Machine Learning for disease diagnosis using omics data

School of Computer Science and Informatics, Cardiff University

MSc Artificial Intelligence

Edward Parkinson

10 October 2021

Abstract

Abstract goes here

Acknowledgements

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

Contents

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 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 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

Aims and Objectives

The aim of this project is to develop an 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 (diasnosis). 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. Here redundant genes need to be eliminated.

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 classification model.

In order to achieve this aim, this project will take as input a dataset 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

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, MEWS, and SOFA [8] [6]. A 2020 meta-analysis by K. 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 [9] [10]. The overwhelming majority of existing studies appear to use a small (fewer than 50) number of features, typically clinical vital signs, co-morbidities, 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, metabolomics is a very active area of research in disease diagnosis and understanding, particularly in areas such as oncology [11]. 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 [12]. In the diagnosis of sepsis specifically, gene-expression based diagnostic approaches have recently been developed and validated in clinical settings [13] [14] in collaboration with California-based Inflamatix Inc. who are developing a commercial diagnostic device. 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 multi-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 dataset used was recruited from neonatal infants at the Neonatal Unit, Royal Infirmary of Edinburgh and analysed by Claire L. Smith, Peter Ghazal and colleagues at the University of Edinburgh 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 control. 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 mean expression level between the two classes [15].

The 2014 project selected the 52 candidate genes from a possible 48,802 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. This approach to pre-selecting genes is not necessarily the most appropriate for selecting 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 selected features may be correlated and interactions between genes is not taken into account [7]. The limitations of univariate feature selection approaches are explained more fully in 4.3.1. 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].

Chapter 4

Problem

4.1 Overview

This challenge in this project is to develop a highly accurate ML classifier capable of differentiating neonatal infants with sepsis ('infected') from healthy neonatal infants ('not-infected'). 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 [16].

4.2 High Dimensional, Low Sample Size Data

The dataset used for the project contains data on 63 neonatal infants, with information on expression levels for 48,802 genes 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 so called '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 present unique challenges, often referred to as the 'Curse of dimensionality' [17]. In particular:

1. **Data sparsity** - the data in 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 population.

2. **Multicollinearity** - 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 (63) 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 the usual ML pipeline tasks such a feature selection, hyper parameter tuning and model evaluation, this work will need to employ techniques to maximise 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 there 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 dataset, the features selected can be verified for their relevance based on prior expert knowledge of their function and equally they may help elucidate the underlying biological processes taking place [18] [16]. Feature selection methods have been characterised into variable ranking methods, wrapper methods, and embedded methods [19]. The following sections briefly outline the principles of each types, and details the specific methods used in this project.

4.3.1 Variable Ranking Methods

Variable ranking feature selection methods are performed as a preprocessing 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 x features, or features with a score above a threshold are then selected. The process is computationally efficient as for a set of n variables, n scores are computed [19]. Section 3.3 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 which 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 and an overly complex classifier. At the same time, useful features will have been discarded.

2. Variable interactions are ignored. A variable that show 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, complimentary genes could be missed from the feature set [19].

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. Mutual information is chosen as it may be used

with discrete and continuous data as in our classification problem, will detect any relationships between the datasets and is insensitive to the number of examples [20]. For each variable in the dataset, the mutual information between variable vector x_i and target vector y is calculated, and the variables ranked by their score. Mutual information is a positive number, the higher the value, the stronger the relationship and it is equal to zero if and only if two random variables are independent.

Formally, the mutual information between variables X and Y is stated as follows, where $H(X)$ is the entropy for X and $H(X|Y)$ is the conditional entropy for X given Y.

$$I(X;Y) = H(Y) - H(Y|X)$$

For a given variable X_i, in the case of binary classification, mutual information is given by:

$$I(Y;X_i) = H(Y) - H(Y|X_i) = C(P[y]) - (P[x_1]C(P[y|x_1]) + P[x_0]C(P[y|x_0])$$

Where entropy is given by C, a function of probability a of the label being equal to 1:

$$C(a) = -alog_2(a) - (1-a)log_2(1-a)$$

The function used here to calculate mutual information is the scikit-learn function mutual_info_classif. This function uses an approach based on k-nearest neighbour distances to discretize the continuous variables for the purpose of calculating mutual information as developed by Ross 2014 [20].

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 [16].

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 a 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:

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

The RFE proceedure is as follows:[16]

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 ith the smallest ranking criterion

The proceedure produces a feature ranking, where feature subsets are nested $F_1 \subset F_2 \subset F_3$.

