Level Set Clustering in a Flow Cytometry Dataset

Master's Thesis
Sauli Herrala
2126665
Department of Mathematics
The University of Oulu

Contents

1 Introduction

Flow cytometers are high throughput machines that quantify the measurement of light scatter and fluorescence emission spectra of individual cells within each analyzed sample. Flow cytometry (FCM) is used routinely in medical research and diagnosis of health disorders. For example in diagnosis blood cancers and HIV infection. Within the flow cytometer, cells are suspended in a stream of fluid which is passed through a laser beam, after which the emitted light is measured. The cells are typically stained with fluorochrome-conjugated antibodies that selectively bind to a specific region or a functional group. Current flow cytometers can analyze over 20 different characteristics of cells [1, 2].

After the analysis the individual cells are clustered into populations based on similarities in measurements. This is usually done by visual inspection of one or two dimensional plots at a time, and manually gating regions of interest. Not only is this process time consuming with high dimensional data, it is also subjective [3].

Because of this FCM is moving towards automated methods [4, 5, 6, 7, 8, 9]. The purpose is to find an automated processing pipeline which automatically gates and matches cell populations across samples. However, accurate population identification is complicated because of asymmetric and overlapping nature of cell populations, and because of outliers due to measurement error. FlowCAP challenges aim to address these issues by comparing multiple solutions across a wide variety of data [10].

In this study it is evaluated how level set tree clustering (LSTC) by using leafsfirst algorithm, and k-means clustering works on the acute myeloid leukemia dataset from the FlowCAP 2 challenge. First the cell population clusters are found with LSTC or k-means and then based on those clusters a support vector machine (SVM) is trained to discriminate healthy and acute myeloid leukemia (AML) cell populations. In LSTC we estimate the multivariate density of a function f and the clusters are chosen on the basis of the modes of that multivariate density. This is computationally a fairly expensive process. The worst case time complexity is $O(dn^2)$, where d is the dimension of the data and n the amount of observations. Because the amount of observations is the dominant term this study will use local outlier factor to remove outliers.

This study has three main objectives. First, to handle computational issues we improve the current LST algorithm by introducing an Nelder-Mead algorithm optimization algorithm in one of the steps. After that this study will look at how well local outlier factor performs. Key questions being: how much of the data is removed as outliers, and do the results get any better or

worse. This evaluation is only done using k-means clustering because LST is computationally more intensive. Finally, the performance of LST is estimated and compared against k-means.

First, some technical background information on how flow cytometry works is given. After that this study will look at few common clustering and classification algorithms that were used in FlowCAP 2 challenge. They are not the exact algorithms, but rather the underlying theory behind what was used. Finally, the theoretical background of LSTC is given.

2 Background

Flow cytometry measures multiple characteristics of particles in a fluid as they move past a beam of light. For example some measurable characteristics are relative size, granularity and fluorescence intensity of the particle. These are determined based on how light scatters and emits fluorescence when the particle goes pass the light source. Forward scatter (FS) correlates with the particles size and side scatter (SS) depends on the density of the particle. Fluorescence can be measured by a stained (marked) particles that will emit light when a certain wavelength of light is used.

The modern cytometers have three main components

- a flow cell, which is a liquid stream that separates the individual particles so that they pass the laser in a single file.
- a measuring system, which consists of laser(s), detector(s) and the electronic system which converts the detected light signals to electronic signals.
- a computer which analyses the signals.

The raw flow cytometry data undergoes multiple pre-processing steps to improve the quality of the data. In situations where there are multiple samples and cytometers doing analyses the between-sample variation poses significant challenge. This variation arises from technical reasons rather than biological ones. Normalization step aims to make the samples more comparable in further analyses. This is achieved by computing and removing the population shifts from either multi-dimensional data or per channel. Common solution for normalization is to look at one or two dimensions and shift the populations by hand, but this can also be done by standard mathematical normalization techniques such as scaling or by landmark registration [11].

Second standard pre-processing step is data transformation. Cell populations are often described as having log-normal expression. So a traditional log

transformation is commonly used in pre- or post-processing of the data. However this usually does not work for normalized data as there can be negative values. In this case the usual procedure is to treat the data as positive and taking account the sign after the transformations. This is done for example in logicle [12], and the Box-Cox transformation [5].

Third step, compensation, is needed because the emission spectra of the fluorochromes overlap. A narrow band-pass optical filter is used at the fluorochromes peak intensity to reduce overlapping, but the result still gives the reading of multiple fluorochromes that overlap in that particular spectrum. This overlap is called spillover, and removing it from the data is called compensation [13]. Removal of spillover is accomplished by running a series of calibration samples each stained for only one fluorochrome. This gives a baseline measurement for each of the fluorochromes per channel. The baseline measurements can be represented as a system of linear equations which is called the spillover matrix. The compensated data is generated by inverting the spillover matrix and multiplying it with the original data.

Fourth step is quality control which aims to detect technical errors in individual samples. This can be done manually by looking at visualizations and summary statistics to determine if some samples deviate from the norm. It is also possible to use automated outlier detection methods such as Grubb's test [14].

3 Data pre-processing

3.1 Logicle transformation

Flow cytometry data is usually described as having log-normal distributions. Because of this log transformation is widely used, but compensated data can include populations with negative values which logarithm cannot handle. The logicle transformation handles this problem through a biexponential formula which is a generalization of hyperbolic functions [12, 32]. The biexponential formula is:

$$S(x; a, b, c, d, e, f) = ae^{bx} - ce^{dx} + f. \tag{1}$$

The parameters are adjusted in such a way that the transformation is near-linear close to zero and near-logarithmic behavior at high values. Linearity around zero can be achieved by choosing functions for which $S(0) = 0$, and $\frac{\partial^2 S(x)}{\partial x^2}\big|_{x=0} = 0$. The functions with this property are the logicle scaling functions.

To fix all five parameters (a, b, c, d, f) we make four choices that reflect our displayed scale. First, we choose T to be maximum data value in the displayed scale. The second parameter M is for the range of the display in decades in either base 10 logarithm or the natural logarithm. If this is held constant among datasets the near-logarithmic behavior at high values is similar. So only the fine tuning of region near zero is needed. Which leads to our third parameter w that determines the strength of linearization near zero. If this parameter is too small the transition to logarithmic behavior starts too early and the compensation artifacts will still be visible. The fourth choice is to handle negative values in the display that might come from compensation. This value needs to be high enough so that truncation of the near-linear region does not happen. The third and fourth choice is expressed as a single value $w = 2p\log(p)/(p+1)$, where w is the width of the negative values. This is done so that the lowest negative values correspond to the approximate edge of the linearised zone. Also it is not necessary to choose w to be the minimum value of x.

The logicle transformation with these constrains in log scale is

$$S(x; T, M, w, p) = Te^{-(M-w)}(e^{x-w} - p^2e^{-(x-w)/p} + p^2 - 1), \tag{2}$$

where $x \geq w$. For a 4 decade data the maximum value is $T = 10000$ and the breath of display is $M = 4\log(10) = 9.21034$. When $x < w$ we enforce symmetry by calculating $\text{sign}(x)S((w - x); w)$.

3.2 Local Outlier Factor

Local outlier factor [38] is used for finding anomalous observations in the data. This is done by looking at local densities with k-nearest neighbors and finding points that have lower density than their neighbors. The formal definition consists of five parts. First we define $\text{distance}_k(p)$ as the distance where point p has at least k-neighbors, and $N_k(p)$ as the set of those points. Given observations $\{x_1, \ldots, x_n\}$ and ball $B_r(p) = \{z \in \mathbb{R}^d \,|\, d(z, p) < r\}$, where function d is a chosen distance metric, the two definitions are now:

$$\text{distance}_k(p) = \inf\{r > 0 \,|\, \#\{x_i \in B_r(p)\} \geq k\}, \tag{3}$$

$$N_k(p) = \{x_i \in B_r(p) \,|\, r = \text{distance}_k(p)\}, \tag{4}$$

In $\text{distance}_k(p)$ we choose more then k points if there are ties in the data. In 2-dimensional space $\text{distance}_k(p)$ can be thought as the radius of a circle which encloses at least k points around point p. The reachability distance of point p with respect to point o is:

$$\text{reachability-distance}_k(p, o) = \max\{\text{distance}_k(o), d(p, o)\} \tag{5}$$

Where $d()$ is the same distance metric that was used in distance_k. Reachability distance is the actual distance between points p and o if point p is not in the k-neighborhood of o. In a sense k acts as a smoothing parameter as higher values give more similarity between multiple reachability distances.

Now local reachability density is defined by:

$$\text{lrd}(p) = \frac{\#N_k(p)}{\sum_{o \in N_k(p)} \text{reachability-distance}_k(p, o)} \tag{6}$$

This is an inverse average of the possible reachability-distances in point p neighborhood.

Finally local outlier factor is defined by:

$$\text{LOF}(p) = \frac{\sum_{o \in N_k(p)} \frac{\text{lrd}(o)}{\text{lrd}(p)}}{\#N_k(p)} \tag{7}$$

Which is an average of the local reachability densities for p in its neighborhood $N_k(p)$. In their article Breunig et al prove that for dense regions $\text{LOF} \leq 1 + \epsilon$ and hence outlier are the points with $\text{LOF} > 1 + \epsilon$.

However implementing this algorithm isn't straightforward as we have to define the k-neighborhood for all the points. A thorough linear search would take $O(n^d)$, where d is the dimension of the data. Using approximate methods such as kd-trees [39] improves performance if $n >> d$.

