PPCI: an R Package for Cluster Identification using Projection Pursuit

by David P. Hofmeyr and Nicos G. Pavlidis

Abstract This paper presents the R package **PPCI** which implements three recently proposed projection pursuit methods for clustering. The methods are unified by the approach of defining an optimal hyperplane to separate clusters, and deriving a projection index whose optimiser is the vector normal to this separating hyperplane. Divisive hierarchical clustering algorithms that can detect clusters defined in different subspaces are readily obtained by recursively bi-partitioning the data through such hyperplanes. Projecting onto the vector normal to the optimal hyperplane enables visualisations of the data that can be used to validate the partition at each level of the cluster hierarchy. **PPCI** also provides a simplified framework in which the clustering models can be modified in an interactive manner. Extensions to problems involving clusters which are not linearly separable, and to the problem of finding maximum hard margin hyperplanes for clustering are also discussed.

Introduction

Clustering refers to the problem of identifying distinct groups (clusters) of relatively homogeneous points within a collection of data, with no explicit knowledge about the group associations of any of the points. Various definitions of what constitutes a cluster have led to a multitude of clustering algorithms (Jain et al., 1999), with no universal consensus and no definition which is appropriate for all applications. We consider the clustering of data sets which are embedded in Euclidean $\mathbb{R}^d$. Here, without access to a "ground truth solution", clusters are essentially determined using the relative distances between points. Difficulties can arise, however, when distances between the original data become less informative about the underlying cluster structure due to the presence of irrelevant and noisy features, as well as correlations among subsets of features (Bach and Jordan., 2006; Kriegel et al., 2009; Niu et al., 2011). Such characteristics frequently arise in high dimensional applications, and can make the clustering problem especially challenging. In this work we focus on the problem of finding low dimensional data representations that best expose the separation of clusters. Unlike some related work we do not aim to optimally preserve, after dimension reduction, the pairwise distances from the original data. Instead we seek the best discrimination of clusters when viewed within these low dimensional representations.

We focus on linear dimension reduction techniques, which lead to linear cluster separators defined using a collection of hyperplanes. Linear dimension reduction may be viewed as a projection into a linear subspace of the input space. Linear projection methods are attractive for their simplicity and relative computational efficiency. However, they are limited in their inability to accurately capture non-linear cluster structures. Despite this limitation, hyperplane separators derived from linear projections have been applied successfully to clustering problems from extremely diverse domains (Boley, 1998; Tasoulis et al., 2010; Pavlidis et al., 2016; Hofmeyr and Pavlidis, 2015; Hofmeyr, 2017). A principled approach to finding high quality linear subspaces for clustering is via projection pursuit. Projection pursuit refers to a class of dimension reduction techniques which seek to optimise over all linear projections of the data a given measure of interestingness, called a *projection index* (Huber, 1985). Principal Component Analysis (PCA) is arguably the most popular projection pursuit method. In PCA the projection index can be formulated as the variance of the projected data. Although PCA has been successfully applied in numerous clustering problems (Boley, 1998; Ding and He, 2004; Tasoulis et al., 2010), there is no guarantee that the projected data will contain any cluster structure. This occurs when clusters are only separated within dimensions of low total data variability. We have found that tailoring the dimension reduction to the clustering problem has the potential to vastly improve performance over off-the-shelf methods, such as PCA.

In this paper we present the R package **PPCI** which provides implementations of three recently developed projection pursuit methods for clustering. The projection indices are motivated by three well established approaches to clustering; namely density clustering, centroid based clustering and clustering by graph cuts. Our experience is that because these approaches incorporate the clustering objectives directly into the projection index, they have the potential to outperform popular general purpose dimension reduction techniques in numerous applied problems (Hofmeyr and Pavlidis, 2015; Pavlidis et al., 2016; Hofmeyr, 2017). More generally, projection pursuit methods that employ a clustering criterion as a projection index have been shown to be effective for revealing cluster structure in the projected data in a number of other works (Peña and Prieto, 2001; Bolton and Krzanowski, 2003; Krause and Liebscher, 2005; Niu et al., 2011, 2014; Hofmeyr et al., 2019). This framework also allows a user to seek projections of the data which are suitable for identifying specifically the types