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 importances in decision tree based models.

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

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

Where the cost contribution from a given example is:

$$Cost(h_\theta(x), y) = -ylog(h_\theta(x)) - (1-y)log(h_\theta(x))$$

add a dot · between y and log

And:

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

Regularisation adds a penalty term to the cost function $J(\theta)$ to constrain the magnitude of the weights. L1 or Lasso 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:

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

you can remove this.

add numbers to all the equations and formulas

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. the features are removed.

decide on using lower or upper case throughout the dissertation for Decision Trees, Neural Networks, ...

Decision Tree based models also have embedded feature selection methods. As an example, in a random forest model classifier, with k features, the importance of each feature can be calculated by averaging the information gain from this feature across all trees where the feature is used.

Given the very large number of features in our dataset, our 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 high classification performance with the chosen models.

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, whether determined by the value of a regularisation parameter, or at each iteration of a recursive feature elimination process.

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 [21].

To solve this, 'nested cross validation' is often applied to small datasets. 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, with 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 (63 examples), 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 mutliple train / test splits (i.e. multiple outer CV loop). The general nested cross validation scheme is illustrated in Figure 4.1.

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.

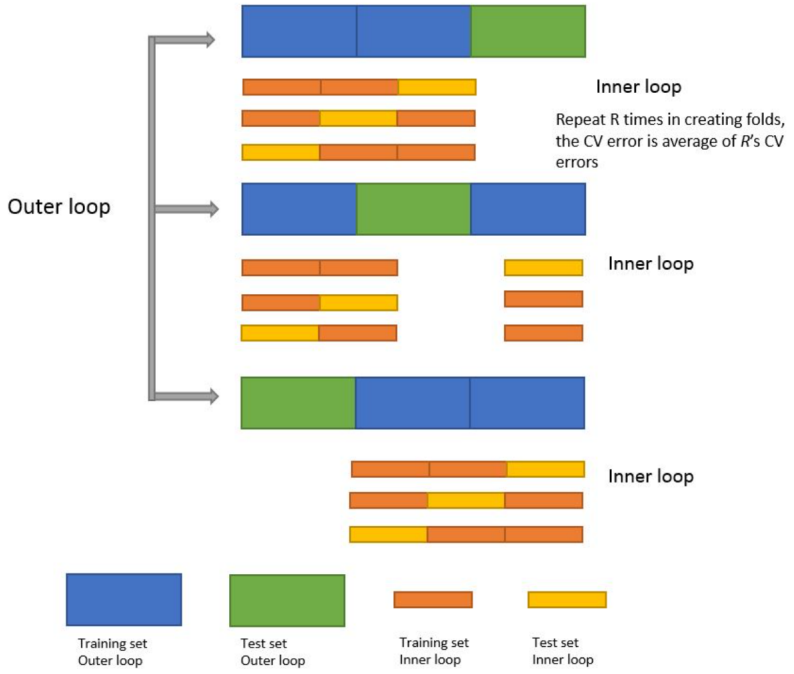

Figure 4.1: Illustration of Nested Cross Validation [22]

Bayesian optimisation using gaussian processes is an alternative approach to searching the hyperparameter 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.

Bayesian optimisation is a machine-learning optimisation methodology that solves the problem:

$$x^* = \arg\max_x f(x)$$

Where the set A of possible values of x, and the objective function f have the following properties: [23] [24] :

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; such that it can 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 or AUC of a machine learning model, and where there is a large potential search space, making $f(x)$ expensive to evaluate, fits these 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 that value that would be given by evaluating the objective function at a given new value of x. The acquisition function 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 [23]:

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. hyper parameter space) by progressively evaluating 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 hyper parameter 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 [24].

Chapter 5

Approach

The project will apply a number of widely used machine algorithms to the neonatal dataset, 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, random forests, xgboost ~~and deep neural networks.~~

5.1 Data Collection

Prior to the start of this project, the RNA collected from the infected and control infants was hybridized onto Illumina Human Whole-Genome Expression BeadChip HT12v3 microarrays comprising 48,802 features (human 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 [6].

5.2 Data Pre-Processing

Data pre-processing consisted of three steps. Firstly, the dataset was de-duplicated to ensure each gene is represented by a single feature in the dataset. The data was de-duplicated based on the column Probe_Id, on the basis of advice from the project sepsis team. 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, meaning all 63 values for each feature were rescaled to give a mean of 0 and standard deviation of 1 ('z-scored'). 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.