4 Clustering

Raw cytometry data is very complex as there can be multiple tube and marker combinations of thousands to millions of cells that makes up a single analysis. That's why an important step in cytometry data-analysis is complexity reduction through identification of important cell subpopulations. The standard methods to do this are sequential gating, and cluster analysis.

Sequential gating involves histogram or scatter plot visualizations as cell population identification tools [3]. The idea is to draw a clear cut off point in the histogram or "gate" an area in the scatter plot to identify populations for further analysis. In multivariate data it is also possible to color the data based on fluorochrome intensity. Manual gating works well in low dimensional data and with clear populations. But sequential gating can still be subjective and not practical in high dimensional analysis as it requires manual work. It is possible to use dimension reduction methods such as principal component

analysis to ease identification, but it doesn't remove the underlying issue of manual work [16]. Sequential gating is still the main solution for population identification, but as dimensionality grows it becomes less feasible.

Clustering involves binning of the data or a clustering algorithm. In binning the data is partitioned into bins in a fixed manner, for example number of cells over a certain threshold or within a certain boundary. Clustering has the same idea, but the regions are defined based on the algorithm's result. Aim of the algorithm is to cluster observations so that they are more similar to each other than to those in other clusters. Elaborating two of the common methods will be the focus of this section. More detailed description of applied methods by each FlowCAP participant can be found in the supplement [10]. There is also a recent survey of common techniques used in flow cytometry by Bashashati and Brinkman [18].

In this study we present two standard clustering algorithms: k-means, and mixture models. They are presented in their classical form, but their variants for flow cytometry data can be found from Bioconductor packages flowMeans [4], and flowClust [5]. But there are also others full package solutions such as FLAME [9].

4.1 k-means clustering

k-means clustering aims to partition data into k clusters by finding the cluster centers that minimize the total within-cluster sum of squares. Given observations $\{x_1, \ldots, x_n\}$ where each x_i is a d-dimensional vector, k-means aims to partition the n observations into sets $C = \{c_1, \ldots, c_k\}$ by minimizing the following:

$$\underset{C}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|^2, \tag{8}$$

where μ_i is the cluster center of points in c_i. So for each observation x_i the chosen cluster is the closest cluster center. This is the classical definition using the Euclidean distance. More general formulation is

$$\underset{C}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in c_i} D(x, \mu_i), \tag{9}$$

where D is a distance function between the cluster center and the observation. For example Mahalanobis distance [19] is a possibility in flow cytometry [4].

There are four main algorithms to solve k-means clustering minimization objective [20, 21, 22, 23].

The practical problem with k-means is that the number of clusters has to be predetermined, and it is more adapt in finding spherical shaped clusters. The number of clusters can be arbitrarily chosen or one could look through a range of options which might provide the best end result, but there are ways to automate this process for example with Bayes information criterion (BIC) [24] or with normality test [25]. But implementing these ideas might be computationally expensive if the algorithm doesn't settle to a small number of clusters. Also as k-means is forced to find clusters there's a boundary problem in densly packed observation spaces, meaning that near the cluster boundaries there are observations which are relatively close to at least two cluster centers. This might not be such a big problem, but in some cases a fuzzy k-means algorithm [26] should be considered as it allows observations to be in multiple clusters.

4.2 Mixture models

In mixture models, or model-based clustering the clusters are described by separate probability distributions. This is done by estimating the mixture distribution that represents the whole dataset. Given observations $\{x_1, \ldots, x_n\}$ where each x_i is a d-dimensional vector, the likelihood for a mixture model with $C = \{c_1, \ldots, c_k\}$ components (clusters) is

$$L(\theta_1, \ldots, \theta_k; w_1, \ldots, w_k \mid x_1, \ldots, x_n) = \prod_{i=1}^{n} \sum_{c=1}^{k} w_c f_c(x_i \mid \theta_c) \qquad (10)$$

where f_c and θ_c are the density function and parameters of the cth component in the mixture and w_c is the prior probability that an observation belongs to the cth component ($w_c \geq 0; \sum_{i=1}^{k} w_i = 1$). Estimates for the unknown parameters θ_c can be obtained by using the Expectation-Maximization algorithm. Similarly to k-means clustering the amount of clusters can decided based on BIC.

As the cluster status is based on the probability of belonging to a certain distribution from this mixture, it is possible to use mixture models in outlier detection. This is done by choosing a threshold probability for cluster assignment. For example for an observation to be clustered it has to have at least 0.1 chance to be in one of the distributions in the mixture.

The conventional density function is the multivariate normal (Gaussian) density, but other distributions are also viable [27]. Multivariate normal density ϕ_c is parametrised by its mean μ_c and covariance matrix Σ_c,

$$\phi_c(x_i|\mu_c, \Sigma_c) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_c)}} \exp\left\{-\frac{1}{2}(x_i - \mu_c)^T \Sigma_c^{-1}(x_i - \mu_c)\right\} \quad (11)$$

Data from mixture of normal densities leads to ellipsoidal groups of points with increased density near the means μ_c. Covariance matrix Σ_c defines the geometric features of the individual clusters, and by using eigenvalue decomposition we get three separate matrices [29]:

$$\Sigma_c = D_c \Lambda_c D_c^T = \lambda_c D_c A_c D_c^T, \quad (12)$$

where D_c is the orthogonal matrix of eigenvectors and matrix Λ_c has the eigenvalues on the diagonal. Furthermore, we split the eigenvalue matrix Λ_c by writing $\Lambda_c = \lambda_c A_c$ whereλ_c is the first eigenvalue and $A_c = \text{diag}(a_{1c}, \ldots, a_{dc})$, and $1 = a_{1c}, \geq \cdots \geq a_{dc} > 0$. Now λ_c defines volume, D_c orientation and A_c shape of a given density ϕ_c. By considering some or all of these parameters as fixed values the clusters will share geometric properties and ease estimation as there are fewer parameters to estimate.

4.3 Level set clustering

4.3.1 Level set

Suppose we have observations $\{x_1, \ldots, x_n\}$ where each x_i is a d-dimensional vector, which are i.i.d from an probability distribution with density function f. Our goal is to identify clusters based on the modes of f following the principle of Hartigan [41]. For any $\lambda \geq 0$ a level set is defined as follows:

$$\Lambda(f, \lambda) = \{x \in \mathbb{R} \mid f(x) \geq \lambda\} \quad (13)$$

A level set tree of a function $f : \mathbb{R} \to \mathbb{R}$, associated with set of levels $\mathcal{L} = \{\lambda_1 < \cdots < \lambda_L\}$, where $\lambda_L \leq \sup_{x \in \mathbb{R}^d} f(x)$, is a tree whose nodes are associated with subsets of $\mathbb{R}^d$ and levels $\mathcal{L}$ in the following way [46]:

1. Write the lowest set as

$$\Lambda(f, \lambda_1) = A_1 \cup \cdots \cup A_K \quad (14)$$

where sets A_i are pairwise separated, and each is connected. The level set tree now has K roots at level λ_1 each root is associated with a set A_i, $i = 1, \ldots, K$.

2. Let node m be associated with set $B \subset \mathbb{R}^d$ and level $\lambda_l \in \mathcal{L}$, $l = 1, \ldots, L$.

   (a) If $B \cap \Lambda(f, \lambda_{l+1}) = \emptyset$, then node m is a leaf node.

   (b) Otherwise, write

$$B \cap \Lambda(f, \lambda_{l+1}) = C_1 \cup \cdots \cup C_M \tag{15}$$

   Similarly as in step 1, now sets C_i are pairwise separated, and each is connected. Now node m has M children (or roots) at level λ_{l+1} each child is associated with a set C_i, where $i = 1, \ldots, M$.

4.3.2 Leafsfirst algorithm for level set trees and clustering

We assume that there is a collection of elementary sets $A_1, \ldots, A_L \subset \mathbb{R}^d$ so that all level sets are unions of these elementary sets in a growing order. The lowest level set is a union of all the upper level sets, and the ith level is only a union from ith to the last level set. Formally for level set $\Lambda(f, \lambda_l)$, where $l = 1, \ldots, L$

$$\Lambda(f, \lambda_l) = \cup_{j=l}^{L} A_j \tag{16}$$

If the elementary sets do not have this property then the function f has to be approximated in such a way that the assumption holds.

An algorithm for level set trees by Klemelä is called the Leafsfirst algorithm [46].

1. The input for the algorithm are the levels $\lambda_1 < \cdots < \lambda_L$ and the elementary sets $A_1, \ldots, A_L$

2. Let the first leaf node be associated with the highest level λ_L and the elementary set A_L. This node is now a "temporary root node", because the node does not yet have a parent node.

3. For $l = L - 1$ to 1: Consider level λ_l and set A_l.

   Create new node with level λ_l. We have to find the associated sets to this node and we have to find the child nodes of the new node. We find which sets associated with the temporary root nodes (those nodes which do not yet have a parent) touch set A_l. At this step we will apply the bounding box technique, which is explained below. We have two possible cases

(a) If set A_l touches sets $B_1, \ldots, B_M$ associated with the temporary root nodes, then create the connection between A_l and $B_1, \ldots, B_M$; the new node is the parent of the nodes that are touched. The set associated with this new parent node is $A_l \cup B_1 \cup \cdots \cup B_M$.

(b) If set A_l does not touch any sets associated with the temporary root nodes, then the new node is a leaf node of the tree and does not have children. The set associated with this node is A_l.

The new node is a new "temporary root node".

4. When we have gone through all the levels, the remaining "temporary root nodes" are the final root nodes.

In the bounding box technique used at step 3 we find the sets that touch the current root nodes. The bounding box of a set A is the smallest rectangle containing A, such that the sides of the rectangle are parallel to the coordinate axes. In step 3 we find those sets which are associated with the current root nodes and are touched by set A_l. Only if A_l touches the bounding box will we progress towards the other leaf nodes to see if A_l touches any of the smaller bounding boxes. If A_l does not touch the bounding box of those sets, then it does not touch any sets inside the bounding box.