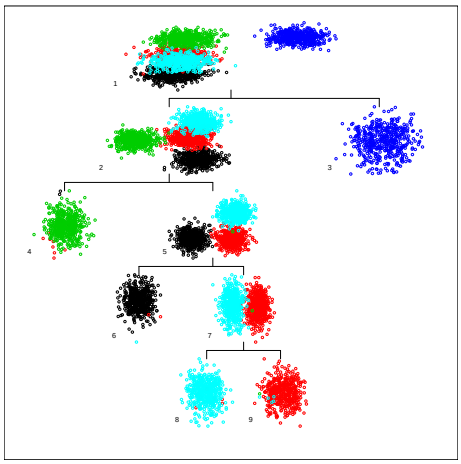


Figure 1: A divisive hierarchical clustering model for a simple simulated data set. Colours indicate the true class labels.

of clusters which are of interest. Alternatively, without any prior knowledge regarding the type of clusters which might be present in a data set, a user may consider multiple alternatives, and rely on the informative visualisations which are made available through dimension reduction to compare the multiple clustering solutions obtained.

In **PPCI** we adopt the approach of recursively identifying univariate subspaces which allow the separation of at least one cluster from the remainder. Each such subspace gives rise to a binary partition of the data, which is formed by a hyperplane separator in the original data space. The recursive manner in which the separators are obtained results in a divisive hierarchical clustering model, taking the form of a binary decision tree. The final clusters, which arise in the leaves of the tree, are formed by the repeated splitting of the data along the path from the root node. An illustration of such a model can be seen in Figure 1. Each scatter-plot shows the data assigned to the corresponding node in the model, based on the binary partitions along the path from the root node at the top of the figure. The colours indicate the true class labels, which are included only for illustrative purposes.

We prefer the approach described above to the alternative of seeking a single multivariate subspace in which to obtain a single flat clustering solution (Ding and Li, 2007; Niu et al., 2011). Utilising multiple subspaces to identify the different clusters allows us to better accommodate clusters defined at different scales. The visualisation possibilities it enables are also more instructive and immediate. Each subspace is of very low dimension, and by definition directly captures what distinguishes a particular cluster, or clusters, from the remainder. On the other hand a single visualisation of a complete clustering taken from a single subspace may not capture very well the definition of any individual cluster(s). This visualisation makes the validation and modification of a clustering solution substantially more straightforward, since the definition of each cluster within the model can be inspected separately.

Related work: As far as we are aware the only existing R package which combines projection pursuit and clustering is **ProjectionBasedClustering** (Thrun et al., 2018; Thrun, 2018). **ProjectionBasedClustering** provides numerous popular dimension reduction techniques, including both linear and non-linear methods. The package also offers visualisation tools, based on the generalised U-matrix (Ultsch and Thrun, 2017), which allow a user to identify and validate clustering solutions within the transformed data. A main focus of **ProjectionBasedClustering** is the identification of a single two-dimensional representation of the data which preserves (either locally or globally) the pairwise distances in the original data, as well as possible. This approach may therefore be more appropriate when the clusters can be distinguished well based on the original distances. In contrast, the algorithms in **PPCI** do not seek to maintain the spatial structure of the high-dimensional data but instead seek one-dimensional projections that maximally separate clusters.

A second related package is **subspace**, which focuses on a different approach to clustering based on projections. These methods define clusters through adjacent grid cells which have high data density when projected onto multiple axes of the input space. As a result, whereas the clusters admitted by these methods are not restricted to being linearly separable, as in **PPCI**, they are restricted to axis-parallel subspaces.

The remaining paper is organised as follows: We begin by introducing the main functionalities of the **PPCI** package, along with a brief discussion on each of the methods included. We then go on to illustrate their implementation on an example application in image clustering. Next we highlight the

usefulness of the visualisations offered in the package towards modifying and validating clustering solutions. Finally we look at extensions to the standard formulation we adopt, including maximum margin clustering and clustering of data whose clusters cannot be directly separated using only linear separators.

Installation and basic functionality of PPCI

The **PPCI** package can be installed from the Comprehensive R Archive Network (CRAN) from within the R console:

```
> install.packages("PPCI")
> library(PPCI)
```

A development version of the package is also available, and may be installed directly from GitHub with the following commands:

```
> if(!("devtools"%in%installed.packages())) install.packages("devtools")
> devtools::install_github("DavidHofmeyr/PPCI")
> library(PPCI)
```

The development version contains additional data sets, and some updates to the package will be introduced in the development version prior to rigorous testing which precedes addition to the release version on CRAN.