I'd delete this, as it is a rather weak explanation.

5.3 Validation Approach

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, ~~which was determined to be a reasonable balance between training and test data~~. 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 the multiple splits 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 proceedure) 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 nested cross validation procedure was manually coded to allow the predictions for the validation examples to be stored and then scored across all folds, as standard python cross validation functions typically do not facilitate this, as they calculate metrics on each fold on average across folds, giving errors when using a leave one out strategy. The code is illustrated in Listing 5.1

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)

    # lists to collect scores from each outer fold
    scores_list = []
    best_parameters_list = []
    mis_classified_list = []

    # create inner loop for hyper parameter optimisation
    for train_ix, test_ix in cv_outer.split(df):

        # split data
        X_train, X_test = df.iloc[train_ix, :], df.iloc[test_ix, :]
        y_train, y_test = labels[train_ix], labels[test_ix]

        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)

        # evaluate the model using a custom function
        scores = score_model(y_pred, y_pred_probab, y_test)
        mis_classified = mis_class_points(X_test, y_pred, y_test)

        # add scores for this train test split to list of all iterations
        scores_list.append(scores)
        best_parameters_list.append(best_parameter)
        mis_classified_list.append(mis_classified)

    return scores_list, best_parameters_list, mis_classified_list
```

5.4 Machine Learning Model Choices

The choice of the class of machine learning classifiers 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 [16].

Random Forest and XGBoost were selected again for their well documented performance in classification tasks, their speed as well as the interpretability of the embedded feature selection.

Deep Neural Networks have recently been demonstrated to be successful in feature selection and classification tasks with high dimensional, low sample size omics datasets [25] [26]. These were excluded from the project due to the complexity of implementation given the timeframe and scope of the project.

5.4.1 Logistic Regression

Logistic regression was implemented using scikit-learn's built in Logistic Regression function, with an L1 penalty, using 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 code for the objective function is given in Listing 5.2 to illustrate.

Listing 5.2: Bayesian Optimisation - Optimising Classification Accuracy

```
# 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):

    # set cross validation strategy and split train and validation sets
    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
        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
result = gp_minimize(evaluate_model, search_space_lr, acq_func="EI", x0=
n_initial_points=20, n_calls=30)
```

The number of iterations of the Bayesian optimisation was determined empirically to ensure convergence result, i.e. a maximum accuracy for the model. In this formulation, the number of features selected is an output of the process of maximising model performance.

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 overfitting. 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 also significantly reduced training times.

Listing 5.3: Bayesian Optimisation - Optimising Fixed Number of Features

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

# set the target number of feature
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=
n_initial_points=20, n_calls=30)
```

5.4.2 Support Vector Machines

A linear support vector machine classifier was initially with L1 regularlisation, to select a fixed number of features, as a baseline for performance. In this case cross validation was not required in the inner loop.

As a second step, a linear SVM was combined with RFE to reduce the set of features used in the final model. The RFE proceedure 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 over 48,200 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 ~~(ability to train the model overnight)~~, two compromises were introduced: (i) Features were discarded in groups of 20, reducing the number of fit models by 20 times and (ii) a variable ranking feature selection method was introduced prior to performing RFE, based on mutual information gain. The variable ranking feature selection and RFE were included in the same pipeline, along with hyper parameter tuning, and performed within the inner cross validation loop, to ensure no data leakage from the initial feature selection step to the test set.

Similarly to the case of logistic regression, hyper parameter 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 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 to optimise
    @use_named_args(search_space_svm)
    def evaluate_model(**params):
        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
        y_train_inner, y_val = y_train[train_index], y_train[val_index]

        # define the pipeline
        svc = SVC(kernel="linear", probability=True)
        svc.set_params(**params)
        rfe = RFE(estimator=svc, n_features_to_select=out_features, step=
        svm_pipeline = Pipeline(steps=[('rfe', rfe),
                                       ('model', svc)])

        svm_pipeline.fit(X_train_inner, y_train_inner)

        y_val_pred = svm_pipeline.predict(X_val)
        y_val_classes.append(y_val)
        y_val_predictions.append(y_val_pred)

    score = accuracy_score(y_val_classes, y_val_predictions)
    return 1.0 - score

# perform gp optimization, and store best hyper parameters
result = gp_minimize(evaluate_model, search_space_svm, n_calls=20)
plot_convergence(result)
plt.savefig("svm_best_features_nf.jpeg")
best_C = result.x[0]

print('Best score: %.3f' % (1.0 - result.fun))
print('Best C: %s' % (best_C))

