An introduction to statistical classification and its application to Field Asymmetric Ion Mobility Spectrometry

Brian Azizi[a], Georgios Pilikos[b]

[a]*Department of Economics, University of Cambridge, Sidgwick Avenue, Cambridge, CB3 9DD, United Kingdom*
[b]*Laboratory for Scientific Computing, University of Cambridge, JJ Thomson Avenue, Cambridge, CB3 0HE, United Kingdom*

Abstract

This paper serves as an introduction to a particular area of Machine Learning, statistical classification, applied on medical data sets for automatic clinical diagnosis. An application of these models is illustrated in order to provide non-invasive diagnostics with the discovery of new gas volatile compounds (VOCs) that could provide the detection of fermentation profiles of patients with Inflammatory Bowel Disease (IBD). To achieve the above, an investigation of ten statistical classification algorithms from the supervised learning literature is undertaken. From this investigation, selected algorithms are applied on medical Field Asymmetric Ion Mobility Spectrometry (FAIMS) data sets to train classification models. From our results on the FAIMS data sets, we show that it is possible to classify unseen samples with a very high certainty and automatically perform medical diagnosis for Crohn's disease and Ulcerative Colitis. In addition, we propose potential future research on other data sets by utilizing the results from the identification of informative regions in feature space.

Keywords: Supervised Learning, Statistical Classification, Automatic Clinical Diagnosis, Feature Identification, Ion Mobility

1. Introduction

By utilizing the emerging technologies and available data around us, AI is increasingly being used in Medicine [1]. Development of algorithms that are able to process large amounts of data and produce valuable information is necessary. In the medical world, where decisions are of vital importance, utilization of medical history data can greatly enhance diagnosis. That is, by collecting samples from positively diagnosed and negatively diagnosed patients, it is possible to identify patterns or specific features that distinguish them for reliable future decision-making.

The scientific field that deals with this problem is called Machine Learning and a more relevant sub field called statistical classification from the supervised learning literature. A model is trained by giving it a number of examples, each belonging to a certain class. The aim is to use this model to accurately predict new, previously unseen examples. Doctors and practitioners can benefit from this technology since models can find patterns and structure in the data that was previously not possible. This can be achieved by the parallel intelligent processing of huge amounts of medical history data available in hospitals around the globe.

The application of these models on FAIMS data sets is inspired by a long tradition of clinicians that have been using their own sense of smell as a diagnostic tool. This traces even back to Hippocrates who suggested that a patient's odour could lead to their clinical diagnosis. Thus, the motives of using these data sets is due to the recent research on non-invasive diagnostics and the discovery of new gas volatile compounds (VOCs) biomarkers [2] [3] [4] [5] [6] that could provide a detection of fermentation profiles of patients with IBD. The pathogenesis of IBD involves the role of bacteria [4]. These bacteria ferment non-starch polysaccharides in the colon that produces a fermentation profile that can be traced in urine smell [4]. Using FAIMS instruments, it is possible to track the resultant VOCs that emanate from urine and identify patterns in their chemical fingerprints to automatically perform medical diagnosis for Crohn's disease and Ulcerative Colitis.

In this paper, we provide a review of classification techniques and test some on medical FAIMS data sets. From our results on the FAIMS data sets, we show that it is possible to classify unseen samples with a very high certainty on certain data sets and propose potential fu-

ture research on other data sets that would potentially allow training for more accurate classification models. Specific informative regions that play a vital role for the creation of the models' decision boundaries on the data sets are also identified and illustrated which are worth investigating further.

The paper is organized as follows. Firstly, in section 2, an in-depth overview of the theory behind the classification methods is given describing potential advantages and disadvantages during training and testing phases along with a description of each algorithm's implementation. In section 3, testing of a subset of these algorithms is described on numerous data sets to investigate their practical performance. In section 4, an introduction to FAIMS technology is given along with the application of selected algorithms on medical FAIMS data sets by testing various scenarios. Finally, a discussion about future research and final remarks can be found in sections 5 and 6 respectively.

2. Methods

Fundamental definitions and notation:

Classification is a form of supervised machine learning. We train a model by using a large number of examples, each belonging to a certain *class*. Our aim is to use the model to accurately predict new, previously unseen examples.

We have K discrete *classes*, that we will index by the letter c, i.e. $c \in \{1, 2, \ldots, K\}$. We have a *training set* containing N *training examples*. Each example consists of two elements, namely the *input vector* (or *feature vector*), denoted by $\mathbf{x}_i$, and the corresponding *label*, denoted by y_i. We will use the letter i to index the training examples, so that $i \in \{1, 2, \ldots, N\}$.

Each label y_i is an integer between 1 and K, indicating the class of training example i. Each input vector $\mathbf{x}_i$ is a column vector containing the values of the *features* of example i in its components. We let D be the total number of features and use $j \in \{1, 2 \ldots, D\}$ to index the features of our input vectors, so that $x_{i,j}$ denotes the j^{th} feature of the i^{th} training example.

We let $\mathbf{X} = [\mathbf{x_1}\, \mathbf{x_2}\, \ldots\, \mathbf{x_N}]^T$ stand for the $N \times D$ matrix containing the training examples in its rows and the input features in its columns. We will also use $\mathbf{y} = [y_1\, y_2 \ldots y_N]^T$ to denote the N-dimensional column vector containing the class of the i^{th} training example in row i.

Finally, when discussing how to make new predictions based on the trained model, we will use $\mathbf{x}^*$ to denote the feature vector of a previously unseen example. Its true class will be labelled by y^* and our prediction of the class will be $\hat{y}^*$.

2.1. *Linear Discriminant Analysis*

Linear Discriminant Analysis (LDA) [7] aims to separate the classes in feature space using linear decision surfaces (*hyperplanes*).

We need to make two slight modifications to our notation. Firstly, we need to attach a dummy 'input' feature $x_{i,0} = 1$ to our input vectors $\mathbf{x}_i$ so that $\mathbf{x}_i = [1\, x_{i,1}\, x_{i,2} \ldots\, x_{i,D}]^T$. Secondly, we will use an alternative representation of our classes: Instead of labels y_i, we shall use target vectors $\mathbf{t}_i$ of length K, where the c^{th} component of $\mathbf{t}_i$ is equal to 1 if training example i is in class c and 0 otherwise (i.e. $t_{i,c} = 1$ if $y_i = c$).

Classification: For LDA, the discriminant function takes the form

$$\hat{y}^* = \arg\max_c (\mathbf{w}_c^T \mathbf{x}^*). \tag{1}$$

That is, we predict $\mathbf{x}^*$ to be in class c which maximizes the expression $\mathbf{w}_c^T \mathbf{x}^*$.

$\mathbf{w}_c$ is a $(D + 1)$-dimensional vector containing the *weight parameters* of the model for class c. The boundary between class c and class d is given by $\mathbf{w}_c^T \mathbf{x} = \mathbf{w}_d^T \mathbf{x}$, so that $(\mathbf{w}_d - \mathbf{w}_c)$ denotes the normal vector of the decision plane.

There is a nice interpretation to the quantity $\mathbf{w}_c^T \mathbf{x}^*$. We can treat it as an estimate of the probability that $\mathbf{x}^*$ belongs to class c, that is

$$p(y^* = c \,|\, \mathbf{w}_c, \text{Data}) = \mathbf{w}_c^T \mathbf{x}^*. \tag{2}$$

Training: The goal of the training phase is to learn the weight parameters $\mathbf{w}_c$ for each class c. We achieve this by minimising an *error function*. The optimization objectives are given by:

$$E(\mathbf{w}_c) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}_c^T \mathbf{x}_i - t_{i,c})^2, \qquad c \in \{1, \ldots, K\}. \tag{3}$$

This is called the *least squares* error function, and we minimize one per class. It is the sum of squares of the prediction errors resulting from a particular choice of weight vector $\mathbf{w}_c$. We aim to find $\mathbf{w}_c$ for which this is the smallest.

Differentiating with respect to $\mathbf{w}_c$ we find that the optimal weight vector satisfies

$$\frac{\partial E(\mathbf{w}_c)}{\partial \mathbf{w}_c} = \sum_{i=1}^{N} (\mathbf{w_c}^T \mathbf{x}_i - t_{i,c}) \mathbf{x}_i = 0. \tag{4}$$

This problem has in fact a closed form solution, and we can state it concisely using matrices. To do that, let $\mathbf{W} = [\mathbf{w}_1\, \mathbf{w}_2\, \ldots \mathbf{w}_K]$ be the matrix containing the weight vectors for all the classes in its columns and let $\mathbf{T} = [\mathbf{t}_1\, \mathbf{t}_2\, \ldots \mathbf{t}_N]^T$ be the matrix containing all the target vectors in its rows.

The solution to LDA can then be written as

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}. \tag{5}$$

The most expensive part in the computation is inverting $\mathbf{X}^T\mathbf{X}$.

Algorithm 2.1: LDA($\mathbf{X}, \mathbf{y}$)

Create $N \times K$ matrix $\mathbf{T}$, where
$T_{i,c} = 1$ if $y_i = c$ and 0 otherwise.

comment: Training Phase:

Compute $\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}$.

comment: Classification:

Compute vector $\mathbf{W^T x}^*$and find largest row k.
Predict $\hat{y}^* = k$.

Discussion: LDA has the advantage of giving a closed-form solution for the weight vectors $\mathbf{w}_c$. Furthermore, the model is computationally inexpensive and easy to interpret.

We can also interpret the quantity $\mathbf{w}_c^T\mathbf{x}^*$ as the probability that $\mathbf{x}^*$ belongs to class c. However, this interpretation may break down on some occasions, as there are no constraints to ensure that $\mathbf{w}_c^T\mathbf{x}^* \in [0, 1]$ for all possible vectors $\mathbf{x}^*$.

As it is a linear method, LDA cannot handle non-linear class boundaries very well. Furthermore, even in the case of linearly separable classes, it may not find the optimal decision boundary. Outliers that are "too correct" in that they lie a long way on the correct side of the decision surface have a disproportionate effect on LDA. The error function penalizes these outliers too heavily and the result is that LDA shifts the boundary in their direction, at the expense of accuracy.

The algorithm's steps can be seen in Algorithm 2.1. For a more detailed derivation and thorough discussion, see [7] [8].