The main functions provided in **PPCI** are `mddc()`, `mcdc()` and `ncutdc()`, which produce divisive hierarchical clustering models. The common suffix `dc` refers to "divisive clustering", with the differentiating components `md`, `mc` and `ncut` referring to "minimum density", "maximum clusterability" and "normalised cut", respectively. In addition the functions `mdh()`, `mch()` and `ncuth()` obtain a single hyperplane separator using projection pursuit. Here the suffix `h` simply refers to "hyperplane". A brief description of these methods, and their associated clustering objectives is given in Table 1. These functions require as input only the data matrix, `X`, and in the case of the divisive clustering algorithms also the number of clusters to extract, `K`. Multiple optional arguments allow the user to modify the hyperparameters of the clustering objectives, as well as the specifics of the optimisation algorithm used to solve the projection pursuit problem. When these optional arguments are not provided by the user, they are assigned default settings which have been chosen in some cases based on supporting theory, and in others based on the authors' experiences with the methods. We note that although these arguments have the potential to result in substantially different outputs, the methods implemented in **PPCI** are often more robust to changes in their parameters than are the corresponding corresponding clustering objectives applied to the full dimensional data. The reason for this is that the projection pursuit will tend to lead to projections of the data for which the particular settings of the parameters yield a strong separation of clusters, and as a result the clustering solution obtained is reasonably stable for relatively small changes in the parameter values. For more information about the default settings, the reader may refer to the package documentation. Table 2 provides a comprehensive list of all arguments accepted by these functions.

Proj. Pursuit	Clustering	Description of method
`mdh`	`mddc`	Uses hyperplanes with minimal integrated density to separate clusters. Such hyperplanes avoid intersecting high density clusters, as defined in connection with density clustering. The implementation is based on the method of Pavlidis et al. (2016).
`mch`	`mcdc`	Uses hyperplanes which maximise the variance ratio clusterability of the induced binary partition. This approach is closely related to the k-means objective. The implementation is based on that of Hofmeyr and Pavlidis (2015)
`ncuth`	`ncutdc`	Uses hyperplanes with minimum normalised cut measured across them, using the method of Hofmeyr (2017). This leads to partitions with low between and high within cluster similarity.

Table 1: Projection pursuit and clustering algorithms in **PPCI**.

The output of the clustering algorithms (`mddc()`, `mcdc()` and `ncutdc()`) is a named list with class `ppci_clustering_solution`. The most important fields in the output are `$cluster`, which contains the

assigned cluster labels, and `$Nodes` which is itself a list with each entry being a named list containing details of the projection pursuit solution at the corresponding node in the hierarchical model. The output of the projection pursuit algorithms (`mdh()`, `mch()` and `ncuth()`) is a named list with class `ppci_hyperplane_solution`. The most important fields are `$v` and `$b` which give the parameters of the optimal hyperplane, and `$cluster` which gives the binary partition of the data. For further details you can use the `help()` function within the R console, providing the relevant function name as argument, e.g. '`help(mdh)`'.

The base function `plot()` can be used to produce instructive visualisations, like the one shown in Figure 1, which allow the user to investigate the solutions obtained. More detailed plots of individual nodes in the hierarchical model can be obtained by specifying the node number of interest. Based on these inspections, a user may wish to modify their clustering solution by pruning away parts of the model, with the function `tree_prune()`, or adding additional hyperplane separators to further extend the model, using the function `tree_split()`. All of these functions will be discussed in greater detail, primarily through instructive examples which illustrate their use in R.