Level set tree approximation by rectangle partitioning , Let $f_0 : \mathbb{R}^d \to \mathbb{R}$ be an initial function whose level sets we want to calculate. We assume that there is no collection of sets $A_1, \ldots, A_l \subset \mathbb{R}^d$. We will use partitioning to construct an approximation of sets for which this assumption holds.

First we choose a rectangle $R \subset \mathbb{R}^d$ containing the regions where we approximate f_0. Partitioning of R consists of rectangles $A = \{A_1, \ldots, A_L\}$. This means that $\cup_{l=1}^{L} A_l = R$ and $A_n \cap A_m = \emptyset$ for $n \neq m$. Let c_i be the center of A_i, $i = 1, \ldots, l$. The function f_0 is evaluated at points c_l and we define

$$f(x) = \sum_{i=1}^{l} f_0(c_i) I_{A_i}(x) \tag{17}$$

Now our initial assumption holds as f is a collection of sets when we choose $\lambda_i = f_0(c_i)$. We also assume, without loss of generality, that $f_0(c_1) < \cdots < f_0(c_l)$. It is possible to do this for any type of partitioning, but rectangles are convenient because it is easy to check if two rectangles touch.

5 Support vector machines

Support vector machines (SVM) [30, 31] are supervised learning models that are based on maximum-margin hyperplanes.

Consider observations and labels $(x_1, y_1), \ldots, (x_n, y_n)$, where $x \in \mathbb{R}^d$ and $y \in \{-1, 1\}$. The training is done by solving the following primal optimization problem:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \leq 1 - \xi_i \\ & x_i \leq 0, i = 1, \ldots, n, \end{aligned} \tag{18}$$

where $\phi(x_i)$ is a function that maps x_i to a higher-dimensional space, and $C > 0$ is the penalty parameter. This form is also called the C-support vector classification. However, in practice we solve the dual problem, because it is more efficient in high dimensionality settings. From Karush-Khun-Tucker conditions we can express the solution as:

$$w = \sum_{i=1}^{n} y_i \alpha_i \phi(x_i) \tag{19}$$

By substituting the dual problem is:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, n \\ & y^T \alpha = 0 \end{aligned} \tag{20}$$

where $e = [1, \ldots, 1]^T$ is the vector of all ones, $C > 0$ is the upper bound for all α_i, and Q is $n \times n$ positive semidefinite with $Q_{ij} = y_i y_j K(x_i, x_j)$, where $K = \phi(x_i)^T \phi(x_j)$ is the kernel function. For future reference we will denote $\frac{1}{2} \alpha^T Q \alpha - e^T \alpha$ with $f(\alpha)$. The decision function is

$$\text{sign}(w^t \phi(x) + b) = \text{sign}(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + b). \tag{21}$$

That is to say if $\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + b$ is positive then we predict 1, and if negative we predict -1.

The kernel functions are used to map the original data points to a higher dimensional space. For example, instead of defining features $\{x, x^2, x^3\}$ we could map these features through polynomial kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$

which produces d dimensional mapping for points x_i and x_j. Some commonly used kernels are:

- linear: $K(x_i, x_j) = x_i^T x_j$
- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0$
- sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

where γ, r, and d are kernel parameters as needed.

In the dual problem matrix Q is usually dense, high dimensional, and in some cases too large to store. To overcome this problem multiple decomposition methods algorithms for SVM have been developed. For example, by Osuna et al [33], Platt [35], Keerthi et al [34], Hsu and Lin [36], and Fan et al [37]. A decomposition method only handles a subset of α per iteration, so only a part of Q is needed. We denote this subset of variables by working set B. For example a decomposition method by Platt [35] called Sequential Minimal Optimization (SMO) reduces the optimization problem into a two-variable subproblem. So at each iteration we only solve a two-variable problem. The algorithm is:

1. Find α^1 as the initial feasible solution, vector of all zeros is a valid choice. Set $k = 1$.
2. If α^k is an optimal solution of $f(\alpha)$, stop. Otherwise, find a two-element working set $B = \{i, j\}$. An algorithm for working set selection will be given later. Define $N = \{1, \ldots, n\} \setminus B$, α_B^k, and α_N^K to be sub-vectors of α^k corresponding to B and N, respectively.
3. Solve the following sub-problem with the variable α_B:

$$\begin{aligned}
\underset{\alpha_i, \alpha_j}{\text{minimize}} \quad & \frac{1}{2} \begin{bmatrix} \alpha_B^T & (\alpha_N^k)^T \end{bmatrix} \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \begin{bmatrix} \alpha_B^T \\ \alpha_N^k \end{bmatrix} - \begin{bmatrix} e_B^T & e_N^T \end{bmatrix} \begin{bmatrix} \alpha_B^T \\ \alpha_N^k \end{bmatrix} \\
& = \frac{1}{2} \alpha_B^T Q_{BB} \alpha_B - (e_B + Q_{BN} \alpha_N^k)^T \alpha_B \\
& = \frac{1}{2} \begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} - (e_B + Q_{BN} \alpha_N^k)^T \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} \\
\text{subject to} \quad & 0 \leq \alpha_i, \alpha_j \leq C, \\
& y_i \alpha_i + y_j \alpha_j = -y_N^T \alpha_N^T,
\end{aligned}$$

where $\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$ is a permutation matrix Q.

4. Set α_B^{k+1} to be the optimal solution of the dual problem, and set $\alpha_N^{k+1} \leftarrow \alpha_N^k$. Set $k \leftarrow k+1$, and go to step 2.

The working set B is updated at each iteration. The working set selection is important as it directly contributes to a faster rate of convergence. The main idea is to find a pair of observations that violates the optimality condition based on the first order [34] or the second order gradient [37]. Here we present the working set selection by Keerthi et al [34] which uses the first order gradient:

1. Select

$$i \in \underset{t}{\operatorname{argmax}}\{-y_t \nabla f(\alpha^k)_t | t \in I_{\text{up}}(\alpha^k)\},$$

$$j \in \underset{t}{\operatorname{argmin}}\{-y_t \nabla f(\alpha^k)_t | t \in I_{\text{low}}(\alpha^k)\},$$

where

$$I_{\text{up}}(\alpha) = \{t | \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\},$$
$$I_{\text{low}}(\alpha) = \{t | \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\}.$$

2. Return $B = \{i, j\}$

where $\nabla f(\alpha) = Q\alpha + e$ is the gradient of $f(\alpha)$. The working set is chosen as the minimum and maximum indices of the gradient, though only the indices defined by I_{up} and I_{down} are considered.

The performance of support vector machines is heavily dependent on the chosen parameters defined by the kernel function. As such it is important to train multiple SVMs and estimate their relative performance. The simplest way to do this is to define a grid of parameters to test through. For example choosing polynomial kernel we could vary $d = \{1, 3, 5\}$ and $\gamma = \{2^{-10}, 2^{-8}, \ldots, 2^{10}\}$, and estimate models for all of the combinations. The relative performance can be estimated by using cross validation where we split the data in k pieces and use $k-1$ pieces to estimate the model and 1 piece to estimate performance, and this is then rotated k times so that every piece has been in performance estimation. More detailed description of this procedure can be found in section 7.1.

6 Kernel Density estimation

The kernel density estimator $\hat{f}_0 : \mathbb{R}^d \to \mathbb{R}$, based on data $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, is defined as

$$\hat{f}_0(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i),\ x \in \mathbb{R}^d, \tag{22}$$

where $K_h(x_1, \ldots\ x_d) = K(x_1/h_1, \ldots, x_d/h_d) / \prod_{i=1}^{d} h_i$, $K : \mathbb{R}^d \to \mathbb{R}$ is the kernel function, and $h = (h_1, \ldots, h_d)$ is the vector of smoothing parameters.

Possible kernel functions are discussed in section 7.1, but here we use the discretized kernel density:

$$\hat{f}(x) = \sum_{l=1}^{L} \hat{f}_0(x_l) I_{A_l}(x), \tag{23}$$

where $I_A(x) = 1$ when $x \in A_l$ and 0 otherwise, and set $A = \{A_1, \ldots, A_l\}$ is a partitioning of rectangle R, which contains the data. That is $\cup_{l=1}^{L} A_l = R$, where R is a rectangle containing observations and $A_n \cap A_m = \emptyset$ for $n \neq m$.

6.1 Adaptive Histogram Partitions

We estimate the adaptive partitioning with a stepwise algorithm of Klemelä [46]. We recursively split the observations into finer sets, as we try to locate some local optimum.

The partitioning is constructed using histogram estimator. We define the histogram corresponding to the partition $\mathcal{P} = \{A_1, \ldots, A_l\}$:

$$\hat{f}_0(x, \mathcal{P}) = \sum_{l=1}^{L} \frac{n_l/n}{\text{volume}(A_l)} I_{A_l}(x), \tag{24}$$

where n_l is the number of observations in A_l.

