Machine Learning for Sepsis Diagnosis Using Omics Data

Edward Parkinson

MSc Artificial Intelligence

School of Computer Science and Informatics

Cardiff University

10 October 2021

Abstract

The clinical signs of sepsis are subtle, and the current diagnosis lacks sensitivity and specificity. Gene-expression based diagnostic approaches have the potential to provide faster and more accurate diagnoses, but to be clinically practical and cost effective, new diagnostic approaches must use a small number of input genes. Here gene-expression data for 63 neonatal infants is used to derive three machine learning based classifiers that predict sepsis in neonatal infants. In contrast to previous analysis of this data set, input genes are not pre-selected with univariate statistical methods; feature selection is incorporated into the machine learning pipeline. The results demonstrate that L1 penalised logistic regression, support vector machines used with recursive feature elimination, and random forests all produce classifiers that predict sepsis infection with high sensitivity and specificity using ten or fewer input genes.

Acknowledgements

I would like to thank Federico Liberatore and W. John Watkins giving for me the opportunity to undertake this project with them, and for their valuable support and guidance throughout.

Contents

1 Introduction **1**

2 Aims and Objectives **3**

3 Background **5**

3.1 Machine Learning Applications in Sepsis Diagnosis 5

3.2 Machine Learning and "Omics" Data 6

3.3 The 2014 Project 6

4 Problem **8**

4.1 Overview 8

4.2 High Dimensional, Low Sample Size Data 8

4.3 Feature Selection 10

4.3.1 Variable Ranking Methods 10

4.3.2 Wrapper Methods 11

4.3.3 Embedded Methods 12

4.3.4 Feature Stability 14

4.4 Nested Cross Validation 14

4.5 Bayesian Optimisation with Gaussian Processes 15

5 Approach **18**

5.1 Data Collection 18

5.2 Data Pre-Processing 19

5.3 Validation Strategy 19

5.4 Machine Learning Model Choices 21

5.4.1 Logistic Regression 21

5.4.2 Support Vector Machines 24

5.4.3 Random Forest 26

5.5 Model Evaluation 27

6 Results and Analysis **31**

6.1 Logistic Regression 31

6.1.1 Constraining the Number of Features 32

6.1.2 Patient Inf075 33

6.1.3 Threshold Selection: Illustration 33

6.1.4 Performance vs Number of Features Trade-Off 36

6.1.5 Logistic Regression: Final Model 36

6.2 Support Vector Machines 37

6.2.1 Performance vs Number of Features Trade-Off 38

6.2.2 Feature Stability 39

6.2.3 SVM: Final Model 40

6.3 Random Forest Results 40

6.3.1 Performance vs Number of Features Trade-Off 41

6.3.2 Feature Stability 41

6.3.3 Random Forest: Final Model 42

6.4 Biological Significance of Selected Genes 42

7 Conclusions **45**

7.1 Critical Evaluation of Results 45

7.2 Further Work 47

8 Reflection **49**

8.1 Learnings from the Research Process 49

8.2 Personal Development 50

8.3 Final Thought 52

Bibliography **53**

A Supporting Python Code **58**

B Code Libraries and Versions **59**

Chapter 1

Introduction

Sepsis is the leading pathway to death from infection of any kind. The World Health Organisation estimates that approximately six million people die of sepsis annually [1]. Speaking at the recent Sepsis Tech and Innovation 2021 conference, Prof. Steven Simpson of University of Kansas presented analysis estimating the annual cost of treating Sepsis to the US healthcare system at a staggering $63bn [2].

The clinical signs of sepsis are subtle, and the current diagnosis lacks sensitivity and specificity. Recent work carried out at Imperial College London as part of the DiAlS study has shown that the introduction of digital sepsis alerts in hospitals (early warning systems used to identify clinical deterioration), reduced the risk of death by 14% [3]. At the same time, a study published in June 2021 by the University of Michigan found that one of the most widely used early warning systems in the US (the Epic Sepsis Model) failed to detect 67% of patients later found to have sepsis [4]. This illustrates the lack of sensitivity in current technologies. The development of faster and more accurate detection systems deployed at scale have the potential to save lives as well as mitigate the overuse of antibiotics and reduce healthcare costs.

Machine learning ("ML") based approaches to the diagnosis of sepsis have been reported by many researchers. These primarily focus on adult patient populations using vital sign, demographic and metabolic patient data as inputs [5]. In contrast, this project uses gene expression data from neonatal infants as the basis for developing a machine learning based classifier for the diagnosis of sepsis, building on work carried out at the university of Edinburgh in 2014 ("the 2014 project") [6].

Chapter 2 will outline the aims and objectives of the project. Chapter 3 provides an brief

overview of the existing research into sepsis diagnosis using machine learning. This is followed in Chapter 4 by a discussion of the nature of the problem being solved, the academic context, why it is difficult and some important methodological background on the techniques used. Chapter 5 details the approach taken to solve the problem and the rationale. The project results and the analysis of those results are the subject of Chapter 6. Chapter 7 summarises the main conclusions of the project and presents ideas for further work. Chapter 8 is a personal reflection on the work, the learnings from it, and areas for future personal development.

Chapter 2

Aims and Objectives

The aim of this project is to develop a ML classifier capable of diagnosing sepsis in neo-natal infants based on the measured expression levels of a small number of genes in their blood. Gene expression can be measured for tens of thousands of individual genes in a given patient at a given time, hence the task is one of 'gene selection' as well as classification. Gene selection studies in biomedicine often have one or both of the following aims [7]:

1. To identify a broad set of relevant genes that help explain the underlying biology of a disease pathway, in this case sepsis. This may include similar genes, whose expression levels are correlated.

2. To identify the minimum subset of genes that provide accurate classification (diagnosis). This is most valuable in a clinical setting, as the smaller the number of genes for which expression levels need to be measured, the faster and cheaper the test. In this case, gene-selection must eliminate redundant genes.

This project builds on the 2014 project which identified a set of 52 genes that both helped explain the underlying pathways involved in sepsis, and provided an accurate classifier [6]. The aim of this project is to identify a minimal set of approximately 5-10 genes capable of reliably distinguishing between healthy and infected neonatal infants. In machine learning terms, this means developing a robust approach to feature selection, as well as a high performing classifier.

In order to achieve this aim, this project will take as input a data set of 63 neonatal infants obtained from the Project Sepsis Research Group at Cardiff University, approximately balanced between healthy and infected patients. The work will test and compare the effectiveness of

alternative ML strategies for feature selection and binary classification. The output will be one or more high performing classifiers, an analysis of their performance, and a description of the required input features.

Chapter 3

Background

This chapter provides a brief overview of the current uses of ML in sepsis diagnosis, and in the broader context of omics data. It concludes with a summary of the previous analysis of the data set used in this project.

3.1 Machine Learning Applications in Sepsis Diagnosis

Over the past decade, ML algorithms have been applied to the problem of sepsis diagnosis in a wide variety of settings. A 2019 meta-analysis of the prediction of sepsis using ML based techniques across seven studies between 2016 and 2018 concluded that ML approaches perform better at predicting the onset of sepsis than traditional clinical scoring tools such as SIRS [1] and SOFA [2] [10]. A 2020 meta-analysis by Fleuren and colleagues also concluded that ML models can accurately predict sepsis onset. Of the 23 papers reviewed, a wide variety of ML approaches were used including SVMs, naïve Bayes, decision trees, and neural networks [5]. Deep neural network architectures such as CNNs, RNNs and LSTMs have also been successfully employed in sepsis predictors [11] [12]. The overwhelming majority of existing studies appear to use a small (fewer than 50) number of features, typically drawn from clinical vital sign, co-morbidity,

[1] The SIRS (systemic inflammatory response syndrome) screening tool identifies patients as being at risk of sepsis where at least two out of body temperature, heart rate, respiratory rate and white blood cell count are outside a defined expected range [8].

[2] The SOFA (Sequential Organ Failure Assessment) score is based on six scores, one for each of the respiratory, cardiovascular, hepatic, coagulation, renal and neurological systems each scored from 0 to 4 with an increasing score reflecting worsening organ dysfunction. A change in the SOFA score of 2 or more is seen as a defining characteristic of sepsis [9].

demographic, and metabolic data, rather than gene expression or other 'omics' data.

3.2 Machine Learning and "Omics" Data

The application of ML methods to the vast amounts of biological data now available across genomics, transcriptomics, proteomics and metabolomics is a very active area of research in disease diagnosis and understanding, particularly in areas such as oncology [13]. The integration of these heterogeneous and high-dimensional data sources gives the potential to develop more powerful ML classifiers for disease diagnosis than currently exist [14]. In the diagnosis of sepsis specifically, gene-expression based diagnostic approaches have recently been developed and validated in clinical settings in collaboration with California-based Inflamatix Inc. who are developing a commercial diagnostic device [15] [16]. The literature on the application of ML techniques to sepsis diagnosis using omics data is however limited, and mainly focuses on adult patients. There is therefore an opportunity to build on the research base in this area to investigate the use of ML approaches more fully with omics data, and in particular in neonatal infants.

3.3 The 2014 Project

The 2014 project focused on the identification of genes that differentiate neonatal infants with a bacterial sepsis infection from healthy neonatal infants. The data set used was recruited from neonatal infants at the Neonatal Unit, Royal Infirmary of Edinburgh and analysed by Smith, Ghazal and colleagues at the University of Edinburgh in 2014 [6].

A common approach in gene selection studies is to select the features (genes), based on their differential expression as a preliminary step prior to classification and without reference to the classification algorithm that is later used. Genes are also often selected based on univariate methods, where the relevance of a gene is estimated in isolation from other genes [7]. Differentially expressed genes are commonly selected using two standard methods. The 'fold-change' method measures the ratio of the absolute value of a gene expression level between two classes, here infected and healthy. Genes are defined as differentially expressed if the ratio exceeds a pre-determined threshold. The 'p-value' approach, provides a probability, a p-value, as the output of a statistical test of the difference in the mean expression level between the two classes

[17].

The 2014 project selected the 52 candidate genes in this way. A statistical cut off of adjusted $P \leq 10^{-5}$ and a quantitative cut off of fold change ≥ 4 were applied to select the 52 genes. Following gene selection, the reduced 52-gene set was modelled using a range of machine learning techniques (random forest, support vector machine, k-nearest neighbour, and logistic regression). The resulting logistic regression based classifier performed well, with an accuracy of 0.98, sensitivity (recall) of 1.00 and specificity of 0.97 [6].

This approach to pre-selecting genes is not necessarily the most appropriate for identifying the minimal set of genes that result in good classification performance, since the selection of the features is independent of the classifier later used to model the data. The chosen features may also be correlated, and interactions between genes are not taken into account [7]. The limitations of univariate feature selection approaches are explained more fully in Section 4.3.1. This project seeks to build on the 2014 project and identify a smaller set of 5-10 genes that can be used to accurately predict neonatal sepsis, using ML techniques that utilise in-built feature selection.

Chapter 4

Problem

4.1 Overview

The challenge in this project is to develop a highly accurate ML classifier capable of differentiating neonatal infants with sepsis ('infected') from healthy neonatal infants ('healthy'). In order to be practically useful in a clinical setting, a diagnostic test based on the gene expression levels must use a limited set of genes (e.g. 5-10), since measuring the expression of each gene increases the time and cost of performing the test. To be clinically credible, the relevance of the genes selected must be verified: the test needs to be explainable based on existing knowledge of the underlying biology [18].

4.2 High Dimensional, Low Sample Size Data

The data set used for the project contains data on 63 neonatal infants, with information on gene expression levels for 48,802 gene probes for each patient. Formally, we have a set of m examples X_k, y_k, where $(k = 1, ...m)$, where $m = 63$, and where there are p variables $x_{k,i}$ where $(i = 1, ...p)$, where $p = 48,802$. The shape of this data set is often referred to as a high dimensional, low sample size (HDLSS) data set.

HDLSS data sets are common when dealing with omics data. Microarray gene sequencing technology produces data sets with tens of thousands of variables, each being the level of expression of a particular gene in a patient. However, the practicalities of recruiting patients

in a clinical setting means that the number examples m is often far lower than the number of observed variables p. The high dimensional nature of the data set presents unique challenges, often referred to as the 'Curse of dimensionality' [19]. In particular:

1. **Data sparsity** - a data set with p dimensions can be thought of as being located in a p-dimensional space. Where $m <<< p$, most of the space is likely to contain no data, i.e. be sparse. This sparsity makes it less likely that a given set of examples is representative of the underlying population.

2. **Multi-colinearity** - where the number of dimensions is larger than the number of examples ($p > m$), at least one variable can be expressed as a linear combination of the others. This means that some of the variables are redundant.

3. **Overfitting** - the higher p, the greater the complexity of the classifier and its ability to model both underlying patterns and noise in the training data.