2.2. *Fisher Discriminant Analysis*

Related to LDA is the idea of *Fisher Discriminant Analysis* (FDA) [7] [9].

Computing the quantities $\mathbf{w}_c^T\mathbf{x}_i$ can be interpreted as a form of dimensionality reduction. We take high dimensional feature vectors $\mathbf{x}_i$ and project them onto one dimension (i.e. onto a line).

Generally, dimensionality reduction leads to a considerable loss of information. However, we can adjust $\mathbf{w}$ to find the line that minimizes the overlap between the classes when projecting them onto it.

The goal of FDA is to do just that by maximising the *Fisher Criterion*, which is defined below. The intuition behind this is to get the largest separation between the classes by maximizing the between-class variance, while simultaneously minimizing the within-class variance, thereby reducing the spread of the individual classes. This should give us the smallest possible class overlap when projected onto one dimension.

As in the case of LDA, we need to introduce a dummy feature $x_{i,0} = 1$ to each example $\mathbf{x}_i$. However, we will adopt a 2-class setting to explain FDA. For that, we take $y_i \in \{0, 1\}$.

Training: FDA trains the weight vector $\mathbf{w}$ by maximising the Fisher Criterion, $J(\mathbf{w})$. To define the criterion, let $\mathbf{m}_0$ and $\mathbf{m}_1$ be the *class mean* vectors, given by

$$\mathbf{m}_0 = \frac{1}{N_0}\sum_{i\,:\,y_i=0}\mathbf{x}_i\,, \qquad \mathbf{m}_1 = \frac{1}{N_1}\sum_{i\,:\,y_i=1}\mathbf{x}_i\,, \tag{6}$$

where $N_1 = \sum_{i=1}^{N} y_i$ is the total number of training examples in class 1 and $N_0 = N - N_1$ is the total number of training examples in class 0.

Let $\mathbf{S_B}$ be the *between-class* covariance matrix:

$$\mathbf{S_B} = (\mathbf{m}_1 - \mathbf{m}_0)(\mathbf{m}_1 - \mathbf{m}_0)^T\,, \tag{7}$$

and let $\mathbf{S_W}$ be the *within-class* covariance matrix, given by

$$\begin{aligned}\mathbf{S_W} = &\sum_{i\,:\,y_i=1}(\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^T \\ &+ \sum_{i\,:\,y_i=0}(\mathbf{x}_i - \mathbf{m}_0)(\mathbf{x}_i - \mathbf{m}_0)^T\,.\end{aligned} \tag{8}$$

Then the Fisher Criterion is given by

$$J(\mathbf{w}) = \frac{\mathbf{w^T S_B w}}{\mathbf{w^T S_W w}} \tag{9}$$

We see that the between-class covariance is in the numerator while the total within-class covariance is in the denominator, so that maximisation of this criterion should give us the desired optimal separation.

The result of the optimization is:

$$\mathbf{w} \propto {\mathbf{S_W}}^{-1}(\mathbf{m}_1 - \mathbf{m}_0). \tag{10}$$

This is known as *Fisher's Linear Discriminant*. It is, however, not quite yet a discriminant, but only a direction for projection. The projected data can then be used to construct a discriminant by choosing a threshold τ. There are various methods for choosing the threshold. Often, we assume that the features are normally distributed and find the value of τ that maximizes the posterior class probabilities, or equivalently, minimizes the misclassification rate.

Algorithm 2.2: FDA($\mathbf{X}, \mathbf{y}$)

comment: Training Phase:

Compute $\mathbf{w} \propto {\mathbf{S_W}}^{-1}(\mathbf{m}_1 - \mathbf{m}_0)$, where
  $\mathbf{m_1}, \mathbf{m_0}$ and $\mathbf{S_W}$ are given by (6) and (8)
Find optimal threshold τ, by minimising
  the misclassification rate.

comment: Classification:

if $\mathbf{w}^T\mathbf{x}^* \geq \tau$
  then $\hat{y}^* = 1$,
  else $\hat{y}^* = 0$.

Classification: To classify a new data point $\mathbf{x}^*$, we compute the quantity $\mathbf{w}^T\mathbf{x}^*$. Then, we set a threshold τ and predict:

$$\begin{cases} \hat{y}^* = 1 & \textbf{if} \quad \mathbf{w}^T\mathbf{x}^* \geq \tau \\ \hat{y}^* = 0 & \textbf{if} \quad \mathbf{w}^T\mathbf{x}^* < \tau \end{cases} \tag{11}$$

Discussion: FDA is a useful method for reducing the dimensionality of a machine learning problem by projecting the features onto a lower-dimensional domain. It handles classes with linear boundaries very well.

In practice, it lacks flexibility. The method of Kernel Fisher Discriminants [10] builds on FDA and extends it to non-linear class boundaries using the *kernel trick* [11].

The algorithm's steps can be seen in Algorithm 2.2. For a more detailed description of FDA, including a multi-class treatment, see [8] [9].

2.3. Adaptive Boosting (AdaBoost)

Boosting belongs to a family of methods called *ensembles*. It combines a number of base classifiers to form a *commitee*. We will describe a particular method of boosting, named *AdaBoost* [12].

AdaBoost can produce substantial improvements in performance compared to using just a single classifier. It can give good results even if we only use base classifiers that have a performance which is only slightly better than random (known as "Weak Learners") [12].

To classify a new example, we combine all the trained base classifiers to form a majority vote. A weights is also associated with each base classifier. The votes of classifiers that performed relatively better during training, count more.

We will explain the algorithm in a 2-class setting. For this algorithm, it is useful to use the labels $y_i \in \{-1, 1\}$ and let $+1$ stand for the positive and -1 for the negative class.

Training: We will train a total of M base classifiers, denoted by $f_m(\mathbf{x})$ with $m \in \{1, \ldots, M\}$. It is common to use *tree stumps* as base classifiers for each iteration. These are weak learners that attempt to separate the classes using *axis-alligned* hyperplanes, i.e. hyperplanes that use one of the axis of the feature space as the direction of their normal. We let $w_i^{(m)}$ stand for the weight of the i^{th} training example just before it is passed on to classifier $f_m(\mathbf{x})$, i.e. during the m^{th} iteration. Initially, the examples receive equal weights so as to have an unbiased initial state and be able to adapt according to the misclassification error only:

$$w_i^{(1)} = \frac{1}{N} \text{ for all } i \in \{1, \ldots, N\}. \tag{12}$$

In each iteration m, we train the base classifier f_m by minimizing the error function :

$$J_m = \sum_{i=1}^{N} w_n^{(m)} \mathbb{I}\left(f_m(\mathbf{x}_i) \neq y_i\right), \tag{13}$$

with respect to the parameters of the classifier $f_m(\mathbf{x})$, where $\mathbb{I}(\bullet)$ is the indicator function which outputs 1 if the condition inside the brackets holds true and 0 otherwise.

Next, we need to compute the quantities:

$$\epsilon_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(f_m(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(m)}}, \tag{14}$$

and

$$\alpha_m = \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right). \tag{15}$$

We use these to update the weights for the next iteration:

$$w_i^{(m+1)} = w_i^{(m)} \exp\left(\alpha_m \mathbb{I}(f_m(\mathbf{x}_i) \neq y_i)\right). \tag{16}$$

Classification: Once we have trained all base classifiers, we can use the ensemble to predict the class of a new data point $\mathbf{x}^*$:

$$\hat{y}^* = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}^*)\right). \tag{17}$$

Algorithm 2.3: AdaBoost($\mathbf{X}, \mathbf{y}, f_1, \ldots, f_M$)

comment: Initialize weights

$w_i^{(1)} \leftarrow 1/N$ **for all** $i \in \{1, \ldots, N\}$

for $m \leftarrow 1$ **to** M

do
- **comment:** Train $f_m(\mathbf{x})$ by
- $\min J_m = \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(f_m(\mathbf{x}_i) \neq y_i)$
- **comment:** Evaluate quantities
- $\epsilon_m \leftarrow \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(f_m(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(m)}}$
- $\alpha_m \leftarrow \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
- **comment:** Update data weights
- $w_i^{(m+1)} = w_i^{(m)} \exp\left(\alpha_m \mathbb{I}(f_m(\mathbf{x}_i) \neq y_i)\right)$

comment: Predict class of new pattern $\mathbf{x}^*$

$\hat{y}^* = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}^*)\right)$

Discussion: AdaBoost is not a classifier per se, but rather a method of combining a number of classification methods.

Decision tree stumps are the traditional choice for the base classifiers. But AdaBoost can use any classification method as a base classifier and for some applications, there are more appropriate base classifiers than decision trees (such as the Haar classifiers in the problem of face detection).

As mentioned above, we can expect the ensemble to converge to a strong classifier, even if the individual base classifiers are only slightly better than random guessing.

However, the choice of base classifiers has an effect on the speed of convergence. In practice, we can make use of prior knowledge about the structure of a data set to help us choose the weak learners.

Boosting generally shows resistance to over-fitting. However, it is affected by misclassified data points that are far away from the decision boundary (i.e. outliers). These outliers tend to create "spikes" in the decision boundary.

The algorithm's steps can be seen in Algorithm 2.3. For a more detailed derivation and discussion, along with a multi-class extension, see [12] [13].

2.4. *Random Forests*

Random Forests [14] is another ensemble learning method. The idea is to train a multitude of *decision trees*, each with some degree of randomization. To classify new data points, we let each tree make a prediction and then average their output to form an ensemble prediction. We randomize by choosing a random subset of our training data for the training of each tree. Furthermore, when performing the optimization in the training phase, we only optimize over a random subset of the available parameters. The reason for this randomization is to ensure that the trees grow differently from one another, increasing the confidence of our final prediction.

A *tree* is a hierarchical structure consisting of *nodes* connected by *edges*. Nodes are divided into *internal* (or *split*) *nodes* and *terminal nodes* (or *leafs*). Each node has exactly one incoming edge, except the uppermost node called the *root* which has none. We will focus on *binary* trees which have exactly two outgoing edges.

Classification: Let T be our *forest size*, i.e. the total number of trees grown and index the trees using $t \in \{1, 2 \ldots, T\}$.