# retrain the model with selected hyper parameters
clf = SVC(kernel="linear", C=best_C, probability=True)
```

```
    feature_elim = RFE(estimator=clf, n_features_to_select=out_features, step
    model = Pipeline(steps=[('feature_elim', feature_elim),
                            ('clf', clf)])

    model.fit(X_train, y_train)

    return model, best_C
```

As with Logistic Regression, following hyper parameter optimisation, the full train set was retrained using the hyper parameters selected and this final model 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.

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 hyper parameter optimisation [7], therefore simpler grid search used over limited hyper parameter space to validate sensitivity to key hyper parameters, rather than the Bayesian optimisation used for logistic regression and SVM.

The training time complexity of a Random Forest is given by:

$$O(k * d * n * log(n))$$

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 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 - 200, 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 '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 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 dataset (train + test) and the selected features identified.

5.5 Model Evaluation

The problem of diagnosing infected from health 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 5.1 .

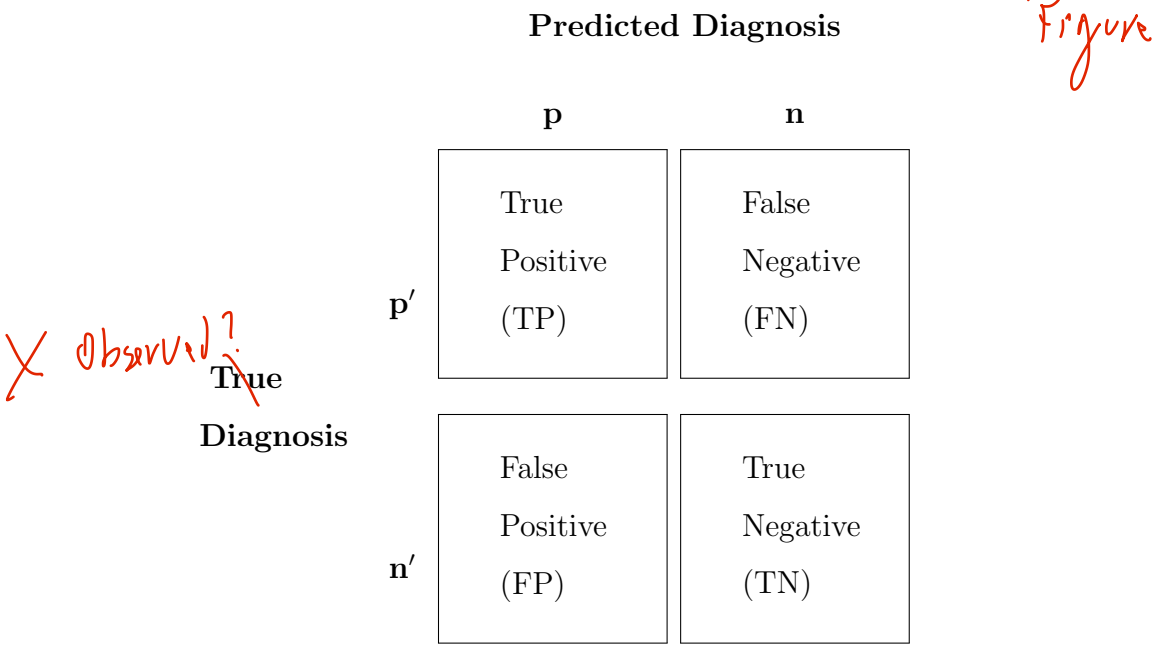

Figure 5.1: Confusion Matrix

Accuracy measures the % of correct predictions in the test set, regardless of whether positive (infected) or negative (healthy). Where the classes are inbalanced, accuracy may give an

unrealisticly positive view of performance, for example 95% accuracy sounds high, however if one class represents 5% of examples, then all of this class could be mis-classified - i.e. the predictor has no power. Our neonatal dataset is artificially imbalanced since the overwhelming majority of real world neonatal infants will be healthy. Accuracy is used for hyper paramter tuning in our case, given the dataset 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 % of all positive cases correctly diagnosed. 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 '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 % of all negative cases correctly diagnosed. 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 % of all of the cases diagnosed as positive that are actually positive. Precision provides a different measure of the impact of false positives in the predictions.

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

Receiver Operating Characteristic ("ROC") is widely used in medical diagnostic binary classification settings. The ROC 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 a probability of belonging to the positive class for each example in the text set. 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 curve can therefore be used to assess the trade off between the sensitivity of a classifier and the false positive rate.