The partitioning is constructed using a gridded observation space. Each coordinate direction is subdivided along the corresponding axis. So the set of split points which defines the possible regions for partitions is:

$$\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_d\}, \tag{25}$$

where $\mathcal{G}_k \subset \mathbb{R}$ is a finite grid of split points in dimension k. We choose $\mathcal{G}_k$ to be the set of midpoints of a single dimension for the observations: $\mathcal{G}_k = \{Z_1^k, \ldots\ Z_{n-1}^k\}$, where

$$Z_i^k = \frac{1}{2}(x_{(i)}^k + x_{(i+1)}^k), \tag{26}$$

where $x_{(1)}^k, \ldots, x_{(n)}^k$ is the ordered statistics of the kth dimension of the observations $x_1^k, \ldots, x_n^k$. The intuition is to start splitting the data along these split points to get different size rectangular partitions. When rectangle $R \in \mathbb{R}^d$ is split at point $s \subset \mathbb{R}$ in dimension $k = 1, \ldots, d$, we obtain sets

$$R_{k,s}^{(\text{left})} = \{(x_1, \ldots, x_d \in R \mid x_k \leq s\}, \tag{27}$$

$$R_{k,s}^{(\text{right})} = \{(x_1, \ldots, x_d \in R \mid x_k > s\} \tag{28}$$

The split point s satisfies

$$s \in S_{R,k} \stackrel{\text{def}}{=} \mathcal{G} \cap \text{proj}_k(R), \tag{29}$$

where $\text{proj}_k(R) = R_k$, when $R = R_1 \times \cdots \times R_d$. When partition $\mathcal{P}$ is split using split point s, it is replaced by partition

$$\mathcal{P}_{A,k,s} = \mathcal{P} \setminus A \cup \left\{R_{k,s}^{(\text{left})}, R_{k,s}^{(\text{right})}\right\} \tag{30}$$

We begin with the set $A = \{R\}$ where R is the rectangle which holds all the data, and we start splitting the data until each rectangle holds at least m points. The algorithm for this is:

1. Start with the partition $\mathcal{P} = \{A\}$, where A at the first iteration is the rectangle that encompasses the whole data, and at the ith iteration is some rectangle from step two.

2. Consider a partition A. If the number of observations in partition A is less than or equal to m (minobs), we finish splitting that partition. Otherwise we split the partition in two by choosing a grid point that maximizes

$$\underset{(k,s)\in I_A}{\text{argmax}} \sum_{i=1}^{n} \log \hat{f}_0(x_i, \mathcal{P}_{A,k,s})$$

where $I_A = \{(k, s) \mid k = 1, \ldots, d, s \in S_{A,k}\}$, $S_{A,k}$ is the set of split points defined in (28), $\mathcal{P}_{A,k,s}$ is the partitioning defined in (29), and $\hat{f}_0$ is the histogram defined in (24).

The worst case computational complexity for a brute force algorithm is $O(dn^2)$ and the best case complexity is $O(dn \log n)$. The first iteration always takes dn steps. In the worst case we start splitting one point at a time so that to find the second split we have $d(n-1)$ steps, to find the third takes $d(n-2)$ steps, and so on. The best case scenario is we split the data in half so to find the second split we have $d(n/2)$ steps, to find the third $d(n/4)$ steps, and so on. In practice both of these scenarios are unlikely, and the choice of minobs parameter holds more importance as with larger value it is more likely to stop splitting early. This, however comes with decrease in accuracy.

6.2 Optimizing adaptive partitioning with Nelder-Mead algorithm

In this study we present a one dimensional approximation to the maximization step using Nelder-Mead algorithm (NM) [40]. NM is a derivative-free optimization technique that finds some local optimum by using heuristics. This is done by using plug-in values in the objective function and determining new plug-in values based on the old values. So with NM we could use the grid points as plug-in values, or estimate the split point in a continuous fashion. We chose the latter as it is more in line with the original NM algorithm.

The objective in NM is minimization so we switch our objective with the known optimization equality $-\max = \min$. NM is generally a multidimensional method, but we will use the one dimensional simplification. It is possible to apply the multidimensional version in two ways. This will be discussed after the algorithm is presented.

1. Initialization: Consider a rectangle A, points $\{x_1, \ldots, x_n\} \subset A$, and dimension k. The initial values are $z_1 = \frac{x_1+x_2}{2}$ and $z_2 = \frac{x_{n-1}+x_n}{2}$.

2. Evaluate
$$f(k,z) = \begin{cases} \underset{(k,s)\in I_A}{\operatorname{argmin}} -\sum_{i=1}^{n} \log \hat{f}_0(x_i, \mathcal{P}_{A,k,s}) & \min(x) < z < \max(x) \\ \infty & \text{otherwise} \end{cases} \tag{31}$$
and order the initial values so that $f(k,z_1) \leq f(k,z_2)$. The function evaluates to infinity when the point z is not inside the rectangle A.

3. Centroid point is now $z_o = z_1$.

4. Reflection point is now $z_r = z_o + \alpha(z_o - z_2)$. If $f(k,z_1) \leq f(k,z_r) \leq f(k,z_2)$ then replace z_2 with z_r and go to step 2.

5. If $f(k, z_r) < f(k, z_1)$, compute expansion point $z_e = z_r + \gamma(z_r - z_o)$. If the expansion point z_e is better than the reflection point z_r replace z_2 with z_e, else replace z_2 with z_r. If neither of these are true proceed to step 6.

6. Now we know that $f(k, z_r) \geq f(k, z_2)$. The contraction point is $z_c = z_o + \rho(z_2 - z_o)$, if now $f(k, z_c) < f(k, z_2)$ replace z_2 with z_c, and go to step 2. If everything "failed" calculate reduction: $z_2 = z_1 + \sigma(z_2 - z_1)$, and go to step 2.

Current implemented values for NM coefficients are $\alpha = 1, \gamma = 2, \rho = 0.25$ and $\sigma = -0.25$. Optimization is repeated for all the dimensions r times, stopped if $|f(k, z_1) - f(k, z_2)| < \epsilon$ or if z_2 is the same as in the previous step. Over the possible split points, one for each dimension, we choose the one that produces the smallest value.

The multidimensional algorithm isn't implemented because of the complexity of the algorithm. After we have found a multidimensional split point there are two issues. First, do we half every dimension of the rectangle or some combination of dimensions. A simple two dimensional rectangle has 3 choices: vertical, horizontal and cross splits. Generally d-dimensional rectangle has $\sum_{i=1}^{d-1} \binom{d}{i}$ combinations. Second, the optimization function is not numerically stable as the function evaluates to infinity if the point is not inside the rectangle. This can happen in steps 4-6, but only in step 6 can we force split points that are outside the rectangle. In 2-dimensional case this is not an issue as the only infinite value will most likely be overwritten in the coming steps. In d-dimensional case we can have up to $d - 1$ split points that are outside the rectangle. The choices are to either let the algorithm recover after multiple iterations or to overwrite all the split points that are outside the rectangle with some split point that is inside. Also step 6 is more likely to happen in d-dimensional case as the split point has to be outside in only one dimension. So one badly behaving dimension might ruin the whole process. Because of that we cannot even be certain that the multidimensional algorithm would have better performance.

An implementation of this algorithm using C++ can be found in the appendices.

7 Model selection

Model selection is used to find the best possible model from a set of possible models. The best usually means a balance between goodness of fit and complexity since in real life settings we also need generalization. The goodness of fit measures describe how well a model fits into the given data. For example BIC criterion is used in choosing the appropriate number of clusters in clustering algorithms. However blindly following goodness of fit statistics can lead to overfitting. The model describes the training data well but doesn't generalize elsewhere. So we have to balance between goodness of fit and generalization error. This dilemma is called the bias-variance tradeoff.

7.1 Cross validation

Cross-validation (CV) is one of the more popular choices for model selection [42]. Consider pairs $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbb{R}^d$, and $y_i \in \{0, 1\}$ are the class labels, and a classification function $f(x)$ which predicts 0 or 1 given the data x. The misclassification error is

$$e = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq f(x_i)), \tag{32}$$

which is the proportion of misclassified observations.

Simplest form of cross validation is holdout, where the data is split into two parts, the first part is the training data used in estimating $f(x)$ and the second is the test data that is used to calculate the error e. More generally we use k-fold cross validation where the data is split randomly into k parts, now $k-1$ splits are used in estimating $f(x)$ and kth split to calculate the error. This is done k times so that every fold is used in the error estimation. This procedure gives us error estimates $e_1, \ldots, e_k$, and then k-fold cross validation error is

$$e_{cv} = \frac{1}{k} \sum_{i=1}^{k} e_i \tag{33}$$

Naturally, cross validation is not constrained to a classification setting. We can switch the error function to mean square error $(1/n) \sum_{i=1}^{n} (x_i - y_i)^2$ and use the cross validation procedure to estimate the test error of a regression function. A survey by Arlot and Celisse details different cross validation procedures and their statistical properties [43].

7.2 Confusion matrix

Confusion matrix or error matrix is a table layout of predicted classes against the true classes. It is summarized as a 2×2 table which visualizes the interrelations and interactions between the predicted outcome and the actual outcome, as follows:

		Predicted outcome	
		Positive	**Negative**
True value	**Positive**	True positives (TP)	False negatives (FN)
	Negative	False positives (FP)	True negatives (TN)

A positive instance is usually our outcome of interest, for example in our case the positive outcome is the positive disease status. The true positives are all the positive cases we predict positive, and true negatives is defined analogously. False negatives are all the misclassified positive outcomes, and false positives are all the misclassified negative outcomes. From the confusion matrix we can derive the following statistics:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \qquad \text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}} \qquad \text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The accuracy measures how often we make the correct classification. Precision only considers the reliability of our positive predictions. The sensitivity and specificity are more commonly used in clinical settings. The sensitivity measures the proportion of correctly identified positive values, while specificity measures the correctly identified negative values. By considering these statistics one can determine what aspect of the model needs improving.