We can use a tree t to classify a new data point $\mathbf{x}^*$. We start by placing $\mathbf{x}^*$ at the root. Each internal node S_m (including the root) of the tree has a predefined *test function* $\phi_m(\mathbf{x})$. In our model, the test function will simply be one of the features of $\mathbf{x}$:

$$\phi_m(\mathbf{x}^*) = x^*_{j_m}, \tag{18}$$

where j_m is optimal in some sense and determined during the training phase.

Depending on whether this feature is above or below a certain *threshold* τ_m, we send $\mathbf{x}^*$ through either the left or right edge, to the next node. We push $\mathbf{x}^*$ through the tree in this manner until we reach a leaf S_L. The leafs contain the conditional distribution of the classes, $p_t(y^* = c \,|\, \mathbf{x}^* \in S_L)$. If we used a single tree for classification, we would simply predict $\mathbf{x}^*$ to be in the class for which this probability is the greatest.

The Random Forests method pushes $\mathbf{x}^*$ through all the trees simultaneously and predicts the class that max-

imizes the average of the posterior probabilities:

$$\hat{y}^* = \arg\max_c \left(\frac{1}{T} \sum_{t=1}^{T} p_t(y^* = c \,|\, \mathbf{x}^* \in S_L) \right). \quad (19)$$

Training: The training phase is in charge of finding the best feature j_m to split on at each node m of each tree t along with the optimal threshold τ_m for it. The Random Forests algorithm does this by maximizing the expected *information gain* (or *entropy reduction*), denoted by $IG_m(j_m, \tau_m)$ as defined in (22). To formalize the information gain, we need to define further variables. Given a particular choice of parameters, let $S_m^{(R)}$ and $S_m^{(L)}$ be the subsets of data points in node m that are sent through the right edge and left edge, respectively. We define the *Shannon entropy* of a node S to be

$$H(S) = \sum_{c=1}^{K} p(y = c \,|\, \mathbf{x} \in S) \log\left(p(y = c \,|\, \mathbf{x} \in S)\right), \quad (20)$$

where we use the empirical distributions of the classes in the node to approximate the probability:

$$p(y = c \,|\, \mathbf{x} \in S) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{I}(y_i = c). \quad (21)$$

We can then define the expected information gain associated with a particular choice (j_m, τ_m) of parameters for node m to be

$$IG_m(j_m, \tau_m) = H(S_m) - \sum_{r \in \{R,L\}} \frac{|S_m^{(r)}|}{|S_m|} H(S_m^{(r)}). \quad (22)$$

Each tree is grown independently of all the other trees and we inject randomness into the training process of each tree.

The randomization of each tree is two-fold. Instead of using all N available training examples to train the tree, we randomly select αN data points from the training set *with replacement* (this is called *bootstrapping*). α is one of the parameters of the model. It is a positive number usually set to be less than 1. When maximizing the information gain criterion to choose the optimal parameters for a node, we do not make all possible choices of features. Instead, we maximize $IG(j_m, \tau_m)$ only over a random subset of features j_m, of size M.

Finally we have to decide on when to stop growing a tree. If a tree is grown too deeply, it will overfit the data and not generalize well. If it is not deep enough, it will not give very strong predictions. Standard stopping criteria allow a tree to grow until a node contains data points of only one class (a *pure* node) or until the number of data points in a node is below a certain threshold. Random Forests are very powerful models. Each individual tree may not be a very strong classifier by itself.

However, by randomizing the training of the trees and combining their output, we can obtain a very accurate and flexible classifier. One advantage of this technique is that the trees can be grown independently from one another. This allows us to improve the computational time of the forest, by growing the trees in parallel over multiple cores. Furthermore, Random Forests give us a measure of the relative importance of features. If a feature is in the first few layers in many decision trees, it is likely to separate the classes well. We also store the histogram of the classes in all the leafs, giving us a measure for the confidence of our predictions.

A potential disadvantage is that trees are prone to overfit the training data if we grow them too deeply. The algorithm's steps can be seen in Algorithm 2.4. For a more detailed discussion and explanation, see [14] [15].

Algorithm 2.4: RANDOM FOREST($\mathbf{X}, \mathbf{y}, T, \alpha, M$)

comment: Training Phase

for $t \leftarrow 1$ **to** T

do
- Draw a bootstrap sample of size αN
- **comment:** Grow tree starting at root
- $m \leftarrow 0$
- **if** (stopping criterion is not met)
  - **do**
    - Select M features randomly
    - Optimize over these features:
    - $\max_{j_m, \tau_m}[IG_m(j_m, \tau_m)]$
    - Grow two daughter nodes.
- $m \leftarrow m + 1$

comment: Classification

Send $\mathbf{x}^*$ through all trees.

Find $p_t(y^* = c \,|\, \mathbf{x}^* \in S_L)$ for all t and c,
where S_L are the leafs

$\hat{y}^* = \arg\max_c \left(\frac{1}{T} \sum_{t=1}^{T} p_t(y^* = c \,|\, \mathbf{x}^* \in S_L) \right)$

2.5. *Naive Bayes*

Naive Bayes [16] is a simple probabilistic classification method. The algorithm estimates the class-conditional distributions of each input feature. If different classes show different distributions of the features, *Naive Bayes* will be able to use that information to separate the classes.

To explain the intuition behind the Naive Bayes algorithm we adopt a different framework than before, namely we assume that our input features are binary, $x_{i,j} \in \{0, 1\}$. For example, $\mathbf{x}_i$ could summarize the presence of certain chemicals in sample i, i.e. $x_{i,j} = 1$ if chemical j is present in sample i and 0 otherwise. There exist modifications to handle continuous input data, for example by assuming the features are normally distributed.

The model has two sets of parameters:

1. $\theta_{j,c} = p(x_{i,j} = 1|y_i = c)$ is the probability of feature j being 1 given that the sample is in class c
2. $\pi_c = p(y_i = c)$ are the prior class probabilities

The key assumption for this model is that the features are conditionally independent given the class labels (hence "Naive"):

$$p(\mathbf{x}_i|\theta, y_i = c) = \prod_{j=1}^{D} p(x_{i,j}|\theta, y_i = c). \tag{23}$$

Ideally, different classes show different feature distributions allowing us to find a separation between them, using the Naive Bayes method.

Using Bayes Theorem, we get

$$p(\mathbf{y}|\mathbf{x}, \theta, \pi) \propto p(\mathbf{y}|\pi) \times p(\mathbf{x}|\mathbf{y}, \theta), \tag{24}$$

where

$$p(\mathbf{y}|\pi) \propto \prod_{i=1}^{N} \prod_{c=1}^{K} \pi_c^{\mathbb{I}(y_i=c)} \tag{25}$$

and

$$\begin{aligned} p(x_{i,j}|y_i = c, \theta) \propto \theta_{j,c}^{\mathbb{I}(y_i=c)\mathbb{I}(x_{i,j}=1)} \\ \times(1 - \theta_{j,c})^{\mathbb{I}(y_i=c)\mathbb{I}(x_{i,j}=0)}. \end{aligned} \tag{26}$$

The full model is then

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}, \theta, \pi) \propto \prod_{i=1}^{N} \prod_{c=1}^{K} (\pi_c^{\mathbb{I}(y_i=c)} \prod_{j=1}^{D} \theta_{j,c}^{\mathbb{I}(y_i=c)\mathbb{I}(x_{i,j}=1)} \\ (1 - \theta_{j,c})^{\mathbb{I}(y_i=c)\mathbb{I}(x_{i,j}=0)}). \end{aligned} \tag{27}$$

Training Training of the Naive Bayes classifier involves estimation of the parameters $\{\theta_{j,c}, \pi_c\}$ of the model. In order to do that, we define N_c to be the total number of training data points in class c and $N_{j,c}$ to be the training data points in class c for which feature j was equal to 1. That is,

$$\begin{cases} N_c = \sum_{i=1}^{N} \mathbb{I}(y_i = c) \\ N_{j,c} = \sum_{i=1}^{N} \mathbb{I}(y_i = c)\mathbb{I}(x_{i,j} = 1). \end{cases} \tag{28}$$

Intuitively, we would then expect:

$$\hat{\pi}_c = N_c/N \qquad \hat{\theta}_{j,c} = N_{j,c}/N_c \tag{29}$$

to be the best estimates of the parameters and it can indeed be shown that these give the maximum likelihood.

Algorithm 2.5: NAIVE BAYES($\mathbf{X}, \mathbf{y}$)

comment: Training Phase

comment: Initialize count parameters

$N_j \leftarrow 0\, N_{j,c} \leftarrow 0$
for $i \leftarrow 1$ **to** n
do $\begin{cases} c \leftarrow y_i \\ N_c \leftarrow N_c + 1 \\ \textbf{for } j \leftarrow 1 \textbf{ to } D \\ \quad \textbf{do} \begin{cases} \textbf{if } x_{i,j} = 1 \\ \quad \textbf{then } N_{j,c} \leftarrow N_{j,c} + 1 \end{cases} \end{cases}$
$\hat{\pi}_c \leftarrow N_c/N,\ \hat{\theta}_{j,c} \leftarrow N_{j,c}/N_c$

comment: Predict class of new pattern x^*

for $c \leftarrow 1$ **to** K **evaluate**
$p(y^* = c|x^*) \propto \hat{\pi}_c \prod_{j=1}^{d} \hat{\theta}_{j,c}^{\mathbb{I}(x_j^*=1)}(1 - \hat{\theta}_{j,c}^{\mathbb{I}(x_j^*=1)})$
Normalize probabilities.
$\hat{y}^* = \arg\max_c p(y^* = c \mid x^*)$

Classification: Once we have these estimates, we can make predictions on the posterior class probabilities of previously unseen data points.

Given a new data point $\mathbf{x}^*$, the posterior probability that $\mathbf{x}^*$ is in class c is given by

$$p(y^* = c|\mathbf{x}^*) \propto \hat{\pi}_c \prod_{j=1}^{D} \hat{\theta}_{j,c}^{\mathbb{I}(x_j^*=1)}(1 - \hat{\theta}_{j,c}^{\mathbb{I}(x_j^*=0)}).$$

We evaluate this expression (up to proportionality) for all classes and then normalize the probabilities so that

$$\sum_{c \in \{1,\ldots,K\}} p(y^* = c \mid \mathbf{X}^*) = 1. \tag{30}$$

Finally, we choose the class $y^* = c$ for which the probability $p(y^* = c|\mathbf{x}^*)$ is largest.