Argument	Description
`X`	Data matrix ($n \times d$) with observations row-wise.
`K`	(`mddc`,`mcdc` and `ncutdc` only) Number of clusters to extract.
`v0`	(optional) Used to obtain data driven initialisations for projection pursuit. A function accepting a single matrix argument (the data being partitioned) and returning a $d \times m$ matrix (for user chosen m). Each column of the output of `v0` is used as an initialisation. Projection pursuit algorithms (`mdh`,`mch` and `ncuth`) also accept directly the $d \times m$ matrix whose columns are used as initialisations.
`split.index`	(optional. `mddc`,`mcdc` and `ncutdc` only) Used to determine the order in which clusters are split (in decreasing order of split indices). A numeric valued function of the optimal projection vector (`v`), the data matrix (`X`) and parameter list `P`.
`minsize`	(optional) Integer valued minimum cluster size.
`verb`	(optional) Verbosity level. Values greater than zero produce plots to illustrate progress of the algorithm.
`labels`	(optional) Vector of class labels. Only used in plots produced for verbosity levels greater than zero. Does not influence clustering/projection pursuit itself.
`maxit`	(optional) Maximum number of iterations in projection pursuit.
`ftol`	(optional) Relative tolerance level for optimisation.
`bandwidth`	(optional. `mdh` and `mddc` only) Used to determine data driven bandwidth parameter for kernel density estimator. In `mddc` a function taking only a matrix argument (the data being partitioned) and returning a scalar. In `mdh` a scalar to be used directly in kernel estimator.
`alphamin`	(optional. `mdh` and `mddc` only) Initial (scaled) bound on the distance of the optimal hyperplane from the mean of the data being partitioned (α in Eq. (4)).
`alphamax`	(optional. `mdh` and `mddc` only) Final/maximum (scaled) bound on the distance of the optimal hyperplane from the mean of the data being partitioned (α in Eq. (4)).
`s`	(optional. `ncuth` and `ncutdc` only) Used to determine data driven scaling parameter for pairwise similarities. In `ncutdc` a function taking only a matrix argument (the data being partitioned) and returning a scalar. In `ncuth` a scalar to be used directly in similarity calculations.

Table 2: Arguments accepted by clustering and projection pursuit functions.

Projection pursuit, hyperplanes and divisive hierarchical clustering

In this section we give a brief discussion of the general framework we use for finding optimal univariate projections for separating clusters. A binary partition based on a univariate projection is formed by a hyperplane in the original space. We thus approach this problem from the point of view of defining an optimal hyperplane to bi-partition the data, and from this derive associated projection indices. We

then discuss briefly how such hyperplanes are combined to produce a divisive hierarchical clustering model.

Projection pursuit for optimal hyperplane separators

Suppose the data we wish to cluster consists of a finite set of points in $\mathbb{R}^d$, $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$. A hyperplane in $\mathbb{R}^d$ is a $d-1$ dimensional linear surface comprised of all $\mathbf{x} \in \mathbb{R}^d$ which satisfy the same linear equation, of the form $\mathbf{v}^\top \mathbf{x} = b$, where $\mathbf{v} \in \mathbb{R}^d$, $||\mathbf{v}|| = 1$ and $b \in \mathbb{R}$. This leads to a parameterisation of a hyperplane through the terms $\mathbf{v}$ and b, as the set,

$$H(\mathbf{v}, b) = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{v}^\top \mathbf{x} = b \right\}.$$

A hyperplane that intersects the interior of the convex hull of the data induces a binary partition based on which side of the hyperplane each observation lies (the half-space to which each observation is allocated),

$$\begin{aligned} \mathcal{X} &= \mathcal{X}^+_{\mathbf{v},b} \cup \mathcal{X}^-_{\mathbf{v},b}, \\ \mathcal{X}^+_{\mathbf{v},b} &:= \{\mathbf{x} \in \mathcal{X} | \mathbf{v}^\top \mathbf{x} \geqslant b\}, \\ \mathcal{X}^-_{\mathbf{v},b} &:= \{\mathbf{x} \in \mathcal{X} | \mathbf{v}^\top \mathbf{x} < b\}. \end{aligned}$$

To define an optimal hyperplane for separating clusters, we must specify a clustering objective. In **PPCI** we consider three popular objectives, including those underlying density, centroid and graph cut based clustering. However, in principle our approach is agnostic to the clustering objective and for the moment we consider an arbitrary function, Q, which measures the quality of a binary partition. The quality of a hyperplane, $H(\mathbf{v}, b)$, can then be written as,

$$\phi(\mathbf{v}, b | \mathcal{X}) = Q(\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}),$$