Choosing a robust approach to feature selection to mitigate these issues is central to this project. An overview of the most relevant techniques is given in Section 4.3.

The very low number of examples also presents significant challenges in model evaluation, given the very limited amount of data available for train, validation and test sets. In order to perform feature selection, hyper parameter tuning and model evaluation, this work will require techniques to maximise the exploitation of the information available in the 63 examples. One such technique is using a nested cross validation strategy for hyper parameter tuning and model evaluation, outlined in Section 4.4.

The combination of complex feature selection approaches (e.g. recursive feature selection described below) and computationally costly cross validation approaches (e.g. leave one out cross validation) has a negative impact on the computational cost and memory requirements of model training, in turn making hyper parameter tuning more expensive. An efficient approach to hyper parameter optimisation will therefore also be valuable in this project. One such approach is Bayesian Optimisation using Gaussian processes, outlined in Section 4.5.

4.3 Feature Selection

Filtering out irrelevant and redundant features is essential to avoid the above issues, in particular overfitting, and to provide faster and less resource-intensive models. In the case of a biological data set, the features selected can be verified for their relevance based on prior expert knowledge of their function. Equally, they may help elucidate the underlying biological processes taking place [20] [18]. Feature selection methods have been characterised into variable ranking methods, wrapper methods, and embedded methods [21]. The following sections briefly outline the principles of each type, and detail the specific methods used in this project.

4.3.1 Variable Ranking Methods

Variable ranking feature selection methods are performed as a pre-processing step, independently of the choice of ML classifier. In a supervised learning classification setting, variables are scored based on their relevance in discriminating between the classes and then ranked. The top ranked features are then selected. The process is computationally efficient as for a set of n variables, n scores are computed [21]. Section 3.1 outlined the variable scoring methods based on p-values for differences in means and fold change that are commonly used in gene selection studies.

Variable ranking methods that select variables according to their individual predictive power of class labels have two significant drawbacks.

1. Multi-colinearity of features remain. It is possible that the selected subset of features will include colinear and therefore redundant features. At the same time, useful features may have been discarded.

2. Variable interactions are ignored. A variable that shows a low predictive power when taken alone may provide significant performance improvement when used along with other variables, as a result of the separation in a higher dimensional space being better than the separation in a single dimension. In this way, complementary genes could be missed from the feature set [21].

That said, given the relative computational efficiency of variable ranking methods, they remain relevant to this project. The variable scoring method used in this work is based on mutual

information, or equivalently information gain. Information gain is chosen for two reasons. Firstly it may be used with discrete and continuous data, as in our classification problem. Secondly, information gain will detect any relationships between the data sets and is insensitive to the number of examples [22]. For each feature in the data set, the information gain between feature vector f_i (again where $i = 1, ...p$ and $p = 48,802$) and target vector y is calculated. The features are then ranked by their score. Information gain is a positive number, the higher the value, the stronger the relationship and it is equal to zero if and only if two features are independent.

Formally, the information gain between variables X and Y is given by Equation 4.1, where $H(X)$ is a measure of the purity of the data set Y and $H(Y|X)$ is a measure of the purity of the data set Y given X.

$$Gain(X;Y) = H(Y) - H(Y|X) \tag{4.1}$$

In the case of binary classification, for a given variable X_i, information gain is given by Equation 4.2, where C is a cost function measuring the impurity of the data set as a function of the probability a of the label being equal to 1.

$$Gain(Y;X_i) = H(Y) - H(Y|X_i) = C(P[y]) - (P[x_1]C(P[y|x_1]) + P[x_0]C(P[y|x_0]) \tag{4.2}$$

The function used in this work to calculate information gain is the scikit-learn function `mutual_info_classif`, which uses entropy as the cost function C, described by Equation 4.3

$$C(a) = -alog_2(a) - (1-a)log_2(1-a) \tag{4.3}$$

This function also uses an approach based on k-nearest neighbour distances to discretize the continuous variables for the purpose of calculating information gain as developed by Ross 2014 [22].

4.3.2 Wrapper Methods

Wrapper or 'search and score' methods take subsets of features and use the prediction performance of a machine learning model to determine the relative usefulness of a given feature

subset. A wrapper method includes a definition of how to search over possible feature subsets, and a methodology for evaluating model performance on each subset, typically via cross validation. Two common approaches to searching over possible subsets are greedy sequential search, and recursive feature elimination (RFE) which has been demonstrated to perform well using high dimensional omics data [18].

The RFE algorithm is a backwards search procedure, starting with all of the input features and pruning features until a minimum subset of features is obtained. The features are pruned by recursively fitting a classifier to the data, and discarding the least relevant features at each iteration.

The relevance of features is determined by a ranking criterion, based on the change to the objective or loss function of the classifier as a result of removing a given feature. The greater the change to the objective function, the more relevant a feature. Assuming a classifier is trained using an objective function J over the training examples, the change in the cost function $DJ(i)$ on removing a feature i is given by Equation 4.4.

$$DJ(i) = (1/2)\frac{\delta_2 J}{\delta \omega_i^2}(D\omega i)^2 \tag{4.4}$$

The RFE procedure works as follows and produces a feature ranking, where feature subsets are nested $F_1 \subset F_2 \subset F_3$.

1. Train the classifier, i.e. optimize weights ω_i with respect to J
2. Compute the ranking criterion for all features ($DJ(i)$)
3. Remove the feature with the smallest ranking criterion

The main drawback of RFE is the computational cost of the algorithm. When features are removed one at a time, complexity scales in proportion to the number of features. In our case, with over 48,000 features, computational cost is a potential issue with this approach.

4.3.3 Embedded Methods

Embedded methods perform feature selection during the training of a predictor, and are therefore typically specific to a given learning method. Examples of embedded methods are L1

regularisation (LASSO) in linear and logistic regression models, and feature selection based on feature importance in random forests [23] [24].

The cost function in a logistic regression model J, over m training examples, with respect to weights vector θ is given by Equation 4.5:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x_i), y_i) \tag{4.5}$$

Where the cost contribution from a given example is described by Equations 4.6 and 4.7.

$$Cost(h_\theta(x), y) = -y.log(h_\theta(x)) - (1-y)log(h_\theta(x)) \tag{4.6}$$

Where:

$$h_\theta(x) = \frac{1}{1 + e^{-}(\theta^T x)} \tag{4.7}$$

Regularisation adds a penalty term to the cost function $J(\theta)$ to constrain the magnitude of the weights. L1 regularisation adds a penalty based on the magnitude of the feature coefficients. In this case, the coefficient of a given feature may be set to zero, resulting in the removal of the feature from the model. The cost function, including the L1 penalty is given by Equation 4.8.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x_i), y_i) + \lambda \sum_{j=1}^{k} |\theta_j| \tag{4.8}$$

The parameter λ determines the size of the penalty and k is the number of features. This illustrates how the feature selection step is 'embedded' within the minimisation of the cost function in the logistic regression model. By increasing the value of λ, the L1 penalty is increased, reducing the coefficients of the features, and resulting in more features having a zero coefficient, i.e. more features are removed.

Decision tree based models also have embedded feature selection methods. As an example, in a random forest classifier, the feature importance of a given feature is calculated based on the mean information gain (also described as the "mean decrease in impurity") over all nodes in a decision tree, weighted by proportion of samples reaching that node, averaged over all trees of the ensemble [25]. Information gain for a given node is calculated as described previously in Equation 4.2. In the case of calculating feature importance, the cost function used to measure purity in this work is the Gini index, an alternative to entropy, and given by Equation 4.9.

$$C(a) = 2a(1-a) \tag{4.9}$$

Given the very large number of features in our dataset, the aim in this project is to avoid variable ranking based methods where possible, and instead use methods that select subsets of variables together based on their predictive power, while at the same time yielding a compact set of features that provide good classification performance with the chosen models.

4.3.4 Feature Stability

A further challenge of working with such a high dimensional data set is the 'stability' of the selected feature set. The objective of this work is not only to identify a minimal set of features for the diagnosis of sepsis, but also a *meaningful* set of features that have underlying biological validity. The stability of the feature selection procedure relates to how the chosen feature subset varies with different samples drawn from the training data. If the feature subset changes dramatically for small changes in the training data, the model could be described as unstable. If the feature subset is invariate to changes in the training data, it could be described as stable [26].

4.4 Nested Cross Validation

The wrapper and embedded methods for feature selection described above both require a cross validation step to evaluate the relative performance of alternative feature subsets. A common challenge with high dimensional, low sample size data sets is having sufficient examples to perform robust feature selection and hyper parameter tuning separately from model evaluation. The tuning of hyper-parameters and selection of features cannot be performed on the same validation set, as this will lead to biased estimates of performance [27].

To solve this, nested cross validation is often applied to small data sets. The general nested cross validation scheme is illustrated in Figure 4.1. The data set is split into train and test sets (the outer CV loop). A k-fold cross validation is applied to the train set (the inner CV loop). The entire feature selection and hyper parameter optimisation pipeline is applied to the inner CV loop, and the optimum features and parameters selected. In the interests of deriving the least biased estimate of model performance, leave one out cross validation can be used in

this inner loop. A model is then retrained on the full train set using the chosen features and parameters. This model is then used to make predictions on the test set, and the predictions scored.

With a low number of examples such as in our case, there may still be high variance in the model performance depending on the examples in the train and test sets. The address this, the generalisation error of the model is estimated by averaging the model performance scores of the test set over multiple train / test splits (the outer loop).

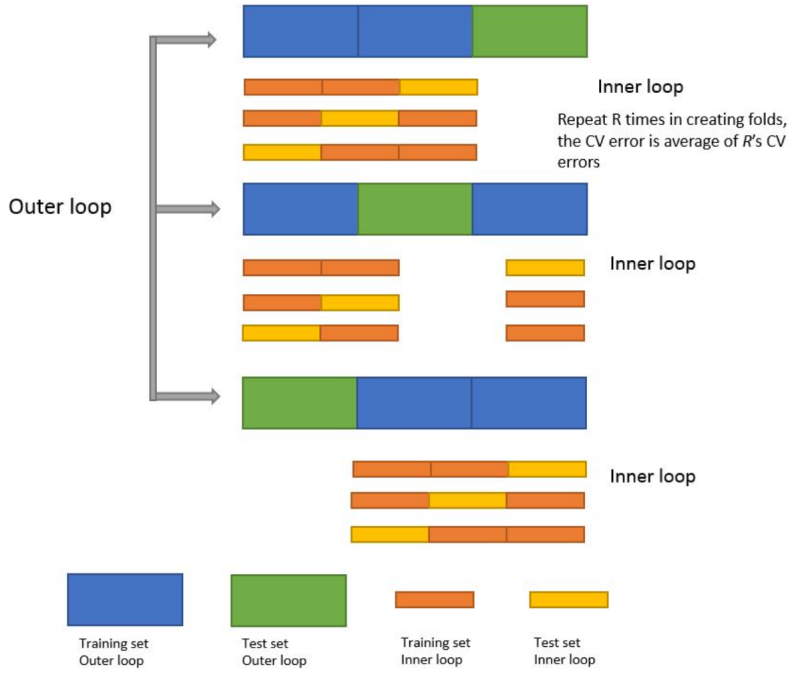

Figure 4.1: Illustration of Nested Cross Validation [28]

4.5 Bayesian Optimisation with Gaussian Processes

The combination of nested cross validation with hyper parameter optimisation in the inner loop adds to the complexity of the models, impacting run times and resource requirements. Grid search based hyper parameter turning can be time consuming and potentially test a large number of poorly performing parameters. To improve model optimisation times, another important methodology used in this project is Baysian Optimisation with Gaussian Processes.

Bayesian optimisation using Gaussian processes is an alternative approach to searching the hyper-parameter space. Rather than testing all possible values as in a grid search, a Bayesian search identifies the most promising next value of a parameter to test, based on the performance

of previous values, relative to some objective function.

Given an objective function $f(x)$ that provides and evaluation of a ML model trained on features x, Bayesian optimisation is a machine-learning optimisation methodology that solves the problem:

$$x^* = \arg\max_x f(x) \tag{4.10}$$

In the problem addressed in this project, the set A of possible values of x, and the objective function f have the following properties [29] [30]:

1. The set of variables A, (in our case hyper parameters and or numbers of features) is a simple set where $x \in R^d; a_i \leq x_i \leq b_i$.

2. The objective function f is continuous and expensive to evaluate, in particular time consuming given the number of potential values of x, and complexity of the function.

3. f is a 'black box' meaning that there is no closed form solution and it is not differentiable; therefore it can not be solved easily with techniques such as gradient descent.

The task of hyper parameter optimisation where there is a complex objective function, such as a nested cross validation procedure, evaluating the accuracy of a machine learning model, and where there is a large potential search space, making $f(x)$ expensive to evaluate, fits the above criteria.