Discussion: Even though conditionally independent features is a stark assumption for most settings, the Naive Bayes classifier tends to give relatively accurate predictions in practice. The algorithm's steps can

be seen in Algorithm 2.5. The training phase is computationally very efficient as it essentially boils down to bookkeeping. Alternative classifiers have been proposed that allow some degree of interdependence between features [17].

In practice, a major danger when working with this classification method is that prediction involves the multiplication of many small numbers. This means that rounding errors have a big effect on predictions and it is not uncommon for underflow to occur. One way to deal with this problem is to make use of the *log-sum-exp trick*. Algorithm 2.6 explains how to implement Naive Bayes predictions using the log-sum-exp trick.

Algorithm 2.6: NB PREDICTION $(\mathbf{x}^*, \hat{\pi}, \hat{\theta})$

for $c \leftarrow 1$ **to** K
do $\begin{cases} L_c = \log \hat{\pi}_c \\ \textbf{for } j \leftarrow 1 \textbf{ to } D \\ \quad \textbf{do} \begin{cases} \textbf{if } x^*_j = 1 \\ \quad \textbf{then } L_c \leftarrow L_c + \log \hat{\theta}_{j,c} \\ \quad \textbf{else } L_c \leftarrow L_c + \log(1 - \hat{\theta}_{j,c}) \end{cases} \end{cases}$
$p_c = \exp(L_c - \log(\sum_{c'=1}^{K} \exp L_{c'}))$
$\hat{y}^* = \text{argmax}_c(p_c)$

2.6. *Logistic Regression*

Logistic regression is another probabilistic approach to classification. We will describe the method in a 2-class framework with each class label $y_i \in \{0, 1\}$ (see Section 2.2). Furthermore, we need to include a *bias* in our input data points, i.e. we introduce a dummy feature $x_{i,0} = 1$ to each data point $\mathbf{x}_i$.

Classification To classify a new data point $\mathbf{x}^*$, the logistic regression technique attaches a weight w_j to each feature x^*_j (where $j \in \{0, 1, \ldots, D\}$). It then passes the weighted sum $\sum_{j=0}^{D} w_j x^*_j = \mathbf{w}^T\mathbf{x}^*$ to the sigmoid function, given by:

$$g(z) = \frac{1}{1 + \exp(-z)}, \tag{31}$$

We interpret the output of the function as the probability that $\mathbf{x}^*$ is in class 1:

$$p(y^* = 1 \,|\, \mathbf{w}) = g(\mathbf{w}^T\mathbf{x}^*). \tag{32}$$

Finally, we can classify $\mathbf{x}^*$ by setting a threshold τ on the probability and predicting:

$$\begin{cases} \hat{y}^* = 1 & \textbf{if} \quad g(\mathbf{w}^T\mathbf{x}^*) \geq \tau \\ \hat{y}^* = 0 & \textbf{if} \quad g(\mathbf{w}^T\mathbf{x}^*) < \tau \end{cases} \tag{33}$$

Typically, we choose $\tau = 0.5$ in order to make predictions symmetric. The sigmoid has the property that it is naturally constrained to lie in the interval $(0, 1)$. This ensures that our interpretation of its output as a probability does not break down as was the case with LDA.

We can easily compute the decision boundary of our model. $g(\mathbf{w}^T\mathbf{x}) = 0.5$ is equivalent to $\mathbf{w}^T\mathbf{x} = 0$. So the decision boundary is given by $\mathbf{w}^T\mathbf{x} = 0$. For a general threshold τ, the boundary is given by $\mathbf{w}^T\mathbf{x}^* = g^{-1}(\tau)$.

We can extend the method to $K > 2$ classes by using a *one-versus-the-rest* classifier. We train a total of $K - 1$ 2-class classifiers, each of which solves the problem of separating points belonging to one particular class from points not in that class. The final prediction, $\hat{y}^*$, is made by comparing the output of all the classifiers and choosing the class corresponding to the highest probability.

Training: The cost function can be derived using maximum likelihood estimation of the weights $\mathbf{w}$. It can be shown that this is convex and local-optima free.

Intuitively, the cost function heavily penalizes high confidence predictions of the wrong class. The training phase is in charge of finding the weights $\mathbf{w}$ of the model. This is done by minimizing:

$$\begin{aligned} J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(g(\mathbf{w}^T\mathbf{x}_i) \\ +(1 - y_i)\log\left(1 - g(\mathbf{w}^T\mathbf{x}_i)\right)\Big) \end{aligned} \tag{34}$$

Noting that for the sigmoid function, $g(z)$, we have:

$$\frac{dg}{dz} = g(z)(1 - g(z)), \tag{35}$$

we can differentiate $J(\mathbf{w})$ with respect to $\mathbf{w}$ to get:

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{w}} = -\frac{1}{N}\sum_{i=1}^{N} y_i(1 - g(\mathbf{w}^T\mathbf{x}_i) \\ +(1 - y_i)g(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i \end{aligned} \tag{36}$$

There is no closed-form solution to this minimization problem, thus gradient descent or Newton's Method for finding minima is required. The steps of the algorithm are summarized in Algorithm 2.7.

2.7. *K-Nearest Neighbours*

K-Nearest-Neighbours (K-NN) [18] is a non-parametric approach to density estimation. This means that we do not make any assumptions about the functional form of the distribution to be estimated. The

method can be extended to the problem of classification. To classify a new data point $\mathbf{x}^*$, K-NN identifies the K points from the training set that are closest to $\mathbf{x}^*$ and then assigns it to the class which has the most points in this set.

Algorithm 2.7: LOGISTIC REGRESSION($\mathbf{X}, \mathbf{y}, \tau$)

comment: Training Phase:

$\mathbf{w} = \arg\min_{\mathbf{w}} J(\mathbf{w})$

comment: Classify new data point $\mathbf{x}^*$

if $\left(1 + \exp(\mathbf{w}^T\mathbf{x}^*)\right)^{-1} \geq \tau$
  then $\hat{y}^* = 1$
  else $\hat{y}^* = 0$

comment: Confidence of estimate

$p(y^* = 1 \,|\, \mathbf{w}) = \left(1 + \exp(\mathbf{w}^T\mathbf{x}^*)\right)^{-1}$

Algorithm 2.8: K-NN($\mathbf{X}, \mathbf{y}, K$)

Store the labelled training examples $(\mathbf{X}, \mathbf{y})$

comment: Classify a new pattern $\mathbf{x}^*$:

for $i \leftarrow 1$ **to** N
  Evaluate $d_i = \|\mathbf{x}^* - \mathbf{x}_i\|$

Find $i_1, \ldots, i_K$ which minimize $\sum_{j=1}^{K} d_{i_j}$.
$\hat{y}^* = \text{mode}\{y_{i_1}, \ldots, y_{i_K}\}$.

Training: The training phase of the K-NN algorithm only involves storing the training data in feature space along with the corresponding class labels.

Classification: Before we make any classifications, we have to decide on the parameter K for the model. Once we have chosen a specific value for K, we can feed the algorithm a new data point $\mathbf{x}^*$. It identifies the K training examples in feature space that are closest to it and then looks for the class c that occurs most frequently among those points. Finally, it predicts $\mathbf{x}^*$ to be in class c. If there is a tie, it can be broken at random or by assigning the class of the nearest point among the tied groups to $\mathbf{x}^*$. When dealing with continuous input data, we generally use the euclidean metric. However, the method works with any valid metric and the optimal choice will depend on the specific data set.

Discussion: The K-NN approach requires the entire data set to be stored in working memory. Computing the distances to the training examples will become computationally expensive for large data sets. One way to deal with this problem is to use an appropriate nearest neighbour search algorithm which searches for the nearest neighbours by only evaluating the distance metric over a subset of all training examples.

Another disadvantage of K-NN is that in the case of skewed class distributions, the algorithm tends to bias predictions in favour of the classes that are overrepresented. We can mitigate this drawback by giving each training examples a weight that is inversely proportional to its distance from the new data point.

The constant K determines the degree of smoothing in the classification process. A small value of K will lead to many small regions of each class, while a large value of K produces fewer larger regions. This reduces the effect of noise in the data but may also lead to overseeing certain structures in the data. The algorithm's steps can be seen in Algorithm 2.8.

*2.8. **Support Vector Machine***

The Support Vector Machine (SVM) [19] is a sparse kernel classifier. That is, it transforms the training data points into another domain (using specific kernel with basis functions) and from that domain chooses only a few basis functions to create a classifier.

In mathematical terms, the model created is defined by the following function:

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i=1}^{N} w_i \boldsymbol{\phi}_i(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b \qquad (37)$$

where the output is a linear combination of the selected basis functions and b is a bias variable.

Training: It utilizes a sparse vector of weights which effectively choose the basis functions to be included in the model. This is done by attempting to minimize a measure of the error in the training set and at the same time maximize the margin between the classes. The margin is defined as the perpendicular distance between the hyper-plane of the classifier and the closest of the data points of any class. The maximum margin solution is found by solving:

$$arg \ \max_{\boldsymbol{w},b} \left\{ \frac{1}{\|\boldsymbol{w}\|} \min_n \left[y_n(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) + b) \right] \right\} \qquad (38)$$

where y_n is the corresponding class of the data point. This is rearranged into the following Lagrangian function with the introduction of N Lagrangian multipliers [19]:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ y_n(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) + b) - 1 \right\} \tag{39}$$

Setting the derivatives of $L(\boldsymbol{w}, b, \boldsymbol{a})$ with respect to $\boldsymbol{w}$ and b to zero and substituting back to the Lagrangian function, the dual representation of the maximum margin problem is formed which is solved only with respect to $\boldsymbol{a}$.

This takes the form of a quadratic programming problem and can be solved using various solvers. From their solution, $\boldsymbol{a}$ is obtained where the majority of the elements are zero. The non-zero elements of $\boldsymbol{a}$ are called the support vectors and correspond to the data points that lie on the maximum margin of the hyperplane.

Classification: In order to classify a new data point $\boldsymbol{x}^*$, the sign of the following function is evaluated:

$$f(\boldsymbol{x}^*) = \sum_{n=1}^{N} a_n \, y_n \, \boldsymbol{\phi}(\boldsymbol{x}^*) + b \tag{40}$$

*2.9. **Relevance Vector Machine***

The Relevance Vector Machine (RVM) [20] [21] was proposed as a model of identical functional form to the SVM but with superior characteristics. It exploits a probabilistic Bayesian learning framework that provides the level of trustworthiness of each classified point.