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 not ability to separate the classes - no better than a random guess.

The ROC curve can be used to determine the trade-off between false positive and false negatives in a given model. In the sepsis diagnosis setting, the cost of false positive and false negatives may not be the same - the trade of is the risk of death from sepsis, weighed against the risk of complications from the unnecessary use of anti-biotics. 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.

The most important measures 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. All metrics will be reported and in some cases, where classifiers only predict the class of an example and do not produce a probability of class membership, it isn't possible to report AUC as the true positive and false positive rates both take a single value.

Chapter 6

Results and Analysis

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, and the means scores over all iterations. In order to identify the selected features, the full data set (train +test) was used to train the model and the genes identified.

On first inspection the results appear promising - the AUC ranges from 0.94 - 1.00 across the nine iterations. However, the selected L1 regularisation parameter varies widely between iterations (ranging from 0.34 to 515) and the final trained model produced non-zero coefficients for 89 different features. This large number of features relative to the number of examples in the data set, the high variance of the hyper parameter C over multiple iterations and the near perfect classification of the test sets indicate that this model may be overfitting the data - the curse of dimensionality strikes!

Iteration	C	Accuracy	Recall	Specificity	Precision	AUC
1	1.0	0.95	1.00	0.93	0.88	1.00
2	164.43	0.86	0.89	0.83	0.80	0.94
3	26.63	0.95	0.92	1.00	1.00	0.97
4	1.0	0.95	1.00	0.89	0.92	1.00
5	31.96	0.90	1.00	0.86	0.78	0.98
6	515.52	0.95	0.89	1.00	1.00	0.99
7	1.0	0.90	1.00	0.78	0.86	1.00
8	0.34	0.90	0.88	0.92	0.88	0.99
9	5.98	0.95	0.88	1.00	1.00	1.00
Mean	n/a	0.92	0.94	0.91	0.90	0.99

Table 6.1: Model Performance: Optimising Classification Accuracy

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 the specific examples misclassified in the test set at each iteration. This will be discussed in more detail in Section 6.1.2 below.

These results show a regularisation hyper-parameter C of between 0.1 and 0.48 across all iterations, significantly more stable than previously. The performance of the model is good, with an mean recall of 0.96, specificity of 0.94 and AUC of 0.99. These results, combined with the fact that each model is restricted to 30 features, give much more confidence that the model is indeed a well performing predictor that is likely to generalise to unseen data, rather than simply over fitting the training data.

Iteration	C	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	0.32	0.86	0.89	0.83	0.80	0.98	165a, 206	Inf075
2	0.17	1.00	1.00	1.00	1.00	1.00	-	-
3	0.26	1.00	1.00	1.00	1.00	1.00	-	-
4	0.2	0.95	0.90	1.00	1.00	0.99	-	Inf075
5	0.48	0.95	1.00	0.91	0.91	0.99	206	-