However, the metric of interest in the FlowCAP2 competition and in this study is the F1-measure which is the harmonic mean of precision and sensitivity:

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{sensitivity}}{\text{precision} + \text{sensitivity}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FN} + \text{FP}}.$$

By not considering true negative values the F1-measure might be more suitable for unbalanced class situations and in practical considerations.

7.3 MISE Comparison

The mean integrated square error (MISE) is used in evaluating density estimators:

$$\mathrm{MISE}(f, \hat{f}) = E \int_{\mathbb{R}^d} (f - \hat{f})^2, \tag{34}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the true density function and $\hat{f} : \mathbb{R}^d \to \mathbb{R}$ is an estimator of the density function. MISE is estimated by using the Monte Carlo method [44] from the target distribution and calculating the mean of integrated square error (ISE),

$$\mathrm{ISE}(f, \hat{f}) = \int_{\mathbb{R}^d} f^2 + \int_{\mathbb{R}^d} \hat{f}^2 - 2 \int_{\mathbb{R}^d} f\hat{f}, \tag{35}$$

The Monte Carlo estimation is done by taking a random sample of size m from the target distribution f, after that we estimate the adaptive partitioning and calculate ISE. This process is repeated K times, and by taking the mean of the ISE values we get MISE.

In this study we use piecewise constant estimator $\sum_{l=1}^{L} a_l I_{A_l}(x)$. This simplifies $\int_{\mathbb{R}^d} \hat{f}^2 = \sum_{l=1}^{L} a_l^2 \mathrm{volume}(A_l)$, and $\int_{\mathbb{R}^d} f\hat{f} = \sum_{l=1}^{L} a_l \int_{A_l} f$. The probability of rectangle $A = [a_1, b_1] \times \cdots \times [a_n, b_n]$ is

$$\int_A f = \sum_{i_1=1}^{2} \cdots \sum_{i_d=1}^{2} (-1)^{i_1 + \cdots + \imath_d} F(u_{1,i_1}, \ldots, u_{d,i_d}), \tag{36}$$

where $u_{i,1} = a_i$, and $u_{i,2} = b_i$ for $i = 1, \ldots, d$, and $F : \mathbb{R}^d \to \mathbb{R}$ is the distribution function.

The smoothing parameter for the kernel density estimator can be chosen based on the normal reference rule [45]:

$$h_i = \Big(\frac{4}{d+2}\Big)^{1/(d+4)} n^{-1/(d+4)} \hat{\sigma}_i, \tag{37}$$

for $i = 1, \ldots, d$ where $\hat{\sigma}_i$ is the sample standard deviation for the ith dimension.

8 Workflow & Results

In this study the AML dataset fro FlowCap 2 challenge is used. It is an open access data which can be downloaded from Flowrepository [15]. It consists of 359 individuals of which 43 have acute myeloid leukemia. From these individuals 8 tubes of blood samples or bone marrow aspirate were collected over one year period. The first tube is an isotype contro, the eight is unstained and the rest of the tubes are stained with different markers. The dataset was analyzed with the following workflow:

1. Each individual sample was logicle transformed and then observations with LOF > 1 were removed as outliers, the size of the neighborhood inf LOF was 100.

2. After pre-processing the individual samples were pooled into datasets according to their markers (tubes), so we have a total of 8 datasets. Each dataset was scaled before clustering, and the proportions of each cluster were used as the features in the support vector machine classification. The clustering methods used:

   - k-means with $k = 25$.
   - Level set clustering with minobs = 1000, itermax = 800, and smoothing parameter $h_i = \frac{4}{d+2}^{1/(d+4)} n^{-1/(d+4)} \hat{\sigma}_i$

3. After a dataset was clustered it was split in half into training and test data using stratified sampling based on the disease status. The training data was used to tune SVM parameters using 20-fold cross validation, and the test data was used to evaluate F1-measure.

   - $C = 2^{-3,\ldots,10}$
   - $\gamma = 2^{-10,\ldots,3}$

4. Step 3 was repeated 100 times to get a "confidence interval" for the F1-measure.

The reasoning for LOF and k-means parameters is given in section 8.2. In the original challenge there was only one test data which was half from the original, but rather than doing the splitting once we repeat the step 3 to get an estimation of variance for F1-measure. Nelder-Mead optimization for adaptive partitioning was written using Armadillo library [47], and R software 3.2.0 was used in all analyses [52]. The appropriate LSTC algorithms can be found in the delt and denpro packages of Klemelä and can

be downloaded from webpages http://cc.oulu.fi/~jklemela/denpro/ and the page http://cc.oulu.fi/~jklemela/delt/. Also pre-processing functions for cytometry and logicle transformation can be found from flowCore package [48], the local outlier from DMwR [50], and support vector machine from e1071 [49]. The kd-tree improvement for LOF was implemented using FNN package [51].

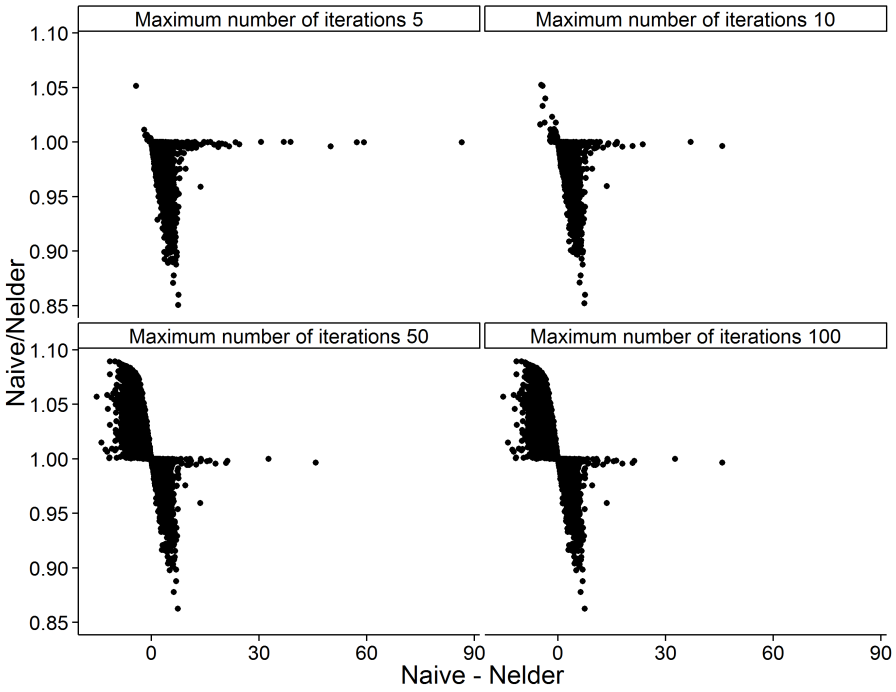
To use the C++ scripts in the appendices the densplitter function in the delt package needs to be changed. This means that either the findsplitter function inside densplitter is switched to NelderMeadFindSplitter (the C++ script) or an appropriate if-else is added.

8.1 Performance of Nelder-Mead algorithm in adaptive partitioning

The objective function in adaptive partitioning can vary a lot in the process of splitting. This is because at first we might be splitting tens of thousands of observations into rectangles, while in the later stages we are doing the fine tuning with hundreds of observations. Figure 1 displays absolute difference and the ratio of objective function values between the naive and NM algorithm for one process from start to finish. The data is 5 dimensional Gaussian mixture with 40000 observations. The minimum number of observations (minobs) inside a rectangle is 20, and maximum iterations varied from 5, 10, 50 and 100. In this example there's no difference between 50 and 100 iterations. Sometimes NM finds a better solution because of its continuous nature. This happens only in the later stages when there are relatively low number of observations and we are dealing with small objective function values. This can be seen from the ratio and the absolute difference. NM can find a 10% better solution, but the absolute difference might only be 10.

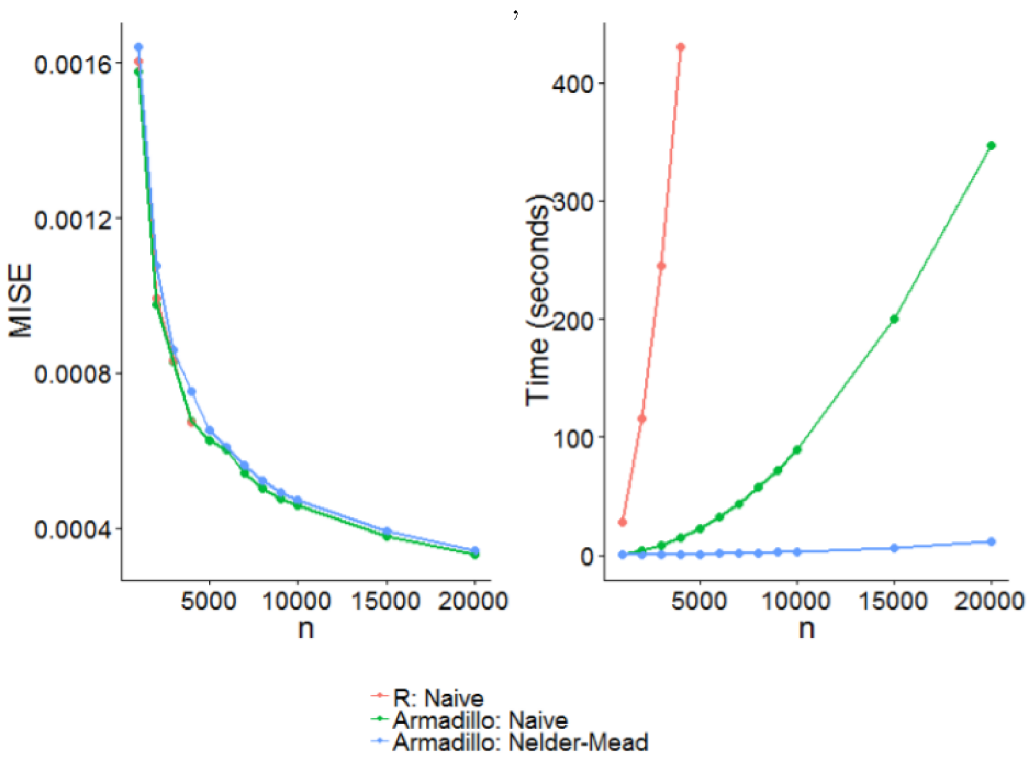
Figure 2 summarizes the MISE estimation for 100 replications and CPU time for a 5-dimensional normal mixture with 40000 observations and standard deviation of 1. Nelder-Mead was carried out to a maximum of 100 iterations (itermax), or achieving tolerance of 10^{-6}. For both algorithms minobs was 10. Both NM and naive algorithms have the same performance in MISE. This was expected because the algorithms give approximately the same results. However Nelder-Mead is much faster. Both algorithms show exponential growth with increasing n, but Nelder-Mead has smaller slope. As noted in section 7.3 the naive algorithm has $O(dn^2)$ worst case complexity, and $O(dn \log n)$ best case complexity. The first step always takes dn steps for the naive algorithm, but for the NM it can be thought to take $d \cdot \text{itermax}$ steps. With reasoning used in section 7.3, NM can be naively thought to