That is, given a data set $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, we have the identical model for SVM without the bias:

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i=1}^{N} w_i \boldsymbol{\phi}_i(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{41}$$

Training: For two-class classification, the goal is to predict the posterior probability of a new data point $\boldsymbol{x}^*$ belonging to a particular class. This is done by formulating a likelihood for our model given the training data and a prior distribution given some knowledge about our data. The likelihood is given by the following expression:

$$P(\boldsymbol{y}|\boldsymbol{w}) = \prod_{n=1}^{N} \sigma\{f(\boldsymbol{x}_n; \boldsymbol{w})\}^{y_n} \; [1 - \sigma\{f(\boldsymbol{x}_n; \boldsymbol{w})\}]^{1-y_n} \tag{42}$$

The likelihood can be used to find the parameters that best fit the data. In order to do so, the expression is maximized given the appropriate values for the parameters and has the same effect as finding the least-squares fit for the data. However, with as many parameters in the model as training points, it could lead to severe overfitting and would probably not provide accurate estimation for new data points.

In this context, imposing a constraint on the parameters would be advantageous so as to force our model to be described by only a portion of the available basis functions resulting in a sparse vector of weights $\boldsymbol{w}$. Thus, a sparse prior probability distribution over $\mathbf{w}$ is necessary. Using the likelihood and the prior, the posterior distribution (ie after observing the training data) is obtained.

Here, the choice of the prior over $\mathbf{w}$ could vary but for the RVM [20], it was chosen as a zero-mean Gaussian prior distribution given by:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{N-1} \mathcal{N}(w_i|0, \alpha_i^{-1}) \tag{43}$$

where $\boldsymbol{\alpha}$ are the hyperparemeters which in turn are described by the following hyperprior:

$$p(\boldsymbol{\alpha}|a, b) = \prod_{i=0}^{N-1} Gamma(\alpha_i|a, b) \tag{44}$$

where,

$$Gamma(\alpha|a, b) = \Gamma(a)^{-1} b^a \alpha^{a-1} e^{-b\alpha} \tag{45}$$

is the Gamma distribution with $\Gamma(a) = \int_0^\infty t^{a-1} e^{-t} \, dt$ the gamma function. To make the hyper-priors flat, the parameters a and b are set to very small values.

Using an approximation procedure [20], the posterior probability distribution $p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\alpha})$ is given by a multivariate Gaussian with covariance and mean:

$$\boldsymbol{\Sigma} = (\boldsymbol{\Phi}^T \boldsymbol{B} \boldsymbol{\Phi} + \boldsymbol{A})^{-1} \tag{46}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{B} \boldsymbol{y} \tag{47}$$

where $\boldsymbol{A} = diag(\alpha_1, \alpha_2, ..., \alpha_N)$ and $\boldsymbol{B} = diag(\beta_1, \beta_2, ..., \beta_N)$ where $\beta_n = \sigma\{f(\boldsymbol{x}_n)\}\,[1 - \sigma\{f(\boldsymbol{x}_n)\}]$

Classification: These can then be used to classify unseen data points using:

$$y_* = \boldsymbol{w}^T \boldsymbol{\phi}(x_*) \tag{48}$$

where $\mathbf{w} = \boldsymbol{\mu}$ and $\boldsymbol{\phi}(x_*)$ contains the value of all included basis functions at the required location. Furthermore, the confidence of the estimate is given by:

$$\sigma_*^2 = \boldsymbol{\phi}(x_*)^T \boldsymbol{\Sigma} \boldsymbol{\phi}(x_*) \tag{49}$$

For large data sets, inversion of large matrices is required. In order to overcome this, a sequential algorithm was proposed [22] [23].

2.10. *Neural Networks*

An alternative to sparse kernel machines is to fix the number of basis functions to be used in the model but allow them to be adaptive in the training phase. The idea behind neural networks [24] is that it utilizes multi-layered logistic regression classifiers with many decision units on each layer to provide a more sophisticated decision boundary.

Recall the model used for classification:

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \, \phi_j(\boldsymbol{x})\right) \tag{50}$$

where $f(.)$ is a nonlinear function that is zero or one depending on the class of the given data point.

Training: In the framework of neural networks, the basis functions $\phi_j(\boldsymbol{x})$ are parametric and their parameters are chosen during training as well as the weights w_j. Each basis function is itself a linear combination of the input data where the coefficients in that linear combination are adaptive parameters.

Thus, we construct M linear combinations of the input variables $x_1, ..., x_D$ where M is the number of basis functions to be used and D is the dimensionality of the input data. These linear combinations are given by:

$$\alpha_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{51}$$

where $j = 1, ..., M$ and the superscript indicates the corresponding layer of the network. All α_j are called the activations of each neuron and are transformed using an activation function $h(.)$ such that:

$$z_j = h(\alpha_j) \tag{52}$$

The choice of $h(.)$ depends on the data but very frequently, the logistic sigmoid function $\sigma(.)$ is used as in logistic regression discussed before. These z_j are the outputs of the hidden units of the first layer and can subsequently be used as inputs to the second layer:

$$\alpha_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{53}$$

where $k = 1, ..., K$.

We can combine all the layers together to acquire the overall network function:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{54}$$

Therefore, the model is a nonlinear function from a set of inputs to a set of outputs controlled by adjustable parameters that are acquired during training. In order to train this network, given a training set $\{(\boldsymbol{x}_1, \boldsymbol{t}_1), ..., (\boldsymbol{x}_N, \boldsymbol{t}_N)\}$ we minimize an error function:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|_2^2 \tag{55}$$

Once the weights are obtained, prediction can be done using the overall network function defined above.

3. Test data sets

All ten algorithms described in the previous section allow the training of classification models. These models are trained given many training samples and are able to automatically decide for the nature of unseen samples.

In order to choose the most suitable for the FAIMS data sets for automatic clinical diagnosis, we first need to test their performance. To do this, we chose five of the techniques that were available in the MATLAB statistical toolbox. These are the LDA, AdaBoost, Random Forests, Naive Bayes and K-NN. For each algorithm, we used four simple data sets to train and test. From these simple data sets, we were able to understand the performance, the advantages and disadvantages of each algorithm and their suitability under different scenaria. Using this preliminary testing, we were able to choose two of the most promising algorithms for their application on three different types of medical FAIMS data sets in section 4.

The first three simple data sets are 2-dimensional with only 2 classes with a linearly separable data set, a non-linearly separable and a data set with a linear class boundary that includes some random noise. An example of the first three type of data sets with the K-NN classifier can be seen in Figures 2, 3 and 4. We also tested the algorithms on a subset of the MNIST dababase [25]. For an example, see Figure 1.

3.1. *Methodology*

First, we split the data sets into training and test sets. The training set is used for the training of the parameters of each classification model. The test set consists of similar but unseen samples and is used for the validation of the performance of the model. We randomly selected 80% of the data for training and used the rest for testing. For three of the algorithms, we had to tweak additional meta-parameters which are not part of the automatic training of the models. These are: the optimal

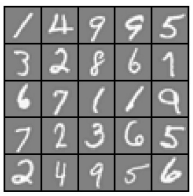

Figure 1: Collection of 25 training data point of the digits data set. The size of each input image is 20×20 pixels and each pixel is a component of our feature vector. The value of a feature measures the intensity of the corresponding pixel.

number of iterations in AdaBoost, the optimal number of trees in Random Forest and the optimal number of neighbours in KNN.

For all these meta-parameters, we trained each model with different values and accordingly recorded the test classification error. The meta-parameter that gave the least classification error was chosen. With different values of the meta-parameters, not only the classification performance but also the training time changes. Therefore, for different application, different meta-parameters might be suitable for different system requirements. In our case, classification performance is priority over training time and thus the meta-parameters were chosen with this in mind.

Once we have determined the optimal meta-parameters, we trained the model on each data set and computed the resubstitution errors, the test errors and the amount of time spent in the training phase and the testing phase. Errors and speeds of the classifiers on each test data sets can be seen in Table 1. Table 2 summarizes the properties of the classifiers that we inferred from our tests.

*3.2. **Discussion***

From these results, the Random Forest and the K-Nearest Neighbours algorithms showed the greatest potential for further investigation. Both have great classification performance on the simple data sets, they are robust to noise and can be trained fast when utilizing parallel computation.

In addition, Random Forest is able to provide importance for specific features which makes it ideal for the understanding of key indicators in medical data sets. This is vital for the further investigation of particular regions in the data sets that would allow for even greater automatic clinical diagnosis.

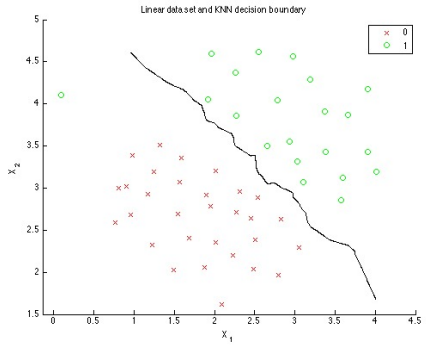

Figure 2: K-NN decision boundary on Linear Data Set

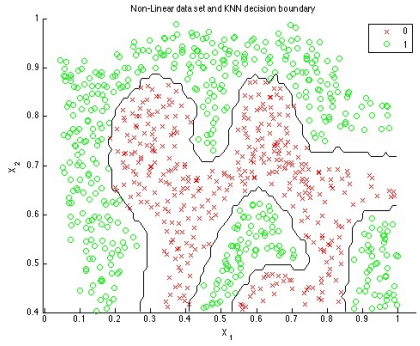

Figure 3: K-NN decision boundary on Non Linear Data Set

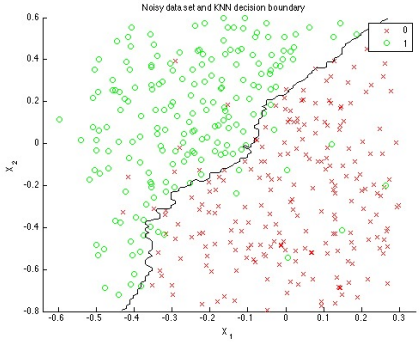

Figure 4: K-NN decision boundary on Noisy Data Set

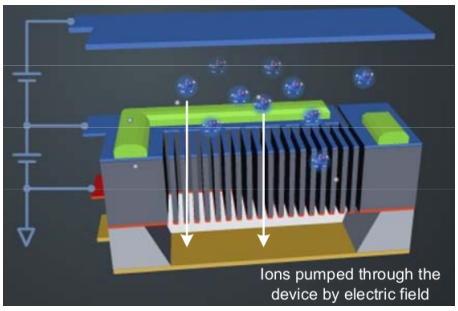

Figure 5: FAIMS setup for ion movement through plates

4. Field Asymmetric Ion Mobility Spectrometry (FAIMS)

As discussed, the detection of airborne gas phase biomarkers that emanate from biological samples like urine and breath can be used for non-invasive diagnostics of IBD diseases such as Crohn's disease and Ulcerative Colitis. An emerging technology that is able to sense and capture these biomarkers is FAIMS [26] [27]. FAIMS is a recent technology that separates gas molecules to be analysed at atmospheric pressure and room temperature [26]. The sample sensed is ionised and thus decomposed of ions of different types and sizes.