There are two components to the Bayesian optimisation process. Firstly a statistical model (a Gaussian process) for modelling the objective function that calculates a probability distribution for the potential values of $f(x)$ at given of x. Secondly an 'acquisition function', that calculates the expected improvement in the objective function at each potential value of x, and therefore determines which value of x is selected for the next iteration of the process. The acquisition function selects the value of x that provides the greater expected improvement in $f(x)$, based on the current probability distribution.

The iterative process can be summarised as follows. The number of iterations of the process T is defined ahead of time [29]:

For t = 1,..., T:

1. Given the set of observations $(x_i, y_i = f(x_i))$ for $i = 1, ..., t$, build a statistical model for the objective function f using Gaussian process regression

2. Use a computationally inexpensive 'surrogate' acquisition function $u(x)$ based on the posterior probability distribution of f to calculate the next point x based on the expected maximum improvement in $f(x)$.

$$x_{t+1} = \arg\max_{x} u(x)$$

3. Calculate the next observation y_{t+1} at x_{t+1}

The process therefore explores the search space (e.g. hyperparameter space) by progressively estimating values of the parameters with the maximum expected improvements in the objective function (e.g. model performance). This means a wider search space can be evaluated in a given time, as less valuable regions are ignored. This is in contrast to a grid search or randomised search procedure where possible hyperparameter values are treated equally, and potentially large numbers of poorly performing values would have to be evaluated to achieve the same result, at significant time cost. A more detailed mathematical discussion of Bayesian optimisation is given in [30].

Chapter 5

Approach

The project will apply a number of widely used machine algorithms to the neonatal data set, incorporating feature selection into the pipeline to build an accurate classifier. The main algorithms under consideration at the outset were L1 regularised logistic regression, linear support vector machines and random forests.

5.1 Data Collection

The data set contains gene-expression level information on 63 neonatal infants. The produce the data set, the RNA collected from the infected and control infants was hybridized onto Illumina Human Whole-Genome Expression BeadChip HT12v3 microarrays comprising 48,802 features (gene probes). The raw data from 63 samples was transformed using standard techniques when using microarray outputs - variance stabilizing transformation and robust spline normalization - designed to remove systematic variation between samples resulting from the process of measurement rather than underlying biological differences. The examples are labelled as either infected (indices with prefix 'Inf') and healthy (indices with prefix 'Con'), based on clinical assessment of the patients at the time of data collection. The infected patients were confirmed to have a bacterial infection, save one virally infected patient with index Inf075 [6].

5.2 Data Pre-Processing

Data pre-processing consisted of three steps. Firstly, the data set was de-duplicated to that there were no duplicate features. The data was de-duplicated based on the column `Probe_Id`, the unique identifier in the dataset that maps to an individual gene probe. A single duplicate feature was removed. Secondly, the data was searched for NaN values: none were identified. Finally, each feature vector x_i was standardised (or 'z-scored'), meaning all 63 values for each feature were re-scaled to give a mean of 0 and standard deviation of 1. L1 regularisation (in logistic regression and SVM models) requires features to be on the same scale. The L1 penalty uses the magnitude of the coefficient of each feature to determine the penalty - having features on different scales would result in features on larger scales dis-proportionally impacting the penalty term, and biasing the selection of features.

5.3 Validation Strategy

Given the very low number of examples, a nested cross validation strategy was implemented, as described in Section 4.4.

The outer cross-validation loop, used for model selection and performance evaluation, uses a 3-fold split, resulting in three sets of training examples of 42 examples, and three corresponding test sets of 21 examples. This was implemented as a repeated k-fold cross validation to enable repeating experiments over multiple train-test splits, while ensuring that each example is contained in the test set every three iterations. Performance metrics could then be averaged over all iterations to reduce the impact of variance in the scores.

Where cross validation is needed for the purpose of feature selection (such as in the RFE procedure) or hyper parameter tuning, the inner cross validation loop implemented leave-one-out cross validation to provide the most reliable estimate of model performance; a worthwhile trade-off with increased computational cost given the small sample size. The resulting 42-fold cross validation, has 41 training examples and one validation example for each fold. The inner cross validation procedure was manually coded to allow the predictions for the validation examples to be stored and then scored across all folds. Standard python cross validation functions such as `GridSearchCV` do not facilitate this, as they calculate metrics on each fold and then average across folds, giving errors when using a leave one out strategy. The code for the

function that performs the outer cross validation loop and model evaluation is given in Listing 5.1. Examples of the code implementing the inner cross validation loops are given in Sections 5.4.1 and 5.4.2.

Listing 5.1: Nested Cross Validation

```
def nested_cv_lr(df, labels, n_repeats):

    # configure the cross-validation procedure
    cv_outer = RepeatedKFold(n_splits=3, n_repeats=n_repeats)

    # containers to collect scores from each outer loop iteration
    scores_list = []
    best_parameters_list = []
    mis_classified_list = []
    selected_features_list = []

    # create outer cv loop for model evaluation
    for train_ix, test_ix in cv_outer.split(df):

        # split data into train and text sets
        X_train, X_test = df.iloc[train_ix, :], df.iloc[test_ix, :]
        y_train, y_test = labels[train_ix], labels[test_ix]

        # execute the inner cv procedure to return trained model
        model, best_parameter = train_lr_baysian(X_train, y_train)

        # predict on the test set and store the selected features
        y_pred = model.predict(X_test)
        y_pred_probab = model.predict_proba(X_test)

        # score predictions, mis-classified examples, and features
        scores = score_model(y_pred, y_pred_probab, y_test)
        mis_classified = mis_class_points(X_test, y_pred, y_test)
        selected_features = model.coef_[0]

        # add scores etc. to containers
        scores_list.append(scores)
        best_parameters_list.append(best_parameter)
        mis_classified_list.append(mis_classified)
```

```
        selected_features_list.append(selected_features)

    # return performance data for all interations
    return  scores_list,
            best_parameters_list,
            mis_classified_list,
            selected_features_list
```

5.4 Machine Learning Model Choices

The choice of the machine learning algorithms is determined by the nature of the problem.

- Logistic regression was selected as a baseline model for its simplicity, interpretability, and the availability of L1 regularisation as an embedded feature selection method.
- SVMs were chosen given their strong performance in classification tasks and prior success in classification tasks using omics data, particularly when combined with recursive feature elimination [18].
- Random Forests were selected again for their well documented performance in classification tasks, their speed and interpretability of the embedded feature selection.

5.4.1 Logistic Regression

Logistic regression was implemented using scikit-learn's built in `LogisticRegression` class, using an L1 penalty, and the `liblinear` solver. The regularisation parameter C was optimised using a Bayesian Optimisation with Gaussian process, using the `gp_minimize` function from the scikit-optimize library.

The objective function to be maximised was defined as the accuracy of classification of the inner leave-one-out cross-validation loop. The number of iterations of the Bayesian optimisation was determined empirically to ensure convergence, i.e. a maximum accuracy. In this formulation, the number of features selected is an output of the process of maximising model performance. The code that performs the model training and hyper parameter optimisation via the inner cross-validation loop is given in Listing 5.2.

Listing 5.2: Bayesian Optimisation - Optimising Classification Accuracy

```
def train_lr_baysian(X_train, y_train):

    # define hyper parameter search space
    hyper_p_c = Real(low=1e-6, high=1000.0, prior='log-uniform', name='C')
    search_space_lr = [hyper_p_c]

    # define the objective function
    @use_named_args(search_space_lr)
    def evaluate_model(**params):

        # create inner cv loop for hyperparameter optimisation
        cv_inner = LeaveOneOut()
        y_val_classes = []
        y_val_predictions = []
        for train_index, val_index in cv_inner.split(X_train):

            X_train_inner, X_val = X_train.iloc[train_index, :],
                                   X_train.iloc[val_index,   :]
            y_train_inner, y_val = y_train[train_index], y_train[val_index]

            # instantiate and fit the predictor
            model = LogisticRegression(penalty='l1', solver="liblinear")
            model.set_params(**params)
            model.fit(X_train_inner, y_train_inner)

            # predict validation set
            y_val_pred = model.predict(X_val)
            y_val_classes.append(y_val)
            y_val_predictions.append(y_val_pred)

        # score the predictions on validation set over all folds
        score = accuracy_score(y_val_classes, y_val_predictions)

        return 1.0 - score

    # perform optimization and store the optimum hyperparameter
    result = gp_minimize(evaluate_model,
                         search_space_lr,
```

```
                        acq_func="EI",
                        x0=[1.0],
                        n_initial_points=20,
                        n_calls=30)

best_parameter = result.x

# retrain the predictor using the optimum hyperparameter
model = LogisticRegression(C=best_parameter[0],
                           penalty='l1',
                           solver="liblinear")
model.fit(X_train, y_train)

# return trained model for scoring
return model, best_parameter
```

An alternative approach was also evaluated, where a simplified objective function targeted a fixed number of features in the final model, irrespective of the performance of the model. The rationale for this approach was to ensure that the model produced has a limited set of features, mitigating the risk of over fitting. In this case, there is no inner cross validation loop, as the number of features in the trained model can be measured without reference to unseen data in a validation set. This significantly reduced training times. The code for this simplified objective function is given in Listing 5.3.

Listing 5.3: Bayesian Optimisation - Optimising Fixed Number of Features

```
# define the target number of features
target_features = n_features

# define the objective function
@use_named_args(search_space_lr)
def evaluate_model(**params):

    # instantiate and fit the predictor
    model = LogisticRegression(penalty='l1', solver="liblinear")
    model.set_params(**params)
    model.fit(X_train, y_train)

    # calculate number of features compared with target
    n_nonzero = np.sum(model.coef_ != 0)
```

```
        return (target_features-n_nonzero)**2

    # perform optimization
    result = gp_minimize(evaluate_model,
                         search_space_lr,
                         acq_func="EI",
                         x0=[1.0],
                         n_initial_points=20,
                         n_calls=30)
```

In both of the logistic regression approaches descrived above, following hyperparameter optimisation, the final model was retrained using the optimum hyperparameter evaluated on the test set. The whole procedure was repeated (via the outer cross validation loop) and scores averaged over multiple iterations of train/test split and average performance recorded. The 'final model' was then trained on the full dataset (train and test sets) and the selected features identified.

5.4.2 Support Vector Machines

A linear support vector machine classifier with L1 regularisation was initially considered as a baseline for performance. This approach used a similar procedure to that shown in Listing 5.3.

As a second step, a linear SVM was combined with RFE to reduce the set of features used in the final model. The RFE procedure was evaluated using leave-one-out cross validation in the inner loop. As discussed previously, the computational cost of RFE is high, and scales linearly with the number of features n. In the case of 48,202 features, eliminating one feature at a time to select a final set of 20 features, RFE would involve fitting over 48,000 SVM models for each fold of the cross validation. In order to reduce the computational cost to a reasonable level, two compromises were introduced: (i) a variable ranking feature selection method was introduced prior to performing RFE, based on information gain. This was used to reduce the number of features to 250. (ii) During RFE, 10% of the features were discarded after each fit of the model, reducing the number of fit models required to 24, in order to select 20 features from 250. The variable ranking feature selection and RFE were included in the same pipeline, along with hyperparameter tuning, and performed within the inner cross validation loop, to ensure no data leakage from the initial feature selection steps to the test set.

Similarly to the case of logistic regression, hyperparameter optimisation was achieved using Bayesian optimisation with Gaussian processes, whereby the objective function solved for the classification accuracy of the pipeline. The full pipeline is illustrated in Listing 5.4

Listing 5.4: SVM Pipeline

```
def train_svm_baysian(X_train, y_train, out_features):

    # define hyper paramaeter search space
    hyper_p_c = Real(1e-6, 100.0, 'log-uniform', name='C')
    search_space_svm = [hyper_p_c]

    # define the objective function
    @use_named_args(search_space_svm)
    def evaluate_model(**params):
        cv_inner = LeaveOneOut()
        y_val_classes = []
        y_val_predictions = []

        # create inner cv loop for hyperparameter optimisation
        for train_index, val_index in cv_inner.split(X_train):

            X_train_inner, X_val = X_train.iloc[train_index, :],
                                   X_train.iloc[val_index, :]
            y_train_inner, y_val = y_train[train_index], y_train[val_index]

            # pipeline for recursive feature elimination and model training
            svc = SVC(kernel="linear", probability=True)
            svc.set_params(**params)
            rfe = RFE(estimator=svc,
                      n_features_to_select=out_features,
                      step=0.1)
            svm_pipeline = Pipeline(steps=[('rfe', rfe),
                                           ('model', svc)])
            svm_pipeline.fit(X_train_inner, y_train_inner)

            # predict validation set
            y_val_pred = svm_pipeline.predict(X_val)
            y_val_classes.append(y_val)
            y_val_predictions.append(y_val_pred)
```

```
        # score the predictions on validation set over all folds
        score = accuracy_score(y_val_classes, y_val_predictions)
        return 1.0 - score

    # perform optimization and store optimum hyperparameter
    result = gp_minimize(evaluate_model, search_space_svm, n_calls=20)
    best_C = result.x[0]

    # retrain the predictor using the optimum hyperparameter
    clf = SVC(kernel="linear", C=best_C, probability=True)
    feature_elim = RFE(estimator=clf,
                       n_features_to_select=out_features,
                       step=0.1)
    model = Pipeline(steps=[('feature_elim', feature_elim),
                            ('clf', clf)])
    model.fit(X_train, y_train)

    # return trained model for scoring
    return model, best_C
```

As with Logistic Regression, following hyperparameter optimisation, the final model was retrained using the optimum hyperparameter evaluated on the test set. Again, the whole procedure was repeated and scores averaged over multiple iterations of train/test split and average performance recorded. The 'final model' was then trained on the full dataset (train and test sets) and the selected features identified.