Figure 1: Comparison on how naive and NM algorithms perform in a single run in terms of the absolute difference and ratio of found optimum value



achieve $O(dn \cdot \text{itermax})$ worst case complexity, and $O(dn \log \text{itermax})$ best case complexity. Unfortunately those are just crude estimates as the time complexity of Nelder-Mead is not known. But that is the intuition why NM is a lot faster when itermax $<<$ n, and that is not accounting for the other two stopping criteria.

Figure 2: A comparison of brute force and Nelder-Mead algorithm for a 5-dimensional Gaussian mixture

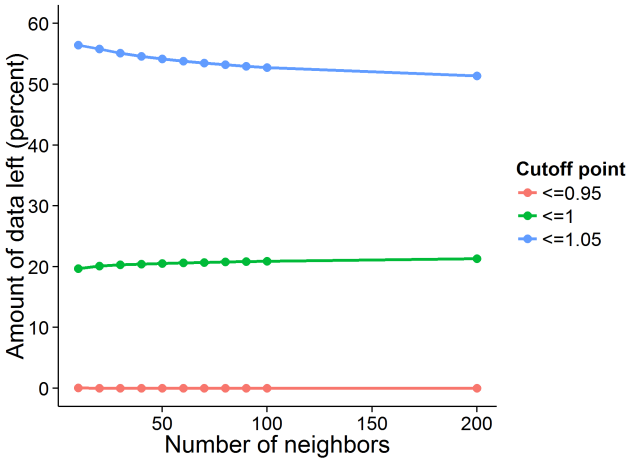


8.2 How local outlier factor effects sample size

Local outlier factor has not been widely used in flow cytometry, only one article was found [53]. Figure 3 displays percentage decrease in data size with different neighborhood sizes for tube 1. LOF is robust for the number of neighbors as the ratio of outliers is more dependent on the cutoff point. For cutoff points 0.95, 1, and 1.05 there were around 0.05%, 20% and 55% of data left for further analysis. This is a practical decrease from around 10 million observations to around 30 thousand, 2 million and 5.5 million observations respectively. There is still a small increase in available data while the cutoff point is 1, and a decrease when it is 1.05 while we increase the amount of neighbors. This is a property of the algorithm as observations will tend towards 1 as the size of the neighborhood grows.

For k-means the initial size of the neighborhood varied, but a single value was chosen for the final results as higher neighborhood sizes didn't increase performance. A comparison of different LOF cutoff points to k-means results

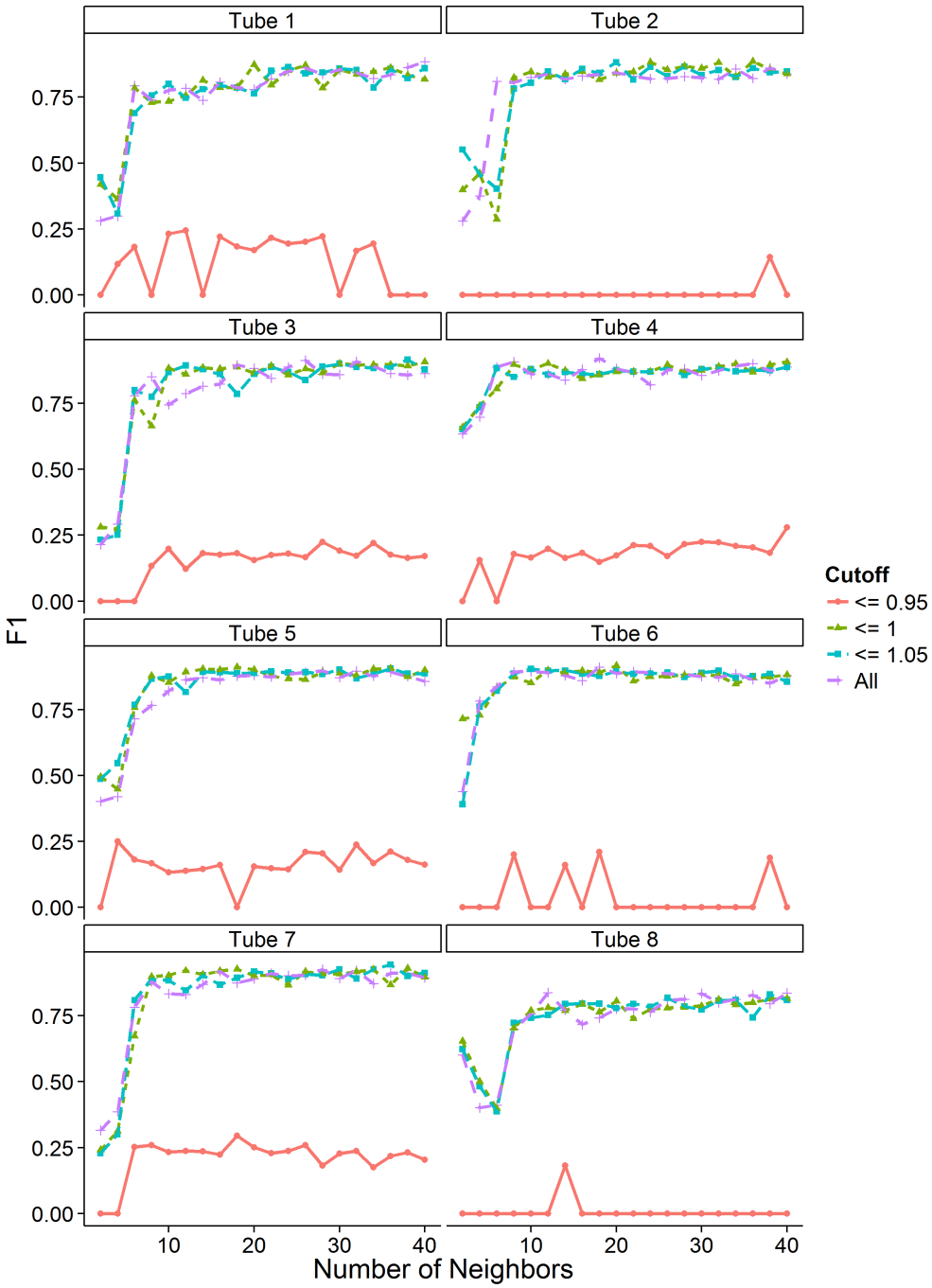
Figure 3: Comparison how local outlier factor with different cutoff points and neighborhood sizes effects the ratio of available data for further analysis



is shown in Figure 4. The F1-measure reaches its peak around 20 neighbors for most tubes, but the F1-measure is mostly stable after 10 neighbors with only slight increase in F1-measure with more neighbors. The LOF cutoff point did not affect the results, unless 0.95 was used. This was expected as with that cutoff there's only a marginal amount of data left, but it's still surprising that for tubes 3, 4 and 5 there are populations that are able to provide some discrimination between the two classes.