6	0.22	0.90	1.00	0.85	0.80	1.00	165a, 181a	-
7	0.1	0.90	0.88	0.92	0.88	0.97	206	Inf075
8	0.31	0.95	1.00	0.91	0.91	1.00	165a	-
9	0.37	1.00	1.00	1.00	1.00	1.00	-	-
Mean	n/a	0.95	0.96	0.94	0.92	0.99	n/a	n/a

Table 6.2: Model Performance: Optimising for 30 Features

6.1.2 Patient 75

As noted in Chapter 3, patient 75 was identified clinically as having a viral rather than bacterial infection. Table 6.2 shows that patient 75 was incorrectly classified as healthy in three iterations of the model, and was the only example classified as a false negative in any iteration. Given this result and the uncertainty around the label of this example, the analysis was repeated excluding patient 75, 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.

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. This assumes a specific threshold probability separating the classes. As discussed in 5.5, these metrics are typically calculated with the default threshold of 0.5. On the other hand, the ~~metric~~ AUC does not assume a threshold, since the ROC curve evaluates model performance at multiple thresholds.

As an illustration of the impact of this, the results in 6.3 show iteration 3 of the model has an AUC of 1.00, implying perfect classification, but an accuracy of 0.95, recall of 1.0 and a

Iteration	C	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	0.17	1.00	1.00	1.00	1.00	1.00	-	-
2	0.21	1.00	1.00	1.00	1.00	1.00	-	-
3	0.17	0.95	1.00	0.91	0.90	1.00	165a	-
4	0.22	1.00	1.00	1.00	1.00	1.00	-	-
5	0.16	1.00	1.00	1.00	1.00	1.00	-	-
6	0.22	1.00	1.00	1.00	1.00	1.00	-	-
7	0.13	1.00	1.00	1.00	1.00	1.00	-	-
8	0.16	0.95	1.00	0.92	0.90	1.00	165a	-
9	0.15	1.00	1.00	1.00	1.00	1.00	-	-
Mean	n/a	0.99	1.00	0.98	0.98	1.00	n/a	n/a

Table 6.3: Model Performance: Optimising for 30 Features (Excluding Patient 75)

precision of 0.90, 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 165a) would be mis-classified as infected. Given there are 20 examples in the test set, this results in an accuracy score of 0.95, and a precision of 0.90. Alternatively, setting the threshold at 0.57, 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 0.45 or $\frac{9}{20}$. When the threshold is raised to 0.57, 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 0.6, a false negative prediction lowers accuracy and recall.

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 small test set of approximately 20 examples on each model iteration means that metrics such as recall, specificity and precision are very sensitive to the chosen threshold probability. Given this, AUC will be used as the primary metric for model evaluation for the remainder of this report.

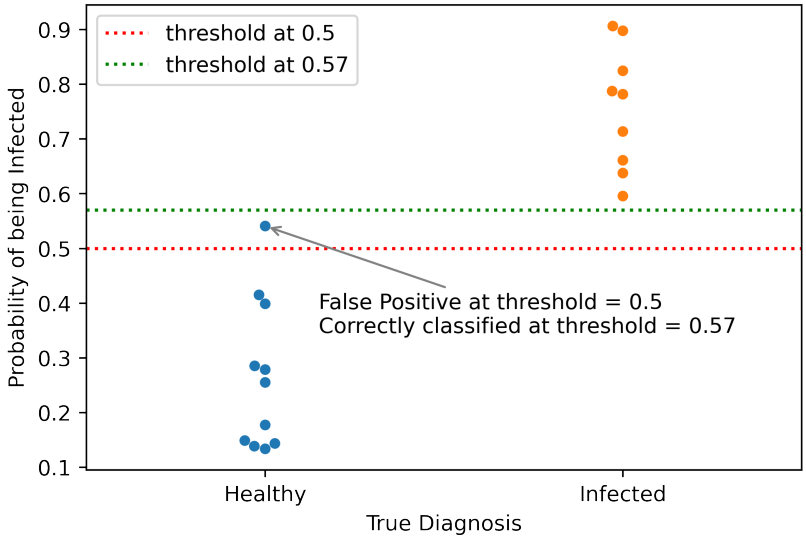

Figure 6.1: Illustration: impact of threshold on classification

Threshold	Accuracy	Recall	Precision
0.10	0.45	1.00	0.45
0.30	0.85	1.00	0.75
0.50	0.95	1.00	0.90
0.57	1.00	1.00	1.00
0.60	0.95	0.89	1.00

Table 6.4: Illustration: impact of threshold on performance metrics