where $\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}$ are respectively the univariate projections of $\mathcal{X}^-_{\mathbf{v},b}, \mathcal{X}^+_{\mathbf{v},b}$ onto the vector $\mathbf{v}$. As discussed previously, the structure in the original data may not allow us to distinguish clusters well, perhaps because of noisy or highly correlated subsets of features. In projection pursuit we are interested in how the clusters form within the associated subspace, where here that subspace is defined by the *projection vector* $\mathbf{v}$. If we can find a projection vector, $\mathbf{v}$, for which there exists a b such that $\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}$ and $\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}$ form well distinguished clusters, according to our adopted clustering objective, then this $\mathbf{v}$ should be a high quality projection from the point of view of projection pursuit. This highlights one of the advantages of adopting hyperplane separators; that for a given value of $\mathbf{v}$ the projected data are one-dimensional, and solving the one-dimensional clustering problem is comparatively easy. As a result, for each value of $\mathbf{v}$ we can select the best possible value of b to be our candidate for evaluating the quality of $\mathbf{v}$ as a projection. If we are to consider the alternative of trying to optimise $\phi(\mathbf{v}, b | \mathcal{X})$ directly in terms of $\mathbf{v}$ and b, then we become far more susceptible to the problem of converging to a local optimum. Such locally optimal solutions in $\phi(\mathbf{v}, b | \mathcal{X})$ are particularly prevalent when the data contain many clusters, and different hyperplanes might produce reasonable separations of different subsets of these. If we select only the best value of b for each $\mathbf{v}$, this allows b to change discontinuously during optimisation of the projection vector. This avoids many of the local optima in the joint function $\phi(\mathbf{v}, b | \mathcal{X})$. Formally, the projection pursuit formulation we use is given by,

$$\mathbf{v}_{\text{opt}} = \underset{\mathbf{v} \in \mathbb{R}^d : ||\mathbf{v}|| = 1}{\arg\max} \; \Phi(\mathbf{v} | \mathcal{X}), \tag{1}$$

$$\Phi(\mathbf{v} | \mathcal{X}) = \max_{b \in \mathbb{R}} \phi(\mathbf{v}, b | \mathcal{X}), \tag{2}$$

where the projection index, $\Phi(\mathbf{v} | \mathcal{X})$, measures the quality of the best hyperplane orthogonal to $\mathbf{v}$. Naturally, if the quality measure, Q, is more conveniently stated through a measure of connectedness or overlap between the elements of the binary partition, then Eqs. (1) and (2) simply become minimisations.

It is worth noting that for some linear dimension reduction problems it is important to either whiten the data to ensure the projection pursuit is not unduly affected by scaling issues, or to design the projection index in such a way that it is invariant to scale (Huber, 1985, pg. 6). Our experience is that sensitivity to scaling issues in projection pursuit for clustering is inherited from the objective, Q. Of the methods we consider, only the centroid based objective is particularly sensitive to scaling. The projection index we consider for this objective is naturally scale invariant, and so this sensitivity is not present. We do not advocate data whitening, nor enforcing scale invariance, unless there is such an

inherited sensitivity. We have found that a complete removal of scale can make identifying clusters more challenging in some cases. It also has the potential to inflate the effect of noise dimensions. However, we have found that simply standardising each column of the data matrix separately, by ensuring it has unit variance, is a reliable pre-processing procedure for many clustering problems based on projection pursuit.

An important technical point is that since $\Phi(\mathbf{v}|\mathcal{X})$ is a maximum of $\phi(\mathbf{v}, \cdot|\mathcal{X})$, it is not guaranteed to be continuously differentiable everywhere, even if $\phi(\mathbf{v}, \cdot|\mathcal{X})$ itself is appropriately smooth. However, with the exception of pathological cases in which the elements of $\mathcal{X}$ have a very precise geometry, it is difficult to construct examples in which $\Phi(\mathbf{v}|\mathcal{X})$ is not continuously differentiable almost everywhere. Lewis and Overton (2013) strongly advocate using BFGS to optimise functions such as ours, over more computationally expensive methods like gradient sampling (Burke et al., 2005). We therefore use the existing optimised implementation of BFGS provided in R's base **stats** package. It is important to reiterate that while the projection pursuit formulation we adopt allows the algorithms to avoid many locally optimal solutions in the clustering objective, there is still no guarantee that gradient-based optimisation techniques, such as BFGS, will obtain the globally optimal solution. Especially when the number of clusters is large, the approach we employ may only converge to a local optimum. Local optima arise primarily from the fact that different hyperplanes can lead to different subsets of clusters being separated from the rest, with some such subsets being better separated than others. In Section 2.4.3 we illustrate how modifying the default initialisation for the projection vector can improve the resulting hyperplane separator, when a potentially better partition is apparent.

In the following we describe in greater detail the objectives which underlie the algorithms implemented in **PPCI**, and the associated projection indices used.