Figure 4: Mean value of F1-measure for comparison of different LOF cutoff points to k-means performance with different number of neighbors

,

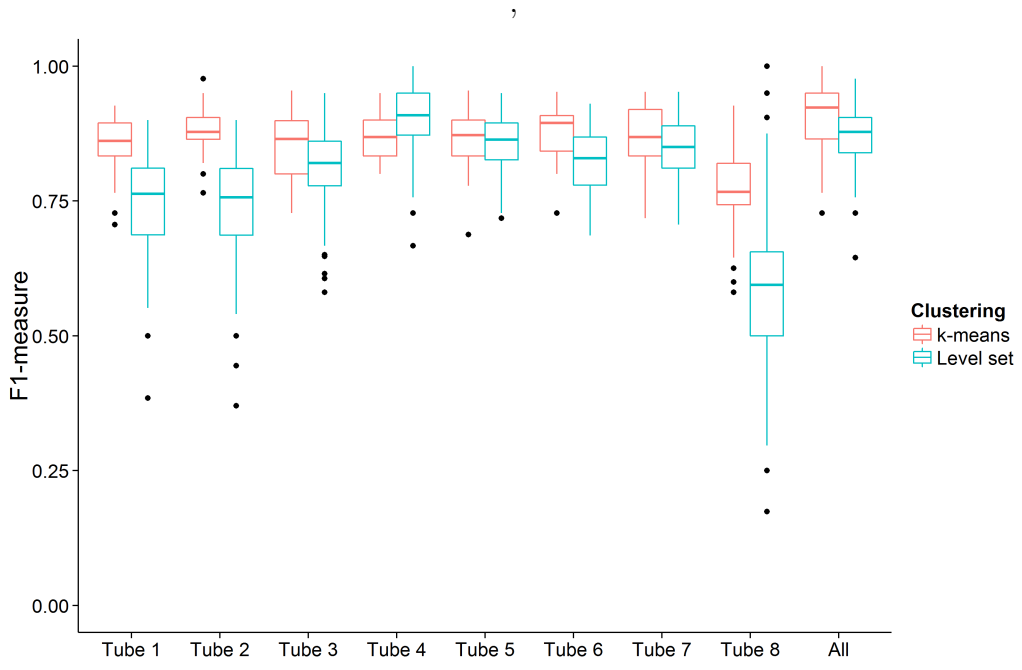


8.3 Comparison of k-means and level set clustering

Figure 5 compares k-means and level set clustering. While level set clustering gives a good overall result it's still noticeably worse in most tubes and in the combined analysis. The individual tube F1-measure median is between 0.75 and 0.92 without considering tube 8 which had 0.55. This is a strong contrast to k-means which measure is around 0.9 in most tubes except tube 8 performing slightly worse with F1-measure of 0.75. In the combined analysis k-means achieved 0.95 and LSTC 0.92. While the difference isn't huge in the FlowCAP 2 challenge there were multiple solutions which achieved F1-measure of 1.

In the combined analysis LSTC performed worse than the individual tube 4 which seems to be sufficient in itself to almost beat the combined k-means analysis. This would indicate that there are problems with parameters of LSTC. The interaction between smoothing parameter and the minimum observation count in adaptive partitioning was not explored.

Figure 5: A comparison of F1-measures between tubes and all tubes for LSTC and k-means



9 Discussion

This is the first time level set clustering by leafsfirst algorithm by Klemelä has been used in this kind of setting. As such expectations were fairly low, and because of that we picked a fairly straightforward task with known results.

We improved the adaptive partitioning scheme with NM, which gave good approximations and a significant speed increase, but the algorithm still had exponential time curve because of the inherit partitioning scheme. Exploring different optimization algorithms should be done in the future. This is because NM is a fairly old method and more refined algorithms are available. Also other stopping criterion in the partitioning process besides the minimum observation count in the rectangle should be explored, for example log likelihood, that is, partitioning a rectangle until log likelihood improves only slightly.

Finding outliers with local outlier factor gave mixed results. While it is good that we can maintain our results with 20% of the original data, the signal-to-noise ratio did not improve. So LOF might not be better than randomly selecting a few hundred of thousands observations. But if we want a systematic way to reduce the size of the data, LOF is an option. Further exploring LOF with different dataset should be done, and of course comparing it to random selection.

Unfortunately LCTS had mixed results and overall did not perform as well as k-means. One of the reasons for this is the high variance in LCTS results that might be a consequence of a poorly defined smoothing parameter or a high minobs parameter. Because of that the interaction between the smoothing parameter and the minimum observation count in adaptive partitioning should be explored. Also because of computational issues more of the code should be translated into C or C++ so that smaller minobs parameters could be used.

One of the interesting questions whether we could define the smoothing parameter adaptively for each rectangle instead of choosing a global smoothing parameter. Or could we somehow estimate a global smoothing parameter based on our partitioning. However in practice different kind of datasets should be evaluated and LCTS compared against different kernel density methods.

References

[1] Pyne, S., Hu, X., Wang, K., Rossin, E., Lin, T. I., Maier, L. M., Baecher-Allan, C., McLachlan, G. J., Tamayo, P., Hafler, D. A., Jager, P. L. D. and Mesirov J. P. (2009), *"Automated High-dimensional Flow Cytometric Data Analysis"*, Proceedings of the National Academy of Sciences, 106(21): 8519-8524.

[2] Gottardo, R., Brinkman, R. R., Luta, G. and Wand, M. P. (2009), *"Recent Bioinformatics Advances in the Analysis of High Throughput Flow Cytometry Data"*, Advances in Bioinformatics, article ID 461763.

[3] Lugli, E., Roederer, M. and Cossarizza A. (2010), *"Data Analysis in Flow Cytometry: The Future Just Started"*, Cytometry A, 77: 705–713.

[4] Aghaeepour, N., Nikolic, R., Hoos, H. H. and Brinkman, R. R. (2011), *"Rapid Cell Population Identification in Flow Cytometry Data"*, Cytometry A, 79(1): 6-13.

[5] Lo K, Brinkman R. R., and Gottardo, R. (2008), *"Automated Gating of Flow Cytometry Data Via Robust Model-based Clustering"*, Cytometry A, 73 (4):321–332

[6] Zare, H., Shooshtari, P., Gupta, A. and Brinkman R. B. (2010), *"Data Reduction for Spectral Clustering to Analyse High Throughput Flow Cytometry Data"*, BMC Bioinformatics, 11: 403

[7] Azad, A., Langguth, J., Fang, Y., Qi, A. and Pothen, A. (2010), *"Identifying Rare Cell Populations in Comparative Flow Cytometry"*, Lecture Notes in Computer Science, 6293: 162 175.

[8] Azad, A., Pyne, S. and Pothen, A. (2012), *"Matching Phosphorylation Response Patterns of Antigen-receptor-stimulated T Cells Via Flow Cytometry"*, BMC Bioinformatics, 13(Suppl 2):S10.

[9] Pyne, S., Hu, X., Wang, K., et al. (2009), *"Automated High-dimensional Flow Cytometric Data Analysis"*, Proceedings of the National Academy of Sciences, 106 (21): 8519–8524.

[10] Aghaeepour, N., Finak, G., The FlowCAP Consortium, The Dream Consortium, et al. (2013), *"Critical Assessment of Automated Flow Cytometry Data Analysis Techniques"*, Nature Methods, 10 (3): 228–238.

[11] Hahne, F., Khodabakhshi, A. H., Bashashati, A., et al. (2010), *"Per-channel Basis Normalization Methods for Flow Cytometry Data"*, Cytometry A, 77 (2): 121–131.

[12] Parks, D. R., Roederer, M. and Moore, W. A. (2006), *"A New "Logicle" Display Method Avoids Deceptive Effects of Logarithmic Scaling for Low Signals and Compensated Data"*, Cytometry A, 69 (6): 541–51 .

[13] Roederer, M. (2002), *"Compensation in Flow Cytometry"*, Current Protocols in Cytomometry, Chapter 1: Unit 1.14.

[14] Le Meur, N., Rossini, A., Gasparetto, M., Smith, C., Brinkman, R. R. and Gentleman, R. (2007), *"Data Quality Assessment of Ungated Flow Cytometry Data in High Throughput Experiments"*, Cytometry A, 71A (6): 393–403.

[15] Spidlen, J., Breuer, K., Rosenberg, C., Kotecha, N. and Brinkman, R. R. (2012), *" FlowRepository - A Resource of Annotated Flow Cytometry Datasets Associated with Peer-reviewed Publications"*, Cytometry A, 81(9): 727-31.

[16] Lugli, E., Pint, M., Nasi, M., et al. (2007), *"Subject Classification Obtained By Cluster Analysis and Principal Component Analysis Applied to Flow Cytometric Data"*, Cytometry A, 71(5): 334–344.

[17] Roederer, M., Moore, W., Treister, A., Hardy, R. R., and Herzenberg L. A. (2001), *"Probability Binning Comparison: A Metric for Quantitating Multivariate Distribution Differences"*, Cytometry, 45: 47–55.

[18] Bashashati, A. and Brinkman, R. R. (2009), *"A Survey of Flow Cytometry Data Analysis Methods"*, Advances in Bioinformatics, article ID 584603.

[19] Mahalanobis, P. C. (1936), *"On the generalized Distance in Statistics"*, Proceedings of the National Institute of Sciences of India, 2(1): 49–55.

[20] Forgy, E. W. (1965), *"Cluster Analysis of Multivariate Data: Efficiency Vs Interpretability of Classifications"*, Biometrics, 21: 768–769.

[21] Hartigan, J. A. and Wong, M. A. (1979), *"A k-means Clustering Algorithm"*, Applied Statistics, 28: 100–108.

[22] Lloyd, S. P. (1982), *"Least Squares Quantization in PCM"*, IEEE Transactions on Information Theory, 28: 128–137.

[23] MacQueen, J. (1967), *"Some Methods for Classification and Analysis of Multivariate Observations"*, In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1: 281–297, Berkeley, CA: University of California Press.

[24] Pelleg, D., and Moore, A. (2000), *"X-means: Extending k-means with Efficient Estimation of the Number of Clusters"*, Proceeding of the 17th International Conference on Machine Learning, 727-734.

[25] Hamerly, G., and Elkan, C. (2003), *"Learning the K in k-Means"*, In Neural Information Processing Systems.

[26] Rousseeuw, P. J., Kaufman, L. and Trauwaert, E. (1996), *"Fuzzy Clustering Using Scatter Matrices"*, Computational Statistics and Data Analysis, 23(1): 135–151.

[27] McLachlan, G. and Peel, D. (2000), *"Finite Mixture Models"*, Wiley-Interscience, New York.

[28] McLachlan, G. J. and Basford, K. E. (1988), (*"Mixture Models: Inference and Applications to Clustering"*, Marcel Dekker, New York

[29] Bandfield, J., and Raftery, A. (1993), *"Model-Based Gaussian and Non-Gaussian Clustering"*, Biometrics, 49: 803-821.