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 selected. The model training and testing was therefore repeated, optimising for successively smaller numbers of features - 20, 15, 10 and 5 features - to determine how classification performance deteriorates with a smaller feature set. The mean AUC over nine iterations was again evaluated. The results are given in Table 6.5 and clearly show that there is very little performance loss by reducing the number of features from 30 to 10. Performance however did deteriorate significantly between 10 and 5 features, the AUC falling from 1.00 to 0.89.

Number of Features	Mean AUC
30	1.00
20	0.99
15	1.00
10	1.00
5	0.89

Table 6.5: Performance vs Features Trade-Off

6.1.5 Final Model

The final logistic regression model was retrained on the full dataset, again excluding patient 75, optimising for 10 features. The resulting model selected 12 features, with a hyperparameter C of 0.081. The convergence plot in Figure 6.2 shows the convergence of surrogate function $f(x)$ in the Bayesian Optimisation procedure. The optimisation converged after 22 evaluations.

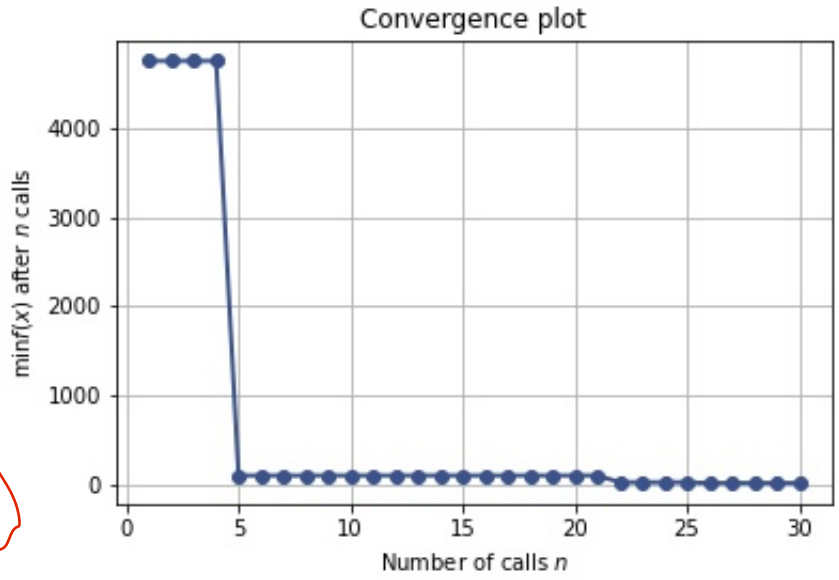

Figure 6.2: Logistic Regression: Bayesian Optimisation Convergence

The final feature set comprises 12 genes: ID3, NMT2, UBE2Q2, HS.276860, LATS2, PKIA, GSN, PSTPIP2, P2RX1, SLC2A3, B4GALT5 and CD3G.

Figure 6.3 shows the z-scored gene expression levels in the original data set for the 62 patients excluding patient 75 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. A discussion of the significance of

the selected genes is provided later in Section 6.4.

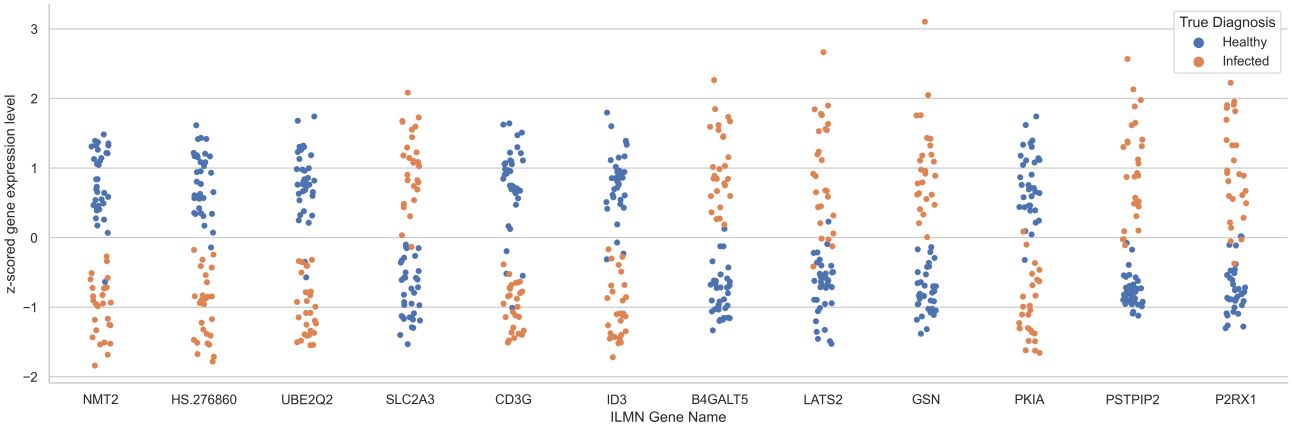

Figure 6.3: Patient gene expression levels for the 12 selected genes

6.2 Support Vector Machines

The SVM models were trained on the dataset excluding patient 75, given the uncertainty around the diagnosis and hence the label for this example.

Initially, a linear SVM model was trained using gradient descent with scikit-learn's built in SGDClassifier class, with L1 penalisation, using Bayesian optimisation for the hyper parameter C and optimising for a given number of features. The initial experiment optimising for 30 features gave an accuracy of 0.95, recall of 0.91 and precision of 0.96. It was not possible to calculate AUC as the classifier used does not output class prediction probabilities. The low average recall in particular, suggested this simple approach of optimising for a number of features using an L1 penalty was be less effective for the SVM model than in the case of logistic regression.

The second approach used combined linear SVM with RFE, as outlined in 5.4.2. In order to limit computation time, the best 250 features were selected based on mutual information gain between the training data set and training labels and the RFE proceedure was used to reduce the remaining 250 features to a final set. A nested cross validation proceedure 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 experiment was repeated, optimising for each of 5, 10, 15, 20 and 30 features. The full results solving for 30 features are provided in 6.6.

	C	Accuracy	Recall	Specificity	Precision	AUC	False Positives	False Negatives