These different ions are passed through metal plates that are being applied an asynchronous high voltage waveform. The ionized molecules are subjected to these high electric field and the difference in their movement is used to detect biomarkers [26] [27]. An example of the setup can be seen in Figure 5 from the Owlstone's whitepaper [1]. Further information can be found in [26] [27].

We analysed three separate medical FAIMS data sets supplied by Owlstone. The first data set consisted of biological samples from cancer patients who took part in a study at University Hospital Coventry & Warwickshire. The second data set consisted of biological samples from Inflammatory Bowel Disease (IBD) patients from a separate study, also at the University Hospital Coventry & Warwickshire. Lastly, we analysed a data set with Methanol in a complex background.

The samples were analysed using Owlstone's Lonestar chemical detector which uses Field Asymmetric Ion Mobility Spectroscopy (FAIMS) to create a chemical 'fingerprint' of a gas or odour that is passed through it.

[1] http://www.owlstonenanotech.com/faims

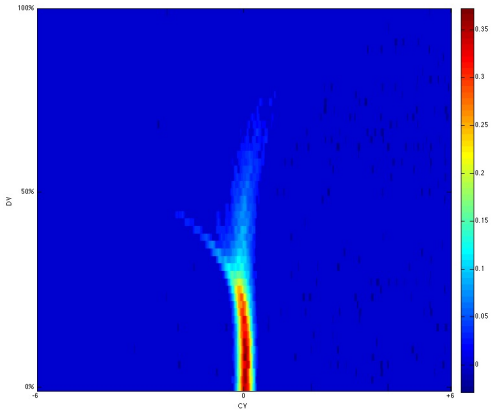

Figure 6: Example of output of chemical fingerprint for a sample from the IBD data using the positive ions.

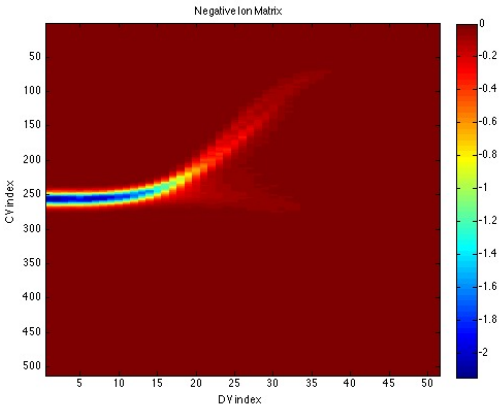

Figure 7: Example of output of chemical fingerprint for a sample from the IBD data using the negative ions.

Our training data consists of these chemical fingerprints. For each sample, we get two such fingerprints - one for the positive ions and one for the negative ions in the gas. See Figures 6, 7 for an example from the IBD Data.

We analysed two versions of this data set. First we analysed the raw data, which consisted of the chemical fingerprints. Each such fingerprint was created by varying the compensation voltage (CV) in 512 steps from -6V to +6V while varying the dispersion voltage ratio (DV) from 0% to 100% in steps of 2% and measuring the resulting ion current. In total we get 52224 raw input features, each giving the intensity of the measured ion current at a particular location (CV & DV value pair).

Secondly, we used software provided by Owlstone

	Linear Data Set[1]				Non-Linear Data Set			
	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)
LDA	0	9.1	0.037	0.426	50.1	45.7	0.046	0.415
AdaBoost	0	18.2	1.132[2]	2.694	8.0	12.1	4.610[3]	2.811
Random Forest	0	0	1.589[5]	13.968	0.1	1.2	6.021[6]	4.248
Naive Bayes	0	9.1	0.046	12.400	14.3	22.0	0.064	1.320
K-Nearest-Neighbours	0	9.1	0.053[7]	0.665	0.3	0	0.071[7]	0.449
	Data Set with Noise				Digits Data Set			
	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)
LDA	7.6	4.8	0.047	0.419	9.2	16.5	0.940	0.855
AdaBoost	6.1	9.6	4.504[3]	3.229	21.4	23.4	6492.0[4]	177.0
Random Forest	1.2	6.0	5.324[6]	5.221	0	7.5	1560.0[6]	5.369
Naive Bayes	7.3	7.2	0.064	1.701	1.6	87.2	31.600	1002.0
K-Nearest-Neighbours	6.1	4.8	0.070[8]	0.645	3.7	5.6	0.226[7]	13.026

(1) 11 data points in Test set, so 9.1% error corresponds to 1 misclassification (which is, in fact, an outlier)
(2) Using 50 iterations.
(3) Using 200 iterations.
(4) Using 4000 iterations.
(5) Using 50 trees.
(6) Using 150 trees.
(7) Using K = 5.
(8) Using K= 15.

Table 1: Errors and speeds of classification methods on test data

	Performance on simple Data Sets	Performance on complex data sets	Robustness to outliers and noise	Speed	Gives confidence level?	Assigns importance to features?
LDA	Good	Poor	Good	Good	No	No
AdaBoost	Moderate	Moderate	Poor	Poor	No	No
Random Forest	Good	Good	Moderate	Moderate*	Yes	Yes
Naive Bayes	Moderate	Poor	Good	Moderate**	Yes	No
KNN	Good	Good	Moderate	Good	No	No

* Allows for parallel computation - great potential speed-up.
** Training is fast - Classifying new data is relatively slow.

Table 2: Summary of classifier performance

to extract peaks in the measured ion current from the chemical fingerprints. For each peak, we were given its location in the fingerprint (CV and DV values) and a measure for its height, width and area. We chose to treat each extracted peak as a separate example, giving all the peaks from one sample the same label.

We focused on two classification methods as they gave the most promising results on the test data sets. We analysed the data using a K-Nearest-Neighbour approach to get some fast preliminary results. Then we moved on to the more sophisticated Random Forests method. In the usual setting, we selected 80% of the samples at random to train the model and used the remaining 20% to test the model. Note that in peak analysis, we extracted the peaks after splitting our data into training and test sets. For all the tests, we aimed to get a reliable measure of the resubstitution error, the test error and the amount of time it took to respectively train and test our model.

4.1. ***Cancer Data***

The cancer data set contained a total of 122 samples, each assigned to one of 3 classes:

- Volunteer $\rightarrow y = 1$ (39 samples)
- Cancer $\rightarrow y = 2$ (70 samples)
- Polyps $\rightarrow y = 3$ (13 samples)

4.1.1. ***Peak Analysis***

We tested a number of different scenarios. First, we ran a number of test on the peaks of the positive ion matrix only. Then we focused on the negative ion peaks. We investigated for the existence of a bias in our results due to the fact that one class is overrepresented in our data.

For that, we made sure that our training set was balanced in that there was an equal number of samples from both classes. It is important to note that we calculated test and resubstitution errors on a per peak basis. As the test errors varied with the choice of our particular test set, we trained and tested our model 50 times, each time using a separate random training and test set, and calculated the mean of the errors. We used the same value for K in each iteration. The results of our KNN analysis of the peaks of the cancer data can be found in Tables 3, 4.

We analysed the peaks from the positive matrix, the negative matrix and the combined data set with the random forest technique. To find a good value for the number of random trees to use, we plotted the test and resubstitution errors as a function of forest size and aimed to choose the point at which the curves began flattening.

We ran the algorithm in a 3-class setting as well as in a 2-class setting for both a random 80%-20% split of the data and a balanced split of the data into training and test set.

One advantage of Random Forests is that it allows measuring the importance of the features. For all features, we measured the increase in prediction error if the values of that feature are permuted across the out-of-bag observations. This measure is computed for every tree, then averaged over the entire ensemble and divided by the standard deviation over the entire ensemble. This measure allows for the identification of key regions in the data set that can be used for further investigation. It allows a more focused investigation on the key indicators. An example of a heat map of the variable importances will be illustrated later that will show its significance in our statistical research and in general for the better training of classification models for automatic clinical diagnostics. Tables 5, 6 show the results of our random forests analysis of the cancer data peaks.

4.1.2. ***Discussion***

From the peak analysis, it can be seen that there is some separation between the classes of interest. Further investigation is required to allow for lower test classification errors. Peak extraction illustrated that we could potentially identify interesting key indicators but a more detailed and focused study is required. Therefore, for the other data sets, we followed a different approach, treating the entire training matrices as one large feature vector.

4.2. ***Inflammatory Bowel Disease Data***

We splitted the IBD data set into two different categories, namely the breath samples and the urine samples. This was done so as to investigate whether different types of data sets are able to distinguish better the classes and allow for a more focused future sample acquisition.