5.4.3 Random Forest

A Random forest classifier was implemented using scikit-learn's built-in `RandomForestClassifier` class. A random forest model is expected to be less sensitive to hyperparameter optimisation [7], therefore a simpler grid search was used over hyper parameter space to validate sensitivity to key hyper parameters, rather than the Bayesian optimisation process used with logistic regression and SVMs.

The training time complexity of a random forest is given by Equation 5.1

$$O(k \cdot d \cdot n \cdot log(n)) \tag{5.1}$$

Where n is the number of examples, d is the number of features evaluated at each tree and k is the number of trees used to build the forest. Given the very small sample size, the number of estimators (trees) used to build the forest was tested at 500, and 1,000 estimators. To maintain a reasonable complexity cost, the maximum number of features used to build each tree was tested in the range 100 - 500, balancing classification performance with training time. Feature selection was performed during the optimisation using scikit-learn's `SelectFromModel` class, selecting a fixed number of features based on the calculated feature importance. A final random forest classifier was then trained on these features only.

The random forest model was run with sample bootstrapping with replacement, enabling scoring of each set of hyper parameters and corresponding selected features using the 'out of bag' accuracy score. Out of bag scoring eliminated the need for cross-validation in the inner loop, with significant savings in training time.

Again, following hyper parameter optimisation, the full train set was retrained with the same pipeline using the selected hyperparameters and scored against the test set. As previously, the entire procedure was repeated and scores averaged over multiple iterations of train/test split and average performance recorded. The final model was then trained on the full data set (train and test sets) and the selected features identified.

5.5 Model Evaluation

The problem of diagnosing infected from healthy patients is in this case a binary classification problem. A range of metrics are widely used to measure classification performance in this setting. Terminology and metric preferences vary between machine learning and medical statistics literature and so a brief overview of the important metrics used in this project is given here. The metrics are best understood in terms of a confusion matrix, as illustrated in Figure 5.1

Accuracy measures the ratio of correct predictions to all predictions in the test set, regardless of whether positive (infected) or negative (healthy). Where the classes are unbalanced, accuracy may give an unrealistically positive view of performance. For example an accuracy score of 0.95 may sound high, however if one class represents 5% of examples, then all of this class could be

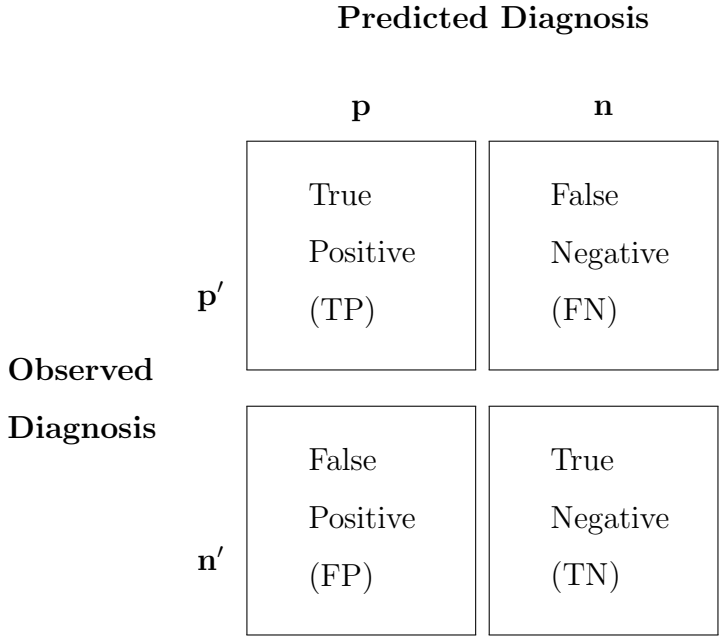

Figure 5.1: Confusion Matrix

mis-classified - i.e. the predictor has no power. Our neonatal data set is artificially imbalanced since the overwhelming majority of real world neonatal infants will be healthy. Accuracy is used for hyper parameter tuning in our case, given the data set is balanced, and correct classification is a reasonable measure when comparing between alternative hyper parameters. Accuracy is recorded for model selection, however it is noted that it is not the most appropriate metric for this task.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Sensitivity, also referred to as **Recall** or the **True Positive Rate** measures the ratio of all correctly predicted positive cases to all true (observed) positive cases. In a condition like sepsis, where cases are easily missed and the mortality rate is very high, sensitivity is one of the most important measures. A diagnostic tool must have a very high sensitivity. That said, high sensitivity must be balanced with false positives. In the real world, a high false positive rate has been shown to result in 'alert fatigue' among clinicians where results of tests are ignored [4]. In addition, administration of unnecessary antibiotics to neonatal patients increases the risk of life threatening necrotizing enterocolitis [6].

$$Recall = \frac{TP}{TP + FN}$$

Specificity, the corollary of recall for the healthy patients, measures the ratio of all correctly predicted negative cases to all true (observed) negative cases. Specificity is particularly useful when viewed in combination with sensitivity. **False Positive Rate**, or $(1 - specificity)$ is used in plotting the ROC curve described below.

$$Specificity = \frac{TN}{TN + FP}$$

$$FalsePositiveRate = 1 - Specificity = \frac{FP}{TN + FP}$$

Precision, also referred to as **Positive Predictive Values (PPV)** measures the ratio cases correctly predicted as positive to all cases predicted as positive. Precision provides an alternative measure of the impact of false positives in the predictions.

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

The Receiver Operating Characteristic ("ROC") Curve is widely used in medical diagnostic binary classification settings. The ROC curve is a probability curve constructed by plotting the True Positive Rate (Recall or Sensitivity) against the False Positive Rate of a classifier. Plotting the ROC curve requires a classifier to output the probability of each example in the test belonging to the positive class. The curve plots the TPR and FPR at different threshold probabilities for the positive class. In the case where the classes can be perfectly separated by the classifier, then the probability for all examples classified as the positive class would be above the threshold, and all probabilities for the negative class would be below the threshold.

The Area Under the ROC Curve (referred to as ROC - AUC) is a measure of how well a classifier is able to separate the classes. An AUC of 1.0 implies that the classes can be perfectly separated at some threshold probability. An AUC of 0.5 implies that a model has no ability to separate the classes.

The ROC curve can be used to determine the trade-off between false positive and false negatives in a given classifier. In the sepsis diagnosis setting, the cost of false positive and false negatives may not be the same - the trade off is the risk of death from sepsis, weighed against the risk of complications from the unnecessary use of antibiotics. The ROC curve can be used to determine the threshold probability for a given model that provides the best balance between

false positives and false negatives.

All of the above metrics will be reported in the results. Given the application context, the most important measure in this work is likely to be AUC. AUC is independent of the threshold probability separating the classes, and takes into account the impact of both false negatives and false positives.

Chapter 6

Results and Analysis

This section outlines the results achieved with each of the three main classification algorithms tested, along with the corresponding genes identified during the feature selection process.

6.1 Logistic Regression

The initial logistic regression model was trained using a repeated 3-fold cross validation strategy, repeated three times, resulting in nine train/test split iterations. As discussed in Section 5.4.1, the L1 regularisation parameter C was tuned using an inner leave-one-out cross validation loop on the training set, and the value of C giving the highest accuracy score on the validation set selected. The full training set was then retrained using this value of C and this trained model then used to make predictions on the test set, which were then scored. Table 6.1 shows the results for all nine train/ test splits including the value of C and number of features selected for each iteration. The mean scores over all iterations are also given.

On first inspection the performance metrics appear promising. The AUC ranges from 0.91 - 1.00 across the nine iterations, with a mean of 0.98. However, the selected L1 regularisation parameter varies widely between iterations (ranging from 0.08 to 34.88). The number of features selected at each training round also varies widely. There appears to be a positive correlation between the magnitude of the regularisation parameter C and the number of features (the lower C, the fewer features) as expected.

In order to identify the selected features, the full data set (train and test sets) was used to train the model, the features extracted and the genes identified. The final trained model produced

Iteration	Best C	Num Features	Accuracy	Recall	Specificity	Precision	AUC
1	1.00	101	0.95	0.88	1.00	1.00	0.98
2	0.08	2	0.90	1.00	0.82	0.83	0.98
3	34.88	210	0.95	1.00	0.91	0.91	1.00
4	1.00	68	0.81	1.00	0.67	0.69	0.91
5	1.28	91	0.95	0.89	1.00	1.00	0.95
6	5.48	139	1.00	1.00	1.00	1.00	1.00
7	0.09	4	0.90	1.00	0.78	0.86	1.00
8	12.16	177	1.00	1.00	1.00	1.00	1.00
9	1.00	64	0.90	0.90	0.91	0.90	0.98
Mean	n/a	95	0.93	0.96	0.90	0.91	0.98

Table 6.1: Model Performance: Optimising Classification Accuracy

non-zero coefficients for 89 different features on the first run, and 336 features on a subsequent run. This large number of features relative to the number of examples in the data, the high variance of the hyper parameter C and the number of features selected over multiple training iterations, combined with the near perfect classification of the test sets, indicate that this model may be over fitting the data - the curse of dimensionality strikes!

This approach of optimising the validation set for classification performance is not regularising the logistic regression model strongly enough to mitigate over fitting or to identify a small and consistent set of features that could have underlying biological validity.

6.1.1 Constraining the Number of Features

In order to force a stronger L1 penalty, and therefore greater sparsity in the output feature set, an alternative approach was tested, this time setting the objective function within the Bayesian optimisation to optimise for a specific number of features, as detailed in Section 5.4.1.

The model was initially trained to optimise for 30 features. Nine iterations of train and test sets produced the results in Table 6.2. These results also detail the specific examples mis-classified in the test set at each iteration.

These results show a regularisation hyper-parameter C of between 0.15 and 0.45 across all

Iteration	Best C	Num Features	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	0.40	49	0.95	0.83	1.00	1.00	0.98	-	Inf_149
2	0.27	33	0.90	0.88	0.92	0.88	0.99	Con_165a	Inf075
3	0.21	24	0.95	1.00	0.86	0.93	1.00	Con_206	-
4	0.19	24	0.95	1.00	0.92	0.89	0.98	Con_165a	-
5	0.27	39	0.95	0.80	1.00	1.00	0.98	-	Inf_149
6	0.45	38	0.95	0.93	1.00	1.00	0.99	-	Inf075
7	0.15	14	1.00	1.00	1.00	1.00	1.00	-	-
8	0.24	31	0.86	0.88	0.85	0.78	0.94	Con_165a, Con_206	Inf075
9	0.34	38	1.00	1.00	1.00	1.00	1.00	-	-
Mean	n/a	32.0	0.95	0.92	0.95	0.94	0.98	n/a	n/a

Table 6.2: Model Performance: Optimising for 30 Features

iterations, significantly more stable than previously. The performance of the model is good, with a mean recall of 0.92, specificity of 0.95 and AUC of 0.98. The number of features selected in each training iteration shows significantly less variance than previously. However, the number of selected features still varies across iterations, with a mean of 32. These results give significantly more confidence that the final model is indeed a well performing classifier that is likely to generalise to unseen data, rather than simply over fitting the training data.

6.1.2 Patient Inf075

As noted in Section 5.1, patient Inf075 was identified clinically as having a viral rather than bacterial infection. Table 6.2 shows that patient Inf075 was incorrectly classified as healthy in three iterations of the model, and was the example most frequently classified as a false negative. Given this result and the uncertainty around the label of this example, the analysis was repeated excluding patient Inf075, yielding the results in Table 6.3. These results show an improved performance across all metrics. In particular the average AUC is 1.00 across all nine iterations, indicating that this model provides near perfect classification. Again, the hyperparameter C is relatively stable across iterations, however the number of features selected, while averaging 27 across iterations, shows some variance between iterations.

