_nightwak_MOD100_400_min_pla	16
dur_night_min_pla	16
ACC_day_MOD100_400_mg_wei	16
ACC_day_LIG30_100_mg_wei	16
Nbouts_LIGB_D1T30_100_pla	15
Nbouts_INB_D10T30_pla	15
Nblocks_INB_D1T30_pla	15
Nblocks_day_VIG400_wei	15
L5TIME_num_pla	15
ACC_TOINday_mg_wei	15
ACC_nightwak_MOD100_400_mg_pla	15
ACC_nightwak_and_IN30_mg_pla	15
ACC_day_OIN30_mg_pla	15
Unnamed: 0	14
sleeplog_wake_wei	14
sleep_efficiency_wei	14
Nbouts_INB_D30T30_wei	14
Nblocks_TOINday_wei	14
Nblocks_TMODday_wei	14
Nblocks_day_OIN30_pla	14
dur_day_SIB_min_wei	14
Nblocks_TINday_pla	13
Nblocks_nightwak_MOD100_400_wei	12
dur_day_SIB_min_pla	12
ACC_nightwak_and_IN30_mg_wei	12
sleeplog_wake_pla	11
Nbouts_LIGB_D1T30_100_wei	11
ACC_nightwak_MOD100_400_mg_wei	11
Nbouts_INB_D30T30_pla	10
Nblocks_LIGB_D1T30_100_wei	10
sleep_efficiency_pla	8
Nacc_available	8
Ndaysleeper	7

Feature	Frequency
dur_nightandday_min_wei	7
Nvaliddays_WD	6
Nvaliddays	6
Nblocks_TMODday_pla	6
window_length_in_hours_wei	5
Nbouts_LIGB_D10T30_100_pla	5
Nblocks_LIGB_D10T30_100_wei	5
Nblocks_TLIGday_pla	4
dur_nightandday_min_pla	4
window_length_in_hours_pla	3
nonwear_perc_day_wei	3
nonwear_perc_day_pla	3