4.2.1. ***Breath Samples***

There were a total 119 breath samples belonging to 3 different classes:

- Volunteer $\rightarrow y = 1$ (42 samples)
- Crohn's Disease $\rightarrow y = 2$ (37 samples)
- Ulcerative Colitis $\rightarrow y = 3$ (40 samples)

Technique	K	Resubstitution Error (%)			Test Error (%)			Training Time (s)	Testing Time (s)
		Mean	*Max*	*Min*	*Mean*	*Max*	*Min*		
3 - class KNN with equal weights	**50**	42.8	47.3	38.4	42.8	***47.3***	38.4	0.034	0.330
3 - class KNN with inverse weights	**50**	0.7	1.2	0	43.5	62.4	28.2	0.041	0.324
3 - class KNN with squared inverse weights	**50**	0.7	1.2	0	45.4	59.4	36.8	0.044	0.323
2 - class KNN with equal weights	**50**	32.3	36.3	28.4	***34.3***	50.0	***17.1***	0.041	0.324
2 - class KNN with inverse weights	**50**	0.7	1.3	0	34.5	54.4	19.4	0.040	0.336
2 - class KNN with squared inverse weights	**100**	0.7	1.2	0	36.0	50.1	27.0	0.043	0.509
2 - class KNN with equal weights and balanced training set	**50**	46.0	48.9	44.3	63.5	72.9	51.3	***0.025***	0.296
2 - class KNN with inverse weights and balanced training set	**50**	0.8	1.8	0	52.0	60.0	44.9	0.032	***0.292***
2 - class KNN with squared inverse weights and balanced training set	**50**	0.6	1.8	0	52.0	60.0	44.9	0.031	0.329

Table 3: KNN results on the positive peaks

Technique	K	Resubstitution Error (%)			Test Error (%)			Training Time (s)	Testing Time (s)
		Mean	*Max*	*Min*	*Mean*	*Max*	*Min*		
3 - class KNN with equal weights	**50**	42.2	46.8	38.1	43.4	60.2	24.2	0.029	0.303
3 - class KNN with inverse weights	**50**	0.7	1.3	0	45.3	61.9	31.8	0.033	0.317
3 - class KNN with squared inverse weights	**150**	0.6	1.3	0	48.0	62.3	35.6	0.039	0.652
2 - class KNN with equal weights	**50**	31.2	36.6	27.6	**30.5**	**46.2**	**8.5**	0.044	0.316
2 - class KNN with inverse weights	**50**	0.8	1.3	0	33.5	57.5	16.4	0.043	**0.300**
2 - class KNN with squared inverse weights	**60**	0.8	1.3	0	35.9	49.0	27.2	0.039	0.382
2 - class KNN with equal weights and balanced training set	**100**	45.3	47.6	40.7	53.4	62.5	39.2	**0.023**	0.489
2 - class KNN with inverse weights and balanced training set	**50**	0.7	1.9	0	48.2	55.7	42.6	0.030	0.338
2 - class KNN with squared inverse weights and balanced training set	**100**	0.4	1.8	0	48.4	52.7	44.4	0.031	0.547

Table 4: KNN results on the negative peaks

KNN: As before, we trained our model with different choices of training and test data sets. As the test errors varied with the choice of our particular test set, we trained and tested our model 50 times, each time using a separate random training and test set, and calculated the mean of the errors. We used the same value for K in each iteration.

KNN results for the breath samples can be found in Table 7. The method had trouble in the 3-class setting in general. Using weights inversely proportional to the distance between the neighbours and the test points gave the best performance with a mean test error of 71.3%.

We got much better results in the 2-class setting. Here, the version with weights proportional to the squared inverse distance gave the best results with a mean accuracy of 64.5%. In our tests, the accuracy varied between 50.0% and 79.2%. Using equal weights for all neighbours gave us an accuracy as high as 83.3% in some cases.

As it can be seen, the resubstitution error is zero in most cases whereas the test error is much larger. This illustrates that there is a great degree of overfitting on the training data. However, the results show promising separation in some cases. If combined with appropriate regularisers, K-NN could allow for a great tool for preliminary classification due to the fact that the training speed is very low.

Random Forests: Due to the large training time of the random forests algorithm and time constraints on our part, we were not able to estimate a mean test error and a measure for the spread of the errors as we did in the KNN analysis for this data set. Thus, one setup was chosen and the results are documented in Table 8.

Using 10 trees gave the best results in terms of accuracy on the test data (62.4%). This is due to the fact that the complexity of the model was the lowest of the ones chosen and allowed for greater generalization. However, we generated a new random split for each test. Therefore, the low test error may be simply an artifact of our choice of training and test data set. The large difference between the test errors and resubstitution errors may indicate that there is a high degree of overfitting and appropriate regularizers could help obtain much better results.

*4.2.2. **Urine Samples***

There were a total of 111 urine samples belonging to 3 different classes:

Technique	Forest Size	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)	Variable Importance				
						DV	*CV*	*area*	*width*	*height*
3-class Random Forest (standard Split)	200	1.9	43.0	68.314	4.957	2.076	**3.292**	2.547	3.089	2.208
2-class Random Forest (standard Split)	200	1.9	***31.9***	56.784	4.168	1.495	**2.522**	1.951	2.057	1.871
2-class Random Forest (balanced Split)	100	***1.0***	45.5	***20.807***	***2.225***	1.491	**2.548**	1.956	2.058	1.877

Table 5: Random Forest results on the positive peaks

Technique	Forest Size	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)	Variable Importance:				
						DV	*CV*	*area*	*width*	*height*
3-class Random Forest (standard Split)	200	2.0	51.7	67.187	4.724	2.023	2.597	2.757	2.432	2.221
2-class Random Forest (standard Split)	200	2.7	36.1	55.028	4.101	1.410	2.083	1.825	1.478	1.704
2-class Random Forest (balanced Split)	100	1.0	47.1	18.987	2.158	0.986	1.383	1.404	1.409	1.152

Table 6: Random Forest results on the negative peaks

Technique	K	Resubstitution Error (%)			Test Error (%)			Training Time (s)	Testing Time (s)
		Mean	*Max*	*Min*	*Mean*	*Max*	*Min*		
3 - class KNN with equal weights	20	59.2	69.5	51.6	75.2	95.8	58.3	0.039	0.830
3 - class KNN with inverse weights	15	0	0	0	71.3	87.5	50.0	***0.018***	0.808
3 - class KNN with squared inverse weights	20	0	0	0	73.7	91.7	58.3	0.022	0.763
2 - class KNN with equal weights	30	35.8	42.1	29.5	***35.5***	58.3	***16.7***	0.025	0.775
2 - class KNN with inverse weights	30	0	0	0	36.7	***50.0***	20.8	0.020	0.773
2 - class KNN with squared inverse weights	30	0	0	0	***35.5***	***50.0***	20.8	0.019	***0.757***

Table 7: KNN IBD Breath Results

Technique	Forest Size	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)
2-class Random Forest	10	10.5	29.1	12697	0.557
2-class Random Forest	50	3.2	50.0	53920	2.245
2-class Random Forest	100	0	37.5	121990	3.988
2-class Random Forest	200	1.1	37.5	228540	7.824

Table 8: Random Forest IBD Breath Results

- Volunteer → $y = 1$ (39 samples)
- Crohn's Disease → $y = 2$ (36 samples)
- Ulcerative Colitis → $y = 3$ (36 samples)

KNN: The KNN results for the urine data set are in Table 9. In the 3-class setting, the algorithm performs much better on the urine data set than on the breath data. This may indicate that there is a smaller degree of noise in the data and the distinction between the Crohn's Disease samples and the Ulcerative Colitis may be clearer compared to the breath samples. The performance in the 2-class setting is very similar to that in the breath data. We were able to achieve a test error as low as 18.2%.

Random Forests: The Random Forests results can be found in Table 10. In our tests in the 2-class setting, the algorithm appears to perform slightly worse than on the breath samples. However, this difference may again be because of the particular test set. We see the same large difference between test errors and resubstitution errors as in the breath samples, indicating that overfitting may be problem in this application as well.

4.2.3. *Heat Map*

In order to get some visual indicator of regions in the fingerprint with high discriminative power, we produced a heat map of variable importance. We calculated the measure of feature importance provided by the Random Forest algorithm for each CV-DV value pair. We plotted these values to find regions of interests, with red indicating the most important and blue indicating the least important regions.

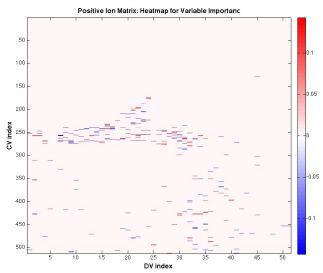

Figure 8: Heatmap for positive ion fingerprint of a sample from the IBD data.

Figures 8 and 9 show two heatmaps for a sample from the IBD data set, one for the positive ion matrix and one for the negative ion matrix. These were created using 100 trees for a random forest.

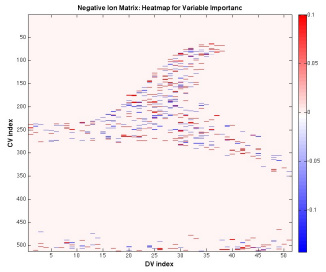

Figure 9: Heatmap for negative ion fingerprint of a sample from the IBD data.

The negative ion matrix appears to carry more information with regards to feature importance as there more areas with high density of red points.

4.3. *Methanol in Complex Background*

The methanol samples were taken under strict laboratory conditions. We received 42 training examples and 12 test examples. We analysed the performance of the KNN and Random Forests algorithms in a 3-class setting and in a 2-class setting.

The samples were classified as

- Diseased → $y = 0$
- Exposed → $y = 1$
- Normal → $y = 2$

For the 2-class analysis, we merged the "Diseased" class and the "Exposed" class. We had a total of 18 samples from each class. The settings of the Lonestar chemical detector were changed for the methanol data: The number of lines in the fingerprint was reduced from 51 to 15. Unlike before, we used a particular choice for our training and test data sets.

KNN: The results of the KNN analysis of the methanol data can be found in Table 11. The algorithm performed very well in a 3-class setting, achieving a test error of 25.0% on the provided test data. Setting the weights of the K neighbours proportional to the inverse of the distance to the test points lead to very low errors when reclassifying the training data (2.4%).

In a 2-class setting, the algorithm achieves a perfect prediction on the test data, finding the correct class of each test example. Using inverse weights also leads to a very low resubstitution error.