6.1.3 Threshold Selection: Illustration

It is important to note that accuracy, recall, specificity and precision are all calculated using the predicted classes of the test set examples. The class prediction is made relative to a pre-defined

Iteration	Best C	Num Features	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	0.18	21	1.00	1.00	1.00	1.00	1.00	-	-
2	0.13	21	0.95	1.00	0.93	0.88	1.00	Con_206	-
3	0.12	12	0.95	1.00	0.92	0.89	1.00	Con_165a	-
4	0.33	29	1.00	1.00	1.00	1.00	1.00	-	-
5	0.23	31	1.00	1.00	1.00	1.00	1.00	-	-
6	0.23	34	1.00	1.00	1.00	1.00	1.00	-	-
7	0.45	36	0.95	1.00	0.93	0.86	1.00	Con_165a	-
8	0.14	15	1.00	1.00	1.00	1.00	1.00	-	-
9	0.26	47	1.00	1.00	1.00	1.00	1.00	-	-
Mean	n/a	27.0	0.98	1.00	0.98	0.96	1.00	n/a	n/a

Table 6.3: Model Performance: Optimising for 30 Features (Excluding Patient Inf075)

threshold probability separating the classes. In this report these metrics are calculated with the commonly used default threshold of 0.5. AUC does not assume a threshold, as the ROC curve describes model performance over the full range of thresholds between 0 and 1.

As an illustration of the impact of the choice of threshold, the results in Table 6.3 show iteration number three of the model has an AUC of 1.00 (implying perfect classification), but an accuracy of 0.95, recall of 1.0 and a precision of 0.89, (implying mis-classified false positive examples). Figure 6.1 plots the predicted probability of being in the positive class for each of the 20 test set examples in this third iteration of the model, against the true class labels. Where the threshold separating the predicted classes is set at 0.5 (horizontal red line) one healthy patient (Patient Con_165a) would be mis-classified as infected. Given there are 20 examples in the test set, and 8 true positive cases, this results in an accuracy score of 0.95 or $\frac{19}{20}$, and a precision of 0.89 or $\frac{8}{9}$. Alternatively, setting the threshold at 0.55, results in all points correctly classified, hence the AUC of 1.0. Table 6.4 illustrates the three performance metrics at a range of thresholds. For example, when the threshold is set at 0.1, all examples are predicted as infected, hence accuracy and precision are both 0.40 or $\frac{8}{20}$. When the threshold is raised to 0.55, all examples are correctly classified and accuracy, recall and precision are all 1.0, explaining the observed AUC of 1.00. At a threshold of 0.7, a false negative prediction lowers accuracy and recall.

A further observation on the data in Table 6.3 is that precision is 0.88, 0.89 and 0.86 for iterations 2, 3 and 7 of the model respectively, despite there being only one mis-classified point in each case. This is due to the variation in the number of positive examples in the test set in each iteration. This is a further illustration of the importance of averaging performance measures over multiple train / test splits when using a low sample size data set.

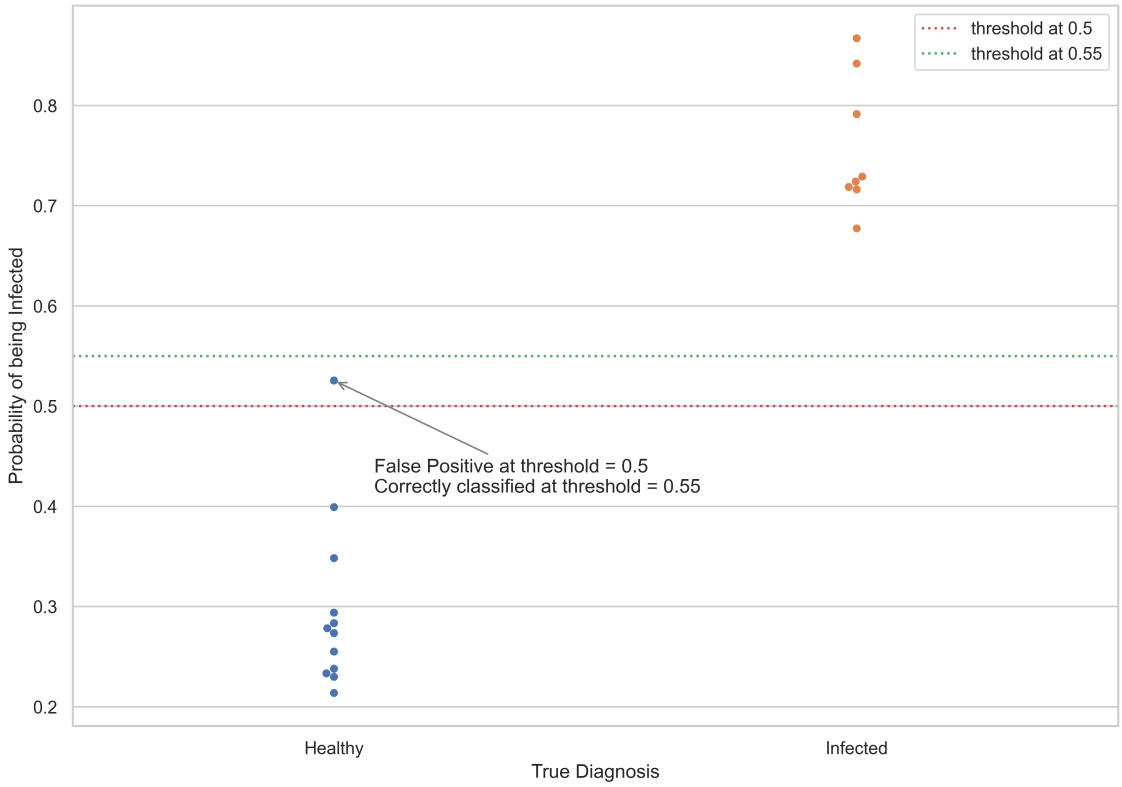

Figure 6.1: Illustration: impact of threshold on classification

Threshold	Accuracy	Recall	Precision
0.10	0.40	1.00	0.40
0.30	0.85	1.00	0.73
0.50	0.95	1.00	0.89
0.55	1.00	1.00	1.00
0.70	0.95	0.88	1.00

Table 6.4: Illustration: impact of threshold on performance metrics

To summarise, all metrics in the above tables other than AUC are stated with respect to a threshold of 0.5. The very low sample size and therefore the small test set of approximately 20 examples on each model iteration means that accuracy, recall, specificity and precision are all sensitive to the chosen threshold probability and the balance of the classes in the test set. Given this, AUC will be used as the primary metric for model evaluation for the remainder of this report.

6.1.4 Performance vs Number of Features Trade-Off

As outlined in Chapter 2, the aim of the work is to develop an accurate classifier with a small set of features. The above results demonstrate near perfect classification using L1 penalised logistic regression when 30 features are targeted. The model training and testing was therefore repeated, optimising for successively smaller numbers of features to determine if classification performance deteriorates with a smaller feature set. The average number of features selected in training and the mean AUC over nine iterations was again evaluated. The results are given in Table 6.5. The results show that there is very little performance loss by reducing the number of features from 30 to 10. However, this should be interpreted with some caution given the variance in the number of features actually selected in each training iteration of the model. Table 6.5 shows the mean number of features actually selected over nine iterations. Below a target of around 10 features, some iterations of the model failed to converge on a non-zero value of C, resulting in all features being eliminated. This illustrates the limitations of this approach to feature selection. Using the L1 regularisation parameter alone appears to be insufficient to finely control the number of features selected in model training. This points to the need for a more systematic approach to test performance with a specific number of features, such as using the recursive feature elimination algorithm discussed in Section 6.2.

Target Number of Features	Mean AUC	Mean Features Selected
30	1.00	27.0
25	0.99	22.0
20	1.00	18.0
15	1.00	15.0
10	1.00	9.0

Table 6.5: Performance vs Features Trade-Off

6.1.5 Logistic Regression: Final Model

The final logistic regression model was retrained on the full dataset, again excluding patient Inf075, optimising for 10 features. The Bayesian optimisation converged after 22 evaluations. The resulting model selected 12 features, with a hyperparameter C of 0.081.

The final feature set comprises 12 genes: ID3, NMT2, UBE2Q2, HS.276860, LATS2, PKIA,

GSN, PSTPIP2, P2RX1, SLC2A3, B4GALT5 and CD3G. Two is these genes overlap with genes identified in the 52-gene classifier in the 2014 project, namely SLC2A3 and B4GALT5.

Figure 6.2 shows the z-scored gene expression levels in the original data set for the 62 patients (excluding patient Inf075) for each of the 12 selected genes. The healthy patients are differentiated from the infected patients, and it is clear that each of the selected genes shows either strongly increased or decreased expression in the infected patients.

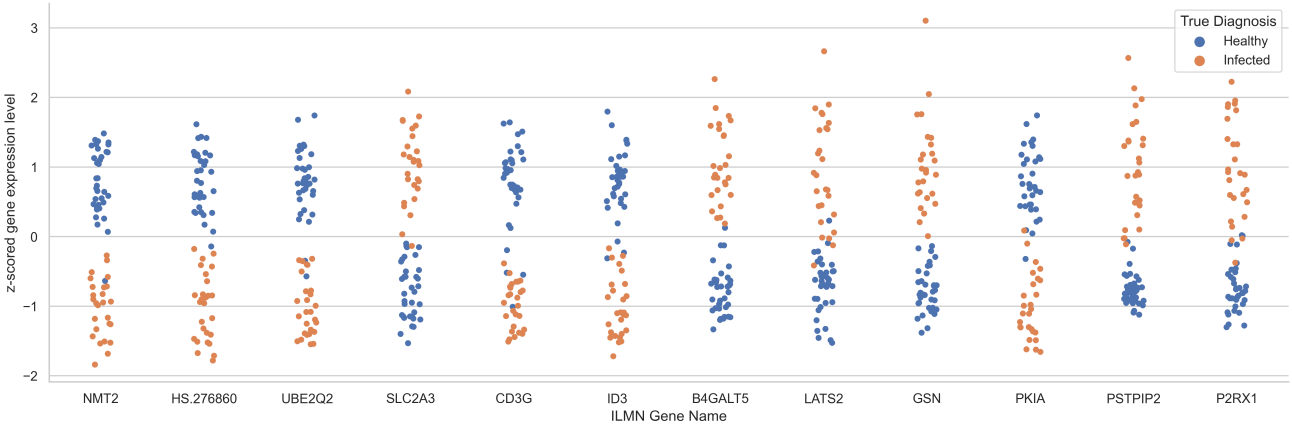

Figure 6.2: Patient gene expression levels for the 12 selected genes

6.2 Support Vector Machines

The results of the initial SVM model, using an L1 penalty to optimise for a fixed number of features were poor, with an average AUC consistently below 0.9 over nine iterations.

The second approach combined linear SVM with RFE, as outlined in Section 5.4.2. The SVM models were trained on the dataset excluding patient Inf075, given the uncertainty around the diagnosis and hence the label for this example. In order to limit computation time, the best 250 features were selected based on information gain between the training data set and training labels and the RFE procedure was used to reduce the remaining 250 features to a final set. A nested cross validation procedure was used, with the 250 feature selection performed independently for each train / test split, and the RFE procedure cross validated in the inner cross validation loop using leave one out cross validation. The full results solving for 30 features are provided in Table 6.6. The AUC is 1.00 for every iteration of the train / test set, indicating perfect classification of the data.

Iteration	C	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	89.093	1.00	1.00	1.00	1.00	1.00	-	-
2	0.005	0.95	1.00	0.94	0.83	1.00	Con_165a	-
3	0.016	0.95	0.93	1.00	1.00	1.00	-	Inf_149
4	16.321	1.00	1.00	1.00	1.00	1.00	-	-
5	0.001	1.00	1.00	1.00	1.00	1.00	-	-
6	0.003	0.95	0.89	1.00	1.00	1.00	-	Inf_149
7	0.004	1.00	1.00	1.00	1.00	1.00	-	-
8	5.251	0.95	0.91	1.00	1.00	1.00	-	Inf_149
9	0.827	0.95	1.00	0.91	0.90	1.00	Con_165a	-
Mean	n/a	0.97	0.97	0.98	0.97	1.00	n/a	n/a

Table 6.6: SVM: 30 Feature Model Performance (Excluding Patient Inf075)

6.2.1 Performance vs Number of Features Trade-Off

The experiment was repeated, optimising for each of 3, 4, 5, 6, 8, 10, 15, 20 and 25 features to understand the impact of a reduced feature set on model performance. In contrast to the case of logistic regression, the number of features selected by the final model in the RFE procedure is precisely the target number. The AUC at each number of features is detailed in Table 6.7. It is clear that the model's classification performance remains strong, even with 3, 4, and 5 features selected.