[30] Cortes, C. and Vapnik, V. (1995), *"Support-vector networks"*, Machine Learning, 20(3): 273.

[31] Hastie, T., Tibshirani, R., and Friedman, J. (2001), *"The Elements of Statistical Learning"*, Springer Series in Statistics.

[32] Moore, W. A. and Parks, D. R. (2012), /Update for the logicle data scale including operational code implementations", Cytometry A, 81: 273–277.

[33] Osuna, E., Freund, R. and Girosi, F. (1997), *"Training Support Vector Machines: An application to Face Detection"*, In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 130–136.

[34] Keerthi, S. S., Shevade S. K., Bhattacharyya, C. and Murthy, K. R. K. (2001), *"Improvements to Platt's SMO Algorithm for SVM Classifier Design"*, Neural Computation, 13: 637–649.

[35] Platt, J. C. (1998), *"Fast Training of Support Vector Machines Using Sequential Minimal Optimization"*, Advances in Kernel Methods - Support Vector Learning, Cambridge, MIT Press.

[36] Hsu, C.-W. and Lin C.-J. (2002), *"A Simple Decomposition Method for Support Vector Machines"*, Machine Learning, 46: 291–314.

[37] Fan, R.-E., Chen P.-H. and Lin, C.-J. (2005), *"Working Set Selection Using Second Order Information for Training SVM"*, Journal of Machine Learning Research, 6: 1889–1918.

[38] Breunig, M. M., Kriegel H.-P, Ng R. T. and Sander J. (2000), *"LOF: Identifying Density-based Local Outliers"*, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD: 93–104.

[39] Bentley, J. L. (1975), *"Multidimensional Binary Search Trees Used for Associative Searching"*, Communications of the ACM, 18 (9): 509.

[40] Nelder, J. and Mead, R. (1965), *"A Simplex Method for Function Minimization"*, Computer Journal, 7: 308–313.

[41] Hartigan, J. (1975), *"Clustering Algorithms"*, John Wiley& Sons.

[42] Geisser, S. (1993), *"Predictive Inference"*. Chapman and Hall, New York.

[43] Arlot, S. and Celisse, A. (2010), *"A Survey of Cross-validation Procedures for Model Selection"*, Statistics Surveys, 4: 40-79.

[44] Metropolis, N. and Ulam, S. (1949), *"The Monte Carlo Method"*, Journal of the American Statistical Association, 4: 335-341.

[45] Silverman, B. W. (1986). *"Density Estimation for Statistics and Data Analysis"*. Chapman and Hall, London.

[46] Klemelä, J. (2009). *"Smoothing of Multivariate Data: Density Estimation and Visualization"*. Wiley, New York.

[47] Conrad, S. (2010), *"Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments"*. Technical Report, NICTA.

[48] Ellis, B., et al. *"FlowCore: Basic Structures for Flow Cytometry Data"*. R package version 1.32.2.

[49] Meyer, D., et al. (2014), *"e1071: Misc Functions of the Department of Statistics (e1071)"*, TU Wien, R package version 1.6-4. http://CRAN.R-project.org/package=e1071

[50] Torgo, L. (2010), *"Data Mining with R, Learning with Case Studies"*, Chapman and Hall/CRC, http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR

[51] Beygelzimer, et al. (2013). *FNN: Fast Nearest Neighbor Search Algorithms and Applications"*, *R package version 1.1. http://CRAN.R-project.org/package=FNN*

[52] R Core Team (2015). "R: A Language and Environment for Statistical Computing"*, R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/*

[53] Szekely, E., Poncelet, P., Masseglia, F., Teisseire, M. and Cezar, R. (2013), "A Density-Based Backward Approach to Isolate Rare Events in Large-Scale Applications"*, Johannes Fürnkranz and Eyke Hüllermeier and Tomoyuki Higuchi. DS'13: Discovery Science, Singapore. Springer, 249-264, Lecture Notes in Computer Science.*

Appendices

```
/*
functions massone and denssr are translated from
the original code by Klemela.
Also the Rcpp bindings are in place.
*/

//[[Rcpp::depends(RcppArmadillo)]]
#define ARMA_64BIT_WORD
#include <RcppArmadillo.h>
using namespace Rcpp;
using namespace arma;

// Returns volume of a "vector" rectangle
// [[Rcpp::export]]
double massone(vec rec) {
  int d = rec.n_elem/2;
  uvec i = 2*linspace<uvec>(0, d-1, d);
  return prod(rec(i+1) - rec(i));
}

// loglikelihood estimation of a rectangle
// [[Rcpp::export]]
double denssr(double volume, int havlkm, int n,
  String method = "loglik", double mix = 0){
  if(havlkm == 0){
    return 0;
  }else if(method == "loglik"){
    return havlkm*log(havlkm/(n*volume));
  }else{
    if(mix == 0){
      return pow(havlkm, 2)/(n*volume);
    }else{
      return mix*(2-mix)*pow(havlkm, 2)/(n*volume);
    }
  }
}
```

```
// Objective function for the Nelder-Mead Algorithm
// [[Rcpp::export]]
double NelderMeadOptiFun(double jakopiste, int d, vec x,
  vec rec, int n, String method = "loglik"){
  int leftobslkm = sum(x < jakopiste);
  int rightobslkm = sum(x > jakopiste);
  //Require that both rectangles have observations
  if(leftobslkm < 1 || rightobslkm < 1){
    return datum::inf;
  }else{
    vec leftrec = rec;
    vec rightrec = rec;
    leftrec(2*d+1) = jakopiste;
    rightrec(2*d) = jakopiste;
    double volumeleft = massone(leftrec);
    double volumeright = massone(rightrec);
    return -(denssr(volumeleft, leftobslkm, n, method) +
              denssr(volumeright, rightobslkm, n, method));
  }
  return datum::inf;
}

/*
Returns the optimal split point using the
Nelder-Mead Algorithm. The algorithm fails
completely if there is only 1 unique point per
dimension.
*/
// [[Rcpp::export]]
List NelderMeadFindSplitter(mat grid, mat x, vec rec,
    int n, String method = "loglik",
    int itermax = 1e2, double tol = 1e-6){
  double alpha = 1, gamma = 2, rho = -0.25, sigma = 0.25;
  double refVal, expVal, conVal, minVal, maxVal, centroid,
    reflection, expansion, contracted, halt;
  int d = x.n_cols, iter = 1, reduction = 0;
  vec val = ones<vec>(2), fval = ones<vec>(2), partX;
  mat results = zeros<mat>(d, 3);
  uvec sorting = ones<uvec>(1);
  for(int i = 0; i < d; i++){
    partX = x.col(i);
    fval = unique(partX);
```

```
//Nelder-Mead algorithm if there are atleast 2
// unique data points
if(fval.n_elem > 1){
  partX = partX(sort_index(partX));
  minVal = (fval(0) + fval(1))/2;
  maxVal = (fval(fval.n_elem - 1) +
            fval(fval.n_elem - 2))/2;
  val(0) = minVal;
  val(1) = maxVal;
  fval = ones<vec>(2);
  fval(0) = NelderMeadOptiFun(val(0), i, partX,
                              rec, n, method);
  fval(1) = NelderMeadOptiFun(val(1), i, partX,
                              rec, n, method);
  iter = 1;
  reduction = 0;
  halt = tol+1;
  while(iter < itermax && halt > tol && reduction < 2){
    sorting = sort_index(fval);
    val = val(sorting);
    fval = fval(sorting);
    centroid = val(0);
    reflection = centroid + alpha*(centroid - val(1));
    refVal = NelderMeadOptiFun(reflection, i, partX,
                               rec, n, method);
    if(fval(0) <= refVal && refVal < fval(1)){
      reduction = 0;
      val(1) = reflection;
      fval(1) = refVal;
    }else if(refVal < fval(0)){
      reduction = 0;
      expansion = reflection + gamma*(reflection - centroid);
      expVal = NelderMeadOptiFun(expansion, i, partX,
                                 rec, n, method);
      if(expVal < refVal){
        val(1) = expansion;
        fval(1) = expVal;
      }else{
        val(1) = reflection;
        fval(1) = refVal;
      }
```

```
        }else{
          contracted = centroid + rho*(centroid - val(1));
          conVal = NelderMeadOptiFun(contracted, i, partX,
                                     rec, n, method);
          if(conVal < fval(1)){
            reduction = 0;
            val(1) = contracted;
            fval(1) = conVal;
          }else{
            reduction = reduction+1;
            val(1) = val(1) + sigma*(val(1) - val(0));
            if(val(1) < minVal){
              val(1) = minVal;
            }
            if(val(1) > maxVal){
              val(1) = maxVal;
            }
            fval(1) = NelderMeadOptiFun(val(1), i, partX,
                                        rec, n, method);
          }
        }
        iter ++;
        halt = std::abs(fval(0) - fval(1));
      }
    }else{
      val(0) = 0;
      fval(0) = datum::inf;
    }
    results(i, 0) = i;
    results(i, 1) = val(0);
    results(i, 2) = fval(0);
}
// Removal of possible infinite values
// and finding the optimum result
sorting = find_finite(results.col(2));
results = results.rows(sorting);
sorting = sort_index(results.col(2));
results = results.rows(sorting);
uword valio;
vec absed = abs(grid.col(results(0, 0)) - results(0, 1));
double minimum = absed.min(valio);
absed = linspace<vec>(0, grid.n_rows-1, grid.n_rows)(valio);
```

```
    return List::create(
      Named("val") = results(0, 1),
      Named("vec") = results(0, 0),
      Named("valio") = absed,
      Named("last") = results(0, 2));
}
```