Technique	K	Resubstitution Error (%)			Test Error (%)			Training Time (s)	Testing Time (s)
		Mean	*Max*	*Min*	*Mean*	*Max*	*Min*		
3 - class KNN with equal weights	15	46.7	53.9	40.4	58.5	77.3	40.9	***0.017***	0.785
3 - class KNN with inverse weights	15	0	0	0	55.9	72.7	31.8	0.018	0.691
3 - class KNN with squared inverse weights	15	0	0	0	53.6	68.2	27.3	0.027	0.691
2 - class KNN with equal weights	30	32.7	43.8	25.8	39.4	63.6	***13.6***	0.024	0.703
2 - class KNN with inverse weights	30	0	0	0	38.7	***59.1***	22.7	0.019	0.704
2 - class KNN with squared inverse weights	30	0	0	0	***37.6***	***59.1***	18.2	0.018	***0.675***

Table 9: KNN IBD urine results

Technique	Forest Size	Resubstitution Error (%)	Test Error (%)	Training Time (s)	Testing Time (s)
2-class Random Forest	10	4.5	45.5	9475	0.377
2-class Random Forest	50	2.3	54.6	45173	1.325
2-class Random Forest	100	0	50.0	93466	3.344

Table 10: Random Forest IBD urine results

Technique	Value of K	Resubstitution Error (%)	Test Error (%)
3-class, equal weights	6	31.0	25.0
3-class, inverse weights	5	2.4	25.0
3-class, squared inverse weights	5	2.4	25.0
2-class, equal weights	6	16.7	0
2-class, inverse weights	5	2.4	0
2-class, squared inverse weights	5	2.4	0

Table 11: KNN Methanol in Complex Backround results

Technique	Forest Size	Resubstitution Error (%)	Test Error (%)
3-class Random Forest	10	9.5	41.7
3-class Random Forest	50	4.8	25.0
3-class Random Forest	100	2.4	41.7
2-class Random Forest	10	4.8	16.7
2-class Random Forest	50	2.4	16.7
2-class Random Forest	100	2.4	16.7

Table 12: Random Forest Methanol in Complex Backround results

Random Forests: The results of the Random Forests analysis can be found in Table 12. In our tests, the random forests algorithm showed a strong performance. It managed to correctly classify 75.0% of the test data in the 3-class setting, with a resubstitution error of 4.8%. In a 2-class setting, it correctly predicted 83.7% of the test examples. We achieved the best performance with 50 random trees in our forests.

5. Future Work

The Random Forest experiments generally had very low resubstitution error while the test error was higher than expected. There is a large amount of overfitting and we suggest to address this issue by using a pruning algorithm or regularization.

We kept many of the hyperparameters of the random forest algorithm constant. Optimization of all the hyperparameters can lead to substantial performance improvements. In most tests, we only utilized 80% of the available data to train a classification model. Given more time, we would try a leave-one-out analysis and cross-validation. More training data in general should give better results.

The KNN algorithm gave promising results, given that the underlying method is so simple. Furthermore, the algorithm is computationally inexpensive. Therefore, an AdaBoost model with KNN classifiers as weak learners may be a promising future approach.

Due to time restrictions, we were not able to test and apply a number of the classification methods described in Section 2. In particular, Neural Networks is a very powerful method for classification of complex patterns.

In the peak analysis, we treated each peak as an independent sample. In order to classify a new sample, one would have to look at the distribution of class predictions of all the peaks in the sample. We ran some rough preliminary tests and noticed that the distribution was highly skewed towards predicting cancer. This may be due to the greater presence of cancer samples in the data set. It may be worth to investigate this further.

We also did not attempt to extract the peaks from the IBD and methanol data sets. While we have no particular reason to believe so, it may be possible to find good separation of the classes in the peaks domain. The heat maps from the IBD data showed that the negative ion matrix may be more informative for classification purposes than the positive one. It would be interesting to run tests with the negative ion matrix for the methanol data.

In previous experiments, wavelets were used to preprocess data before classification [2]. This led to some good results and combined with an algorithm such as the random forest, may improve the method's accuracy.

Another promising approach may be to treat the Lonestar fingerprints as images and use an edge detection algorithm. We would scan each sample to detect edges in the fingerprint. Next we would plot a histogram of the orientation of the edges. Finally, we would attempt classification of the samples using the histograms as feature vectors that might lead to discriminative areas.

6. Conclusion

In this paper, we provided an overview of ten different classification algorithms along with their potential advantages and disadvantages. We described the intuition behind their theory and illustrated how they can be used for different data sets. Next we selected five of these methods and performed tests on a number of artificially generated data and also one real-world data set. Our aim was to investigate how these algorithms perform in practice.

From these five algorithms, we chose two (K Nearest Neighbours and Random Forests) that performed best on the simple data sets and used them to analyse various data sets from Field Asymmetric Ion Mobility Spectrometry. In particular, we investigated 2 different studies (Colorectal Cancer, Inflammatory Bowel Disease). Our aim was to find a reliable way to separate the classes of patients within a study using FAIMS. We attempted to separate all three classes as well as just patients from volunteers. Using K Nearest Neighbours and Random Forests, we achieved high prediction accuracy in a number of cases.

We attempted to find regions that provide key indicators between the classes on the chemical fingerprint produced by FAIMS. We did this by creating heat maps using a measure for feature importance from the Random Forests model.

In addition, we studied samples taken from methanol in oil that were produced under laboratory conditions. These samples contained less contamination and noise and were able to give us some further indication on the performance of the classification methods. We achieved perfect predictions using K Nearest Neighbours in some cases. Random forests gave also very promising results.

However, we repeatedly encountered a large difference between resubstitution errors and test errors with this algorithm. This indicates that further optimization of the random forests parameters may lead to substantial performance improvements.

Acknowledgements

The study was funded by Owlstone. Data for the Methanol in complex background was collected by Rebecca O'Donnell and Dr Max Allsworth under Owlstone's LuCID (Lung Cancer Indicator Detection) project funded by an SBRI Healthcare development contract. Data for the IBD was collected in a joint project with Owlstone, Warwick University, and University Hospital Coventry and Warwick, utilizing funding from the Technology Strategy Board (TSB).

References

[1] V. L. Patel, E. H. Shortliffe, M. Stefanelli, P. Szolovits, M. R. Berthold, R. Bellazzi, A. Abu-Hanna, The coming of age of artificial intelligence in medicine, Artificial Intelligence in Medicine 46 (1) (2009) 5–17.

[2] R. Arasaradnam, N. O. Ouaret, M. Thomas, E. Hetherington, M. N. Quraishi, C. Nwokolo, K. D. Bardhan, J. Covington, Identification of inflammatory bowel disease (ibd) using field asymmetric ion mobility spectrometry (faims), Gut 61 (Suppl 2) (2012) A70.

[3] J. A. Covington, L. Wedlake, J. Andreyev, N. Ouaret, M. G. Thomas, C. U. Nwokolo, K. D. Bardhan, R. P. Arasaradnam, The detection of patients at risk of gastrointestinal toxicity during pelvic radiotherapy by electronic nose and faims: A pilot study, Sensors 12 (10) (2012) 13002–13018.

[4] J. A. Covington, E. W. Westenbrink, N. Ouaret, R. Harbord, C. Bailey, N. Connell, J. Cullis, N. Williams, C. U. Nwokolo, K. D. Bardhan, R. P. Arasaradnam, Application of a novel tool for diagnosing bile acid diarrhoea, Sensors 13 (9) (2013) 11899–11912.

[5] R. P. Arasaradnam, E. Westenbrink, M. J. McFarlane, R. Harbord, S. Chambers, N. OConnell, C. Bailey, C. U. Nwokolo, K. D. Bardhan, R. Savage, J. A. Covington, Differentiating coeliac disease from irritable bowel syndrome by urinary volatile organic compound analysis a pilot study, PLoS ONE 9 (2014) e107312.

[6] R. P. Arasaradnam, M. J. McFarlane, C. Ryan-Fisher, E. Westenbrink, P. Hodges, M. G. Thomas, S. Chambers, N. O'Connell, C. Bailey, C. Harmston, C. U. Nwokolo, K. D. Bardhan, J. A. Covington, Detection of colorectal cancer (crc) by urinary volatile organic compound analysis, PLoS ONE 9 (2014) e108750.

[7] R. A. FISHER, The use of multiple measurements in taxonomic problems, Annals of Eugenics 7 (2) (1936) 179–188.

[8] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] K. Fukunaga, Introduction to statistical pattern recognition, Academic press, 1990.

[10] B. Scholkopft, K.-R. Mullert, Fisher discriminant analysis with kernels, Neural networks for signal processing IX.

[11] T. Hofmann, B. Schölkopf, A. J. Smola, Kernel methods in machine learning, Annals of Statistics 36.

[12] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: Proceedings of the Second European Conference on Computational Learning Theory, Springer-Verlag, London, UK, UK, 1995, pp. 23–37.

[13] Y. Freund, R. E. Schapire, Experiments with a new boosting algorithm (1996).

[14] L. Breiman, E. Schapire, Random forests, in: Machine Learning, 2001, pp. 5–32.

[15] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning., Tech. Rep. MSR-TR-2011-114, Microsoft Research (Oct 2011).

[16] R. O. Duda, P. E. Hart, Pattern Classification and Scene Analysis, John Willey & Sons, 1973.

[17] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers (1997).

[18] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theor. 13 (1) (2006) 21–27.

[19] B. E. Boser, I. M. Guyon, V. N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press, 1992, pp. 144–152.

[20] M. E. Tipping, Sparse bayesian learning and the relevance vector machine, J. Mach. Learn. Res.

[21] M. E. Tipping, The relevance vector machine, in: Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999], 1999, pp. 652–658.

[22] A. C. Faul, M. E. Tipping, Analysis of sparse bayesian learning, in: Advances in Neural Information Processing Systems 14, MIT Press, 2001, pp. 383–389.

[23] M. E. Tipping, A. Faul, Fast marginal likelihood maximisation for sparse bayesian models, in: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, 2003, pp. 3–6.

[24] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Inc., New York, NY, USA, 1995.

[25] http://yann.lecun.com/exdb/mnist/.

[26] A. Wilks, M. Hart, A. Koehl, J. Somerville, B. Boyle, D. Ruiz-Alonso, Characterization of a miniature, ultra-high-field, ion mobility spectrometer, International Journal for Ion Mobility Spectrometry 15 (3) (2012) 199–222.

[27] A. A. Shvartsburg, R. D. Smith, A. Wilks, A. Koehl, D. Ruiz-Alonso, B. Boyle, Ultrafast Differential Ion Mobility Spectrometry at Extreme Electric Fields in Multichannel Microchips, Analytical Chemistry 81 (2009) 6489–6495.