Number of Features	Average AUC
30	1.00
25	1.00
20	1.00
15	1.00
10	1.00
8	0.99
6	1.00
5	0.99
4	0.99
3	0.99

Table 6.7: SVM: Performance vs Features Trade-Off

6.2.2 Feature Stability

The strong performance of the SVM classifier, even with a very low number of features is an encouraging result. This raises the question of the stability of the features selected. As discussed in Section 4.3.4, for the chosen feature set to be meaningful, it must be relatively stable to changes in the training data. During model training, the subset of genes selected for each train/test iteration was captured and the frequency of each gene tallied across the nine iterations. The results of this are shown in Figure 6.3. The frequency count for the top 20 genes selected over all nine iterations is shown for three illustrative cases, the 30 feature model, 10 feature model and 5 feature model. The results indicate some instability in the feature sets. In the 30 feature model, the most frequently selected gene(s) are only selected in seven out of nine training rounds, compared with four out of nine in the 10 gene predictor, and 3 out of nine in the five gene predictor. A more quantitative analysis of the stability of the selected features is the subject of further work and might provide a quantitative basis for selecting the optimum number of features.

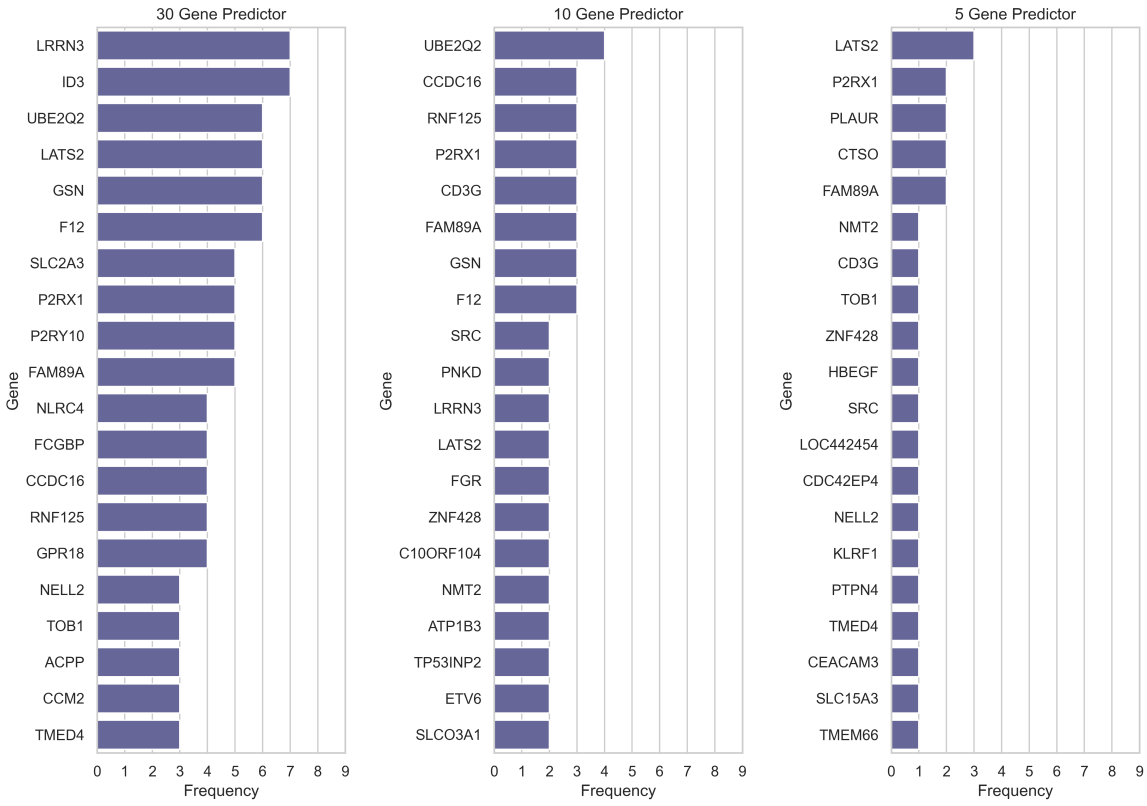

Figure 6.3: SVM: Gene frequency in feature set over nine training iterations

6.2.3 SVM: Final Model

As for logistic regression, the final model was retrained on the full data set, again excluding patient Inf075, filtering down to 250 features based on information gain and optimising the model for five features using an RFE procedure. The resulting model selected five features, with a hyperparameter C of 0.007. The selected genes were UBE2Q2, ID3, P2RX1, GSN and LATS2. Despite the instability in the selected genes in the 5 gene predictor during training rounds, these five genes selected by the final model over the whole data set show good correspondence to the most frequently occurring genes in the 30 gene predictor during training. This gives some confidence in the validity of the selected genes. There is however no overlap between these five genes and the 52-gene classifier in the 2014 project.

Figure 6.4 shows the z-scored gene expression levels in the original data set for the 62 patients excluding patient Inf075 for each of the five selected genes. The healthy patients are differentiated from the infected patients, and it is again clear that each of the selected genes shows either strongly increased or decreased expression in the infected patients.

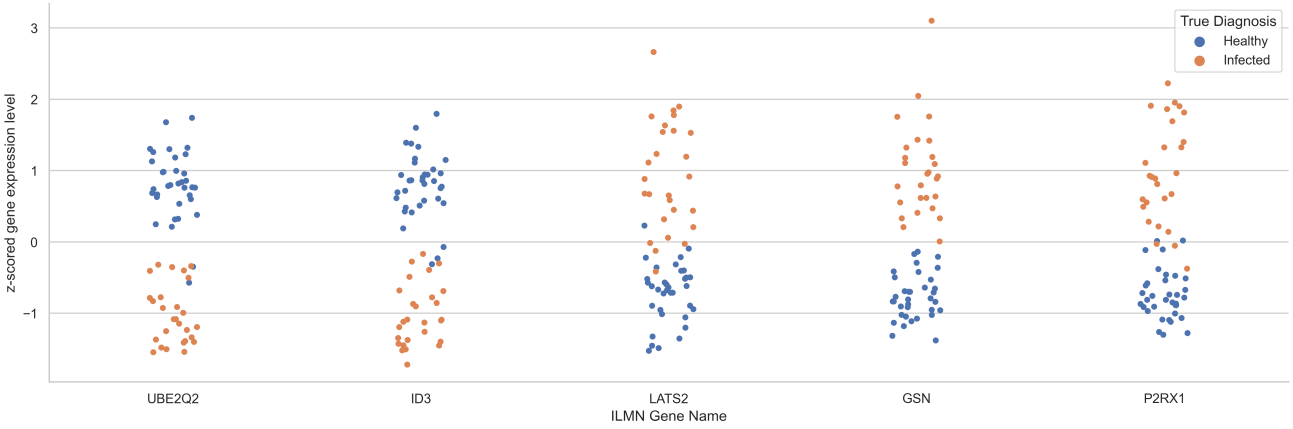

Figure 6.4: SVM: Patient gene expression levels for the five selected genes

6.3 Random Forest Results

The random forest model was trained using the data set excluding patient Inf075. The random forest model selecting 30 features achieved an average AUC of 1.0. Table 6.8 gives the performance of the model over nine iterations. Performance was not improved by increasing the number of estimators or maximum features evaluated at each node, all iterations selected 500, and 100 as the best hyper parameters respectively.

Iteration	Accuracy	Recall	Specificity	Precision	AUC	FP	FN
1	1.00	1.00	1.00	1.00	1.00	-	-
2	0.95	0.90	1.00	1.00	1.00	-	Inf_149
3	0.95	0.89	1.00	1.00	1.00	-	Inf_132a
4	0.95	0.88	1.00	1.00	1.00	-	Inf_149
5	1.00	1.00	1.00	1.00	1.00	-	-
6	1.00	1.00	1.00	1.00	1.00	-	-
7	0.95	0.88	1.00	1.00	0.99	-	Inf_149
8	1.00	1.00	1.00	1.00	1.00	-	-
9	1.00	1.00	1.00	1.00	1.00	-	-
Mean	0.98	0.95	1.00	1.00	1.00	n/a	n/a

Table 6.8: Random Forest: 30 Feature Model Performance (Excluding Patient Inf075)

6.3.1 Performance vs Number of Features Trade-Off

Table 6.9 shows that performance was equally good with the reduced number of features selected. A random forest model with just 3 features gave near perfect classification, AUC of 0.99 on average over nine iterations.

6.3.2 Feature Stability

Similarly to the SVM case, the subset of genes selected during model training for each train/ test iteration was captured and the frequency of each gene tallied across the nine iterations. The results of this are shown in Figure 6.5. The frequency count for the top 20 genes selected over all nine iterations is shown for three illustrative cases, the 30 feature model, 10 feature model and 5 feature model. The results indicate greater instability in the feature sets than in the case of SVMs. In the 30 feature model, the most frequently selected gene(s) are only selected in six out of nine training rounds, compared with three out of nine in the 10 and 5 gene predictors. There is also less overlap between the top 20 genes selected across the 30, 10 and 5 gene predictors than in the SVM results. The wider variation in the selected genes in the training iterations in the random forest model compared with the SVM model is likely a

Number of Features	Average AUC
30	1.00
25	1.00
20	1.00
15	0.99
10	1.00
8	1.00
6	1.00
5	1.00
4	1.00
3	0.99

Table 6.9: Random Forest: Performance vs Features Trade-Off

result of the pre-selection of the 250 genes in the SVM case. A more thorough investigation of the stability of the features sets and the underlying reasons should be the subject of further investigation.

6.3.3 Random Forest: Final Model

The final model was trained on the full dataset and the best five features identified. The final features selected were LILRA6, RNF125, BCL3, FLOT1 and LBH. Again, none of these genes feature in the 52-gene classifier in the 2014 project.

Figure 6.6 shows the z-scored gene expression levels in the original data set for the 62 patients excluding patient 75 for each of the five selected genes. The healthy patients are differentiated from the infected patients, and it is again clear that each of the selected genes shows either strongly increased or decreased expression in the infected patients.

6.4 Biological Significance of Selected Genes

The results of this project were discussed with Professor Peter Ghazal, Sêr Cymru Chair in Systems Medicine, Systems Immunity Research Institute at Cardiff University to gather feedback on biological meaningfulness of the selected genes and their consistency with the 2014 project.

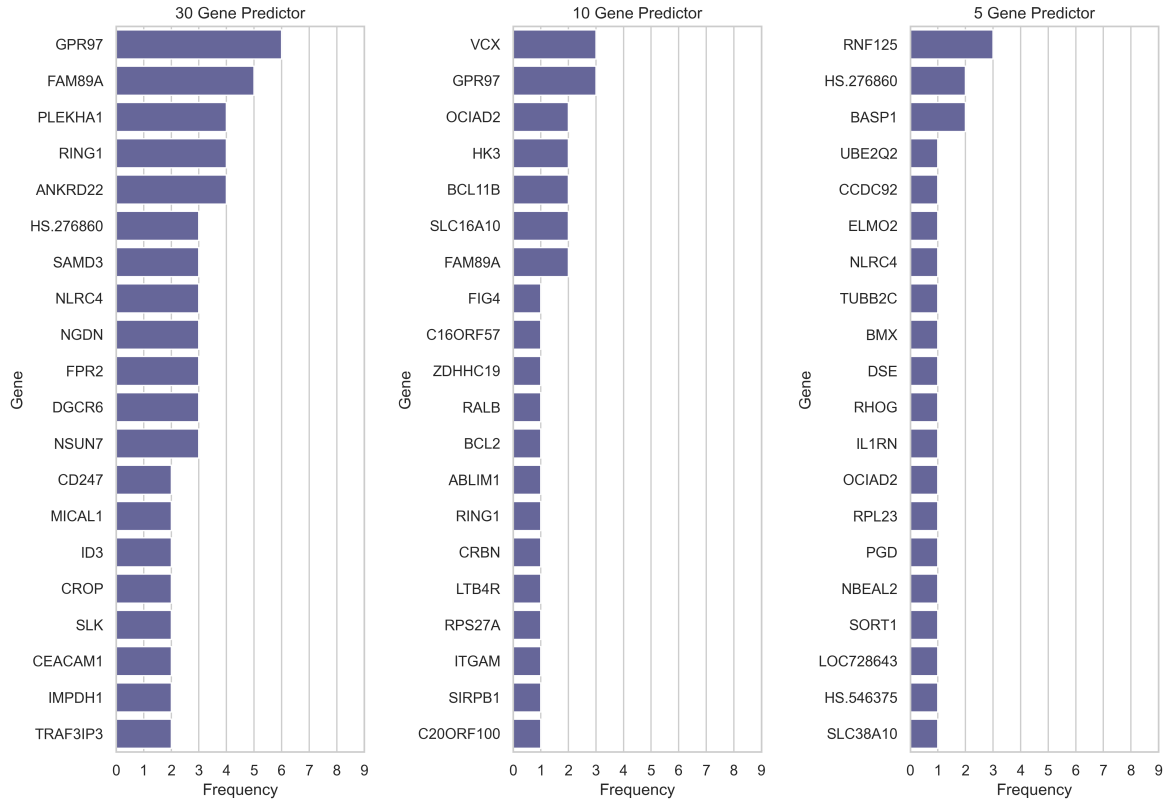

Figure 6.5: Random Forest: Gene frequency in feature set over nine training iterations