1	0.0	0.95	1.00	0.92	0.89	1.00	165a	-
2	35.24	0.95	0.89	1.00	1.00	0.96	-	149
3	14.93	1.00	1.00	1.00	1.00	1.00	-	-
4	66.26	0.95	0.90	1.00	1.00	0.99	-	149
5	0.01	1.00	1.00	1.00	1.00	1.00	-	-
6	0.02	0.95	1.00	0.92	0.88	1.00	165a	-
7	3.25	1.00	1.00	1.00	1.00	1.00	-	-
8	0.01	0.90	0.89	0.92	0.89	0.96	165a	149
9	9.13	1.00	1.00	1.00	1.00	1.00	-	-
Mean	n/a	0.97	0.96	0.97	0.96	0.99	n/a	n/a

Table 6.6: SVM: 30 Feature Model Performance (Excluding Patient 75)

The AUC at each number of features is detailed in 6.7. Again, it can be seen that the model provides near perfect classification, AUC > 0.99 at all levels.

Number of Features	Average AUC
30	0.99
20	0.99
15	0.99
10	0.99
5	0.98

Table 6.7: SVM: Performance vs Features Trade-Off

As for logistic regression, the final model was retrained on the full dataset, again excluding patient 75, filtering down to 250 features based on mutual information and optimising the model for 5 features using an RFE proceedure. The resulting model selected 5 features, with a hyperparameter C of 0.007:

UBE2Q2	ID3	P2RX1	GSN	LATS2

Table 6.8: SVM: Selected Features

6.3 Random Forest Results

As before, the random forest model was trained using the data set excluding patient 75. The random forest model performed very well. Selecting 30 features achieved an average AUC of 1.0. Table 6.9 gives the performance of the model over 9 iterations. Performance was not improved by increasing the number of estimators or maximum features evaluated at each node; all iterations selected 500, and 100 respectively.

	No. Est	Max Feats	Accuracy	Recall	Specificity	Precision	AUC	FP	FN
1	500	100	1.00	1.00	1.00	1.00	1.00	-	-
2	500	100	0.95	0.88	1.00	1.00	0.99	-	149
3	500	100	1.00	1.00	1.00	1.00	1.00	-	-
4	500	100	0.90	0.92	0.89	0.92	0.99	165a	149
5	500	100	1.00	1.00	1.00	1.00	1.00	-	-
6	500	100	1.00	1.00	1.00	1.00	1.00	-	-
7	500	100	0.95	1.00	0.92	0.90	1.00	165a	-
8	500	100	1.00	1.00	1.00	1.00	1.00	-	-
9	500	100	0.85	0.62	1.00	1.00	1.00	-	114, 116, 149
Total	n/a	n/a	0.96	0.94	0.98	0.98	1.00	n/a	n/a

Table 6.9: Random Forest: 30 Feature Model Performance (Excluding Patient 75)

Table 6.10 shows that performance was equally good with reduced number of features selected. A random forest model with only 5 features gave near perfect classification, AUC of 0.99 on average over 9 iterations.

Number of Features	Average AUC
30	1.00
20	1.00
15	0.99
10	1.00
5	0.99

Table 6.10: Random Forest: Performance vs Features Trade-Off

The final model was trained on the full dataset and the best five features identified. The final

features are given in Table 6.11.

GCA	LOC646849	SLC26A8	NSUN7	C20ORF100

Table 6.11: Random Forest: Selected Features

6.4 Results Summary

TBC

Chapter 7

Conclusions

Conclusions TBC

Chapter 8

Reflection

Reflections

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] 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

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

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

Chapter 9

Appendix

Github Repository: `https://github.com/parkyed/sepsis_ml_omics_msc`