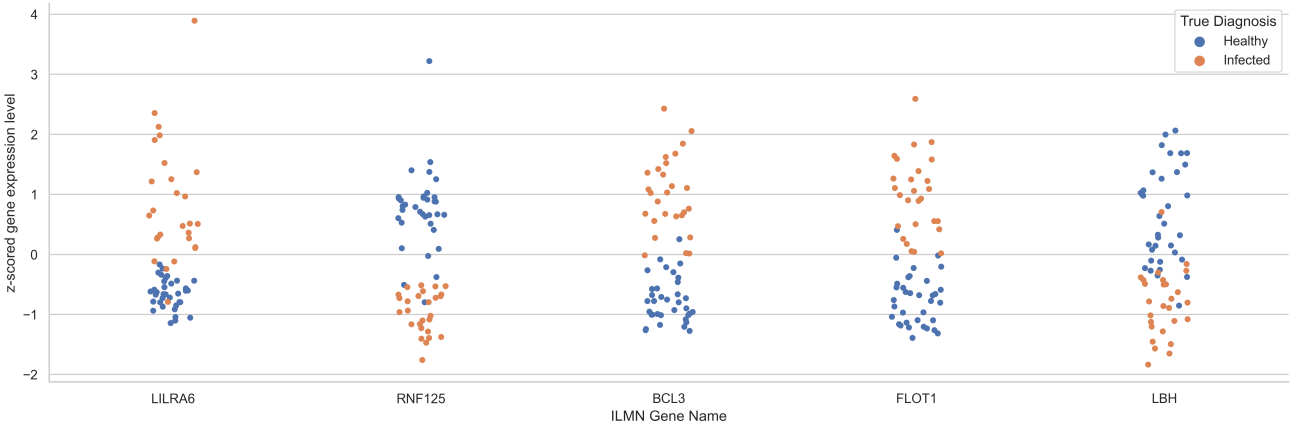

Figure 6.6: Random Forest: Patient gene expression levels for the five selected genes

The feedback received is that these results are explainable in terms of the known function of many of the genes identified and fit well with the group's existing knowledge. In particular, genes identified by the logistic regression and svm models as being under expressed in infected patients such as CD3G and ID3, are known T-Cell markers. CD3G for example plays a role in the activation of T-Cells, a key part of the body's immune response to infectious disease [31]. A number of the over expressed genes identified in the logistic regression and svm models are all

involved in the 'metabolic process to fight disease'. In general, the results are consistent with the research group's previous 2014 study and ongoing research into the sepsis disease pathway.

Chapter 7

Conclusions

7.1 Critical Evaluation of Results

The project has successfully demonstrated machine learning classifiers that accurately predict healthy from infected neonatal infant patients in the data set under investigation. In addition, it has been shown that excellent classification performance is achievable with a very low number of features, as low as three of the potential 48,202 genes are needed to separate the classes. The validity of the selected genes, in particular for the logistic regression and SVM models, has been confirmed by a world expert in the field. There are however some concerns over the stability of the set of genes selected by each model. And, given the low number of samples in the data set under investigation, it is not known how well these classifiers generalise to unseen data.

The logistic regression results demonstrate that a strong L1 penalty produces an accurate classifier with as few as ten features. Attempts to further reduce the number of features highlighted the limitations of using the magnitude of the regularisation parameter alone to control for the number of selected features as the number of features selected in the output model varied widely. The results given in Table 6.5 are therefore approximate, based on average features selected over multiple iterations. This could be improved in two ways. Firstly, the Bayesian optimisation procedure could be replaced with a simpler search procedure for adjusting the regularisation parameter to adjust the number of output features, for example a binary search process. This would help determine whether the issue lies in the Bayesian optimisation. Secondly, a wrapper method such as recursive feature elimination or sequential backward feature

selection could be performed with the logistic regression model to guarantee the number of output features, as was carried out for the SVM model. The ten features selected by the final logistic regression model contained two features that were identified in the 52-gene classier in the 2014 research, and the majority of the genes were confirmed by an expert to be significant based on their known biological function. Despite the shortcomings, the logistic regression classifier and the corresponding feature set perform the classification task extremely well.

The model combining support vector machines with recursive feature elimination gave very high classification performance on the data set under investigation, with as few as three features. The features selected on each training iteration of the model were relatively stable, and the final model selected a set of features consistent with those selected most frequently in training. There was however no overlap between the final five selected features and the 52-gene classifier from previous work, however the selected genes were confirmed to be biologically relevant. The SVM-RFE process was somewhat compromised by the need to pre-select 250 features from 48,202 before applying RFE in order to reduce the computational complexity. This was achieved using a variable ranking method with the limitations described in 4.3.1, namely features may be correlated and interactions between variables are ignored. These results are very promising, however further study is required to validate the performance of the classifier on unseen data and further investigate the stability of the selected features.

The random forest classifier achieved equally good classification performance to the SVM-RFE classifier, when measured by AUC. As with SVM, very accurate classification was possible with as few as three features. In the random forest model, the stability of the chosen feature set is an issue, perhaps indicating there are numerous three to five feature subsets that result in good performance and that the selection of any one particular feature subset is somewhat random. This could be a result of the randomised bootstrapping of features in tree training. The five selected features in the final model also showed no overlap with the 52-gene classifier. The potential inconsistency in the selected features raises questions over biological validity of the feature set, and the explainability of the result. While the model performs well in classifying this data set, it may not generalise.

Overall, this project has contributed three novel and highly performing classifiers for the diagnosis of sepsis in neonatal infants, with corresponding feature sets drawn from gene expression data. In each case the feature space has been reduced from a potential 48,202 features to just 5-10 genes, a sufficiently low number of genes to potentially make these classifiers practically

useful in a clinical setting. The validity of the selected features has been confirmed based on their known function.

7.2 Further Work

The impact of class imbalance must be investigated. The analysis has been performed with an artificially balanced data set of approximately 50-50 split of health and infected patients. In reality, the classes are highly imbalanced, with vast majority of infants being healthy. Even in the likely clinical setting for a diagnostic sepsis test for neonatal infants, the majority of the patients will presumably not have sepsis. The performance of each of the classifiers should be examined on additional imbalanced data sets that better represent the natural distribution of sepsis cases in patients.

The stability of the feature sets produced by each model have only been evaluated qualitatively in this work. At least 15 quantitative measures of the feature stability of a feature selection procedure have been reported [26]. Further work should identify the most appropriate measure for this case and apply it to the results. A better and more rigorous analysis of feature stability would be a valuable tool to determine the best model or models to take forward.

Patient Inf075 was removed from the data set having been identified as having a viral rather than a bacterial infection. The reduction of sepsis diagnosis to a binary classification problem is overly simplistic. To be clinically useful and enable clinicians to determine appropriate treatment, a diagnostic test may need to distinguish healthy patients from a number of different sepsis diagnoses, for example viral and bacterial and for a range of severity. Research is ongoing in this area. Sweeny, Wong and colleagues reported a 7-gene signature capable of distinguishing between bacterial and viral infections [32]. This project should be extended to take into account the nature and severity of the infection - it should be framed as a multi-class classification problem.

Additional machine learning algorithms should be tested, in particular neural networks. Deep neural networks have recently been demonstrated to be successful in feature selection and classification tasks with high dimensional, low sample size omics data sets [33] [34]. These were excluded from the project due to the complexity of implementation given the time frame and scope of the project, however should be investigated in further work.

Finally and perhaps most importantly, the results of the project must be validated on multiple independent patient cohorts. This validation is crucial as the low number of examples mean these models may have over fit the data. The range of patients should also be extended to include children and adults to confirm whether the sepsis biomarkers identified in neonatal infants have the same diagnostic power in older patients.

Chapter 8

Reflection

This chapter is a personal reflection on the learnings gained from the dissertation phase of the MSc Artificial Intelligence, and as such is written in the first person.

8.1 Learnings from the Research Process

Conducting a literature review as been a learning process for me. Many of the MSc coursework projects required background reading, however this was typically reading around a methodology to solve an already defined problem. In the case of this extended dissertation, the literature review served not only to educate me on the existing research in the area and the theoretical underpinnings of particular technique, but to help define the research plan. The project could have taken an number of different directions, and there is an overwhelming amount of literature on the subject of bio marker identification using machine learning, on machine learning with omics data, and on techniques for managing HDLSS data sets. The literature also cuts across the fields of machine learning, bioinformatics and medicine.

Deciding what to read and what was relevant was a new challenge. I feel I have improved my ability to filter articles based on reading abstracts, introductions and conclusions in order to determine the relevance of a particular source. I have also learned to use tools such as Google Scholar and Cardiff University Library resources to evaluate the authority of particular sources - mainly based on citation indices. Another new practical skill has been building a managing a bibliography. I experimented with various free tools before deciding on ZoteroBib to build the bibliography. As my library grows I will undoubtedly need to use a full reference manager

tool, such as Zotero or Mendelay.

A related learning is the importance of reading the literature for the purpose of designing the experiments and shaping the methodology. At the start of the project, I saw the purpose of the literature review as simply gathering background information. I started the project by going straight into implementation, building ML models using approaches I had learned in the taught modules. Some of my initial decisions on methodology led to problems, in particular around training times. As an example, I initially ran all models (logistic regression, SVMs, Random Forest) with all 48,202 features, using nested cross validation loops to score hyper parameters for accuracy. Run times were over 12 hours for some models, significantly slowing down my work. Early on I also tried to wrap logistic regression and random forests models in recursive feature elimination wrapper methods, again leading to very long run times. If I had read more of the articles I now have in my bibliography earlier on in the project, I would have understood how the computational complexity of wrapper methods scales with the number of features. I would also have learned that logistic regression with Lasso penalisation and random forests have embedded feature selection, which can be used without additional forwards or backwards search processes to select features. These computationally more efficient processes in fact produced strong results.

I feel my overall time management worked well. Based on experiences with large pieces of coursework during the taught phase of the MSc, in particular a significant project for the Foundations of Statistics and Data Science module, I allocated eight of the twelve weeks to experimentation and four weeks for write up. This was about right, and left some time additional experiments during the write up phase to enhance the work.

8.2 Personal Development

The dissertation phase has really improved my knowledge and skills in machine learning. I have had to deepen my knowledge in many areas that were only touched on in the taught course. In order to apply methods to a novel situation and to justify my decisions, I have needed to understand them at a deeper level. Good examples would be feature selection methods and logistic regression classifiers. I took the time to understand my logistic regression results by looking at the predicted probabilities and understanding the apparent discrepancies between the different performance measures like recall and AUC as outlined in Section 6.1.3. This helped

refine my understanding of how logistic regression works. As well as deepening my knowledge, I have had to investigate completely new areas of theory, in particular Bayesian optimisation and nested cross validation, both of which I described in Chapter 4.

The work has enhanced my python coding skills, in particular reducing my reliance on pre-built library functions and forcing me to code some methods myself. A good example is the nested cross validation procedure with a grid search of hyper parameters and model scoring used in the random forest model. Standard python libraries contain functions to perform a grid search with cross validation (e.g. scikit-learn's `GridSearchCV` function). However, in order to achieve what I wanted, that is to score model predictions over all folds, I needed to break these functions down and build parts of them myself. This has been a great learning in how these functions work and some of the limitations of the existing libraries.

The process of writing up has also been educational. I have enjoyed beginning to learn how to use some of the great visualisation tools such as Seaborn, and building a better understanding of LaTex and all of the flexibility and power it has.

During course of dissertation phase, I applied for PhD position to continue with this research and was awarded scholarship to pursue a PhD. I am very much looking forward to continuing with this work. There are a number of areas I have identified for development as I embark on the PhD.

Coding skills. I need to continue to build on my coding and general software engineering skills. The size of the project meant my code in a jupyter notebook became unwieldy. I need to find an approach that works for me to modularise and manage my code, and to get better at reusing code. There may be tooling that can help and this is something that I plan to research.

Mathematical understanding of the underlying algorithms. As much as I have learned in a short space of time on the MSc course, the field of machine learning is vast and understanding the underlying theory is important to guide application. I will continue to build my knowledge of the theory that underpins the algorithms and techniques that I am using.

Working with cloud infrastructure. Inevitably, future work will involve compute heavy models requiring GPUs and parallelisation. I deferred learning how to run models on the university cloud as part of this project, given the time investment in learning relative to the length of the project. This is a priority for me now as I continue with research - an investment in learning how to use more powerful tools will undoubtedly pay significant dividends in the

future.

The broader scientific context, in particular the field of bioinformatics, is also an important area of development for me. In the case of this project, the source of the data and how it was pre-processed was opaque to me. As I move forwards I need to build my understanding of the clinical and scientific context around the extraction of omics data from patient samples, potential data issues and pre-processing steps. Equally, I will need to improve my ability to interpret my own results and their significance by building some knowledge of the biomarkers that are of particular interest and significance in this field.

8.3 Final Thought

Undertaking this project has been a fantastic introduction to the field of machine learning in biomedicine, in particular biomarker identification. The project has given me a small insight into the challenges of biological data sets, in particular in relation to high dimensional low sample size data, and the need for explainability in the results.

This is not an area covered in the taught part of the MSc course, but it is however a very active area of research with applications in medical diagnosics, pharmaceutical development, and day to day clinical decision making. The ongoing improvements in omics sequencing technologies mean that there is an ever increasing amount of data available for research in this area, and I am excited about the prospect of continuing on to a PhD.

Bibliography

[1] . World Health Organization, Executive Board, "Improving the prevention, diagnosis and clinical management of sepsis: report by the secretariat," World Health Organization, Governing body documents, 2017.

[2] T. G. Buchman, S. Q. Simpson, K. L. Sciarretta, K. P. Finne, N. Sowers, M. Collier, S. Chavan, I. Oke, M. E. Pennini, A. Santhosh, M. Wax, R. Woodbury, S. Chu, T. G. Merkeley, G. L. Disbrow, R. A. Bright, T. E. MaCurdy, and J. A. Kelman, "Sepsis Among Medicare Beneficiaries: 2. The Trajectories of Sepsis, 2012–2018*," *Critical Care Medicine*, vol. 48, no. 3, pp. 289–301, Mar. 2020. [Online]. Available: http://journals.lww.com/10.1097/CCM.0000000000004226

[3] K. Honeyford, G. S. Cooke, A. Kinderlerer, E. Williamson, M. Gilchrist, A. Holmes, The Sepsis Big Room, B. Glampson, A. Mulla, and C. Costelloe, "Evaluating a digital sepsis alert in a London multisite hospital network: a natural experiment using electronic health record data," *Journal of the American Medical Informatics Association*, vol. 27, no. 2, pp. 274–283, Feb. 2020. [Online]. Available: https://academic.oup.com/jamia/article/27/2/274/5607431

[4] A. Wong, E. Otles, J. P. Donnelly, A. Krumm, J. McCullough, O. DeTroyer-Cooley, J. Pestrue, M. Phillips, J. Konye, C. Penoza, M. Ghous, and K. Singh, "External Validation of a Widely Implemented Proprietary Sepsis Prediction Model in Hospitalized Patients," *JAMA internal medicine*, vol. 181, no. 8, pp. 1065–1070, Aug. 2021.

[5] L. M. Fleuren, T. L. T. Klausch, C. L. Zwager, L. J. Schoonmade, T. Guo, L. F. Roggeveen, E. L. Swart, A. R. J. Girbes, P. Thoral, A. Ercole, M. Hoogendoorn, and P. W. G. Elbers, "Machine learning for the prediction of sepsis: a systematic review and meta-analysis of diagnostic test accuracy," *Intensive Care Medicine*, vol. 46, no. 3, pp. 383–400, Mar. 2020. [Online]. Available: http://link.springer.com/10.1007/s00134-019-05872-y

[6] C. L. Smith, P. Dickinson, T. Forster, M. Craigon, A. Ross, M. R. Khondoker, R. France, A. Ivens, D. J. Lynn, J. Orme, A. Jackson, P. Lacaze, K. L. Flanagan, B. J. Stenson, and P. Ghazal, "Identification of a human neonatal immune-metabolic network associated with bacterial infection," *Nature Communications*, vol. 5, no. 1, p. 4649, Dec. 2014. [Online]. Available: http://www.nature.com/articles/ncomms5649

[7] R. Díaz-Uriarte and S. Alvarez de Andrés, "Gene selection and classification of microarray data using random forest," *BMC Bioinformatics*, vol. 7, no. 1, p. 3, Jan. 2006. [Online]. Available: https://doi.org/10.1186/1471-2105-7-3

[8] R. C. Bone, R. A. Balk, F. B. Cerra, R. P. Dellinger, A. M. Fein, W. A. Knaus, R. M. Schein, and W. J. Sibbald, "Definitions for Sepsis and Organ Failure and Guidelines for the Use of Innovative Therapies in Sepsis," *Chest*, vol. 101, no. 6, pp. 1644–1655, Jun. 1992. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S001236921638415X

[9] S. Lambden, P. F. Laterre, M. M. Levy, and B. Francois, "The SOFA score—development, utility and challenges of accurate assessment in clinical trials," *Critical Care*, vol. 23, no. 1, p. 374, Nov. 2019. [Online]. Available: https://doi.org/10.1186/s13054-019-2663-7

[10] M. M. Islam, T. Nasrin, B. A. Walther, C.-C. Wu, H.-C. Yang, and Y.-C. Li, "Prediction of sepsis patients using machine learning approach: A meta-analysis," *Computer Methods and Programs in Biomedicine*, vol. 170, pp. 1–9, Mar. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016926071831602X

[11] M. P. Sendak, W. Ratliff, D. Sarro, E. Alderton, J. Futoma, M. Gao, M. Nichols, M. Revoir, F. Yashar, C. Miller, K. Kester, S. Sandhu, K. Corey, N. Brajer, C. Tan, A. Lin, T. Brown, S. Engelbosch, K. Anstrom, M. C. Elish, K. Heller, R. Donohoe, J. Theiling, E. Poon, S. Balu, A. Bedoya, and C. O'Brien, "Real-World Integration of a Sepsis Deep Learning Technology Into Routine Clinical Care: Implementation Study," *JMIR Medical Informatics*, vol. 8, no. 7, p. e15182, Jul. 2020. [Online]. Available: https://medinform.jmir.org/2020/7/e15182

[12] S. M. Lauritsen, M. E. Kalør, E. L. Kongsgaard, K. M. Lauritsen, M. J. Jørgensen, J. Lange, and B. Thiesson, "Early detection of sepsis utilizing deep learning on electronic health record event sequences," *Artificial Intelligence in Medicine*, vol. 104, p. 101820, Apr. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0933365719303173

[13] G. de Anda-Jáuregui and E. Hernández-Lemus, "Computational Oncology in the Multi-Omics Era: State of the Art," *Frontiers in Oncology*, vol. 10, p. 423, Apr. 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fonc.2020.00423/full

[14] M. Zitnik, F. Nguyen, B. Wang, J. Leskovec, A. Goldenberg, and M. M. Hoffman, "Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities," *Information Fusion*, vol. 50, pp. 71–91, Oct. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1566253518304482

[15] T. E. Sweeney and P. Khatri, "Benchmarking Sepsis Gene Expression Diagnostics Using Public Data*:," *Critical Care Medicine*, vol. 45, no. 1, pp. 1–10, Jan. 2017. [Online]. Available: http://journals.lww.com/00003246-201701000-00001

[16] S. Thair, C. Mewes, J. Hinz, I. Bergmann, B. Büttner, S. Sehmisch, K. Meissner, M. Quintel, T. E. Sweeney, P. Khatri, and A. Mansur, "Gene Expression–Based Diagnosis of Infections in Critically Ill Patients—Prospective Validation of the SepsisMetaScore in a Longitudinal Severe Trauma Cohort," *Critical Care Medicine*, vol. Publish Ahead of Print, Apr. 2021. [Online]. Available: https://journals.lww.com/10.1097/CCM.0000000000005027

[17] J. J. Chen, C.-A. Tsai, S. Tzeng, and C.-H. Chen, "Gene selection with multiple ordering criteria," *BMC Bioinformatics*, vol. 8, no. 1, p. 74, Mar. 2007. [Online]. Available: https://doi.org/10.1186/1471-2105-8-74

[18] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines," *Machine Learning*, vol. 46, no. 1, pp. 389–422, Jan. 2002. [Online]. Available: https://doi.org/10.1023/A:1012487302797

[19] N. Altman and M. Krzywinski, "The curse(s) of dimensionality," *Nature Methods*, vol. 15, no. 6, pp. 399–400, Jun. 2018. [Online]. Available: http://www.nature.com/articles/s41592-018-0019-x

[20] A. Bommert, X. Sun, B. Bischl, J. Rahnenführer, and M. Lang, "Benchmark for filter methods for feature selection in high-dimensional classification data," *Computational Statistics & Data Analysis*, vol. 143, p. 106839, Mar. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016794731930194X

[21] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[22] B. C. Ross, "Mutual Information between Discrete and Continuous Data Sets," *PLoS ONE*, vol. 9, no. 2, p. e87357, Feb. 2014. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0087357

[23] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, pp. 267–288, 1996. [Online]. Available: https://www.jstor.org/stable/2346178

[24] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://link.springer.com/10.1023/A:1010933404324

[25] "Ensemble methods." [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html#l2014

[26] S. Nogueira, K. Sechidis, and G. Brown, "On the Stability of Feature Selection Algorithms," *Journal of Machine Learning Research*, vol. 18, no. 174, pp. 1–54, 2018. [Online]. Available: http://jmlr.org/papers/v18/17-514.html

[27] A. E. Teschendorff, "Avoiding common pitfalls in machine learning omic data science," *Nature Materials*, vol. 18, no. 5, pp. 422–427, May 2019. [Online]. Available: http://www.nature.com/articles/s41563-018-0241-z

[28] Y. Zhong, J. He, and P. Chalise, "Nested and Repeated Cross Validation for Classification Model With High-Dimensional Data," *Revista Colombiana de Estadística*, vol. 43, no. 1, pp. 103–125, Jan. 2020. [Online]. Available: https://revistas.unal.edu.co/index.php/estad/article/view/80000

[29] M. K. Gilles Louppe. (2016) Bayesian optimization with skopt. [Online]. Available: https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html#sphx-glr-auto-examples-bayesian-optimization-py.htm

[30] P. I. Frazier, "A Tutorial on Bayesian Optimization," *arXiv:1807.02811 [cs, math, stat]*, Jul. 2018, arXiv: 1807.02811. [Online]. Available: http://arxiv.org/abs/1807.02811

[31] "Gene cards, the human gene database." [Online]. Available: https://www.genecards.org/cgi-bin/carddisp.pl?gene=CD3G

[32] T. E. Sweeney, H. R. Wong, and P. Khatri, "Robust classification of bacterial and viral infections via integrated host gene expression diagnostics," *Science Translational*

Medicine, vol. 8, no. 346, pp. 346ra91–346ra91, Jul. 2016. [Online]. Available: https://stm.sciencemag.org/lookup/doi/10.1126/scitranslmed.aaf7165

[33] B. Liu, Y. Wei, Y. Zhang, and Q. Yang, "Deep Neural Networks for High Dimension, Low Sample Size Data," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 2287–2293. [Online]. Available: https://www.ijcai.org/proceedings/2017/318

[34] D. Singh, H. Climente-González, M. Petrovich, E. Kawakami, and M. Yamada, "FsNet: Feature Selection Network on High-dimensional Biological Data," *arXiv:2001.08322 [cs, stat]*, Dec. 2020, arXiv: 2001.08322. [Online]. Available: http://arxiv.org/abs/2001.08322

Appendix A

Supporting Python Code

This thesis is accompanied by the following files containing the input data and python code used to produce the results in Chapter 6.

File Name	Contents
`genomic_data.csv`	Input data set
`sepsis_preprocessing.py`	Data preprocessing steps to deduplicate and standardise input data
`sepsis_evaluation.py`	Shared functions for model scoring and gene identification
`sepsis_lr.py`	Logistic regression analysis
`sepsis_svm.py`	Support vector machine analysis
`sepsis_rf.py`	Random forest analysis

Table A.1: Source Code Files

All input data, python code files and output illustrations, as well as the PDF version of this thesis are available in this github repository: `https://github.com/parkyed/sepsis_ml_omics_msc`

Appendix B

Code Libraries and Versions

The following python code libraries were used during the course of this work.

Library	Version
scikit-learn	0.24.1
scikit-optimize	0.8.1
scipy	1.6.1
pandas	1.2.3
numpy	1.20.1
matplotlib	3.3.4
seaborn	0.11.1

Table B.1: Code Libraries and Versions