Minimum density hyperplanes (`mdh` and `mddc`): In density clustering the data set, $\mathcal{X}$, is assumed to represent a sample of realisations of a random variable $\mathbf{X}$ on $\mathbb{R}^d$, with unknown probability density function p. Broadly speaking clusters are defined as connected regions of high density which surround the modes of p (Comaniciu and Meer, 2002; Hartigan, 1975). A consequence is that these *high density clusters* are separated by connected low density regions. Inevitably the unknown density p is replaced by a data driven estimate $\hat{p}$, which leads to a pleasing interpretation of clusters as connected data dense regions separated by regions of comparative sparsity.

A principled and frequently more practically viable approach to partitioning high density clusters is to focus on the low density separators which pass between them. An optimal hyperplane in this context should pass through only low density regions, and should separate modes of p. The Minimum Density Hyperplane approach (Pavlidis et al., 2016, MDH) seeks to find the hyperplane with minimum total (integrated) density, by solving the problem,

$$\min_{\mathbf{v}} \Phi(\mathbf{v}|\mathcal{X}) = \min_{b\in\mathbb{R}} \phi(\mathbf{v}, b|\mathcal{X}), \tag{3}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) = \hat{I}(\mathbf{v}, b) + C \max\{0, -\alpha\sigma_{\mathbf{v}} - b, b - \alpha\sigma_{\mathbf{v}}\}^{1+\epsilon}, \tag{4}$$

where $\hat{I}(\mathbf{v}, b)$ is an estimate of the integrated density on $H(\mathbf{v}, b)$ and the second term in Eq. (4) is a penalty which effectively constrains the optimal hyperplane to pass within α standard deviations of the mean of the data (assumed here to be equal to $\mathbf{0}$), measured in the direction of $\mathbf{v}$. Notice that the integrated density on the hyperplane $H(\mathbf{v}, b)$ is equal to the density function associated with the one-dimensional random variable $\mathbf{v}^\top\mathbf{X}$, evaluated at b. This means that evaluating $\hat{I}(\mathbf{v}, b)$ requires only evaluating the univariate density estimate from the projected data $\{\mathbf{v}^\top\mathbf{x}_i\}_{i=1}^n$ at the point b. In principle any density estimate can be used, where the MDH method uses a kernel estimate with Gaussian kernels and bandwidth parameter h, so that $\hat{I}(\mathbf{v}, b) = \frac{1}{\sqrt{2\pi}nh}\sum_{i=1}^n \exp\left(-\frac{(\mathbf{v}^\top\mathbf{x}_i - b)^2}{2h^2}\right)$. The value of α above is dynamically adjusted over the range $[\alpha_{\min}, \alpha_{\max}]$ during optimisation, and the best solution which also passes between the modes of the estimated density from the projected data is returned.

Maximum clusterability clustering (`mch` and `mcdc`): The term *clusterability* has been used to refer to the strength, or conclusiveness, of the cluster structure in a data set (Ackerman and Ben-David, 2009). The *variance ratio* clusterability is defined as ratio of the between and within cluster variability (Zhang, 2001), and is related to Fisher's discriminant function used in Linear Discriminant Analysis. It is easy to show that the solution that maximises the variance ratio clusterability also minimises the k-means objective. Determining the clusterability of a data set is therefore of the same complexity as identifying the optimal k-means solution, which is known to be NP-hard. In the univariate setting, however, the k-means solution becomes tractable and so the variance ratio may be used practically to define a projection index. The Maximum Clusterability Divisive Clustering algorithm (Hofmeyr and Pavlidis, 2015, MCDC) uses the variance ratio of the best clustering by a hyperplane orthogonal to $\mathbf{v}$

as projection index, which may be written as,

$$\Phi(\mathbf{v}|\mathcal{X}) = \max_b \phi(\mathbf{v}, b|\mathcal{X}),$$

$$\phi(\mathbf{v}, b|\mathcal{X}) := \frac{\left|\mathcal{X}^+_{\mathbf{v},b}\right| \left(\mu_{\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}} - \mu_{\mathbf{v}^\top \mathcal{X}}\right)^2 + \left|\mathcal{X}^-_{\mathbf{v},b}\right| \left(\mu_{\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}} - \mu_{\mathbf{v}^\top \mathcal{X}}\right)^2}{\frac{n}{n-1} \sum_{i=1}^{n} \left(\mathbf{v}^\top \mathbf{x}_i - \mu_{\mathbf{v}^\top \mathcal{X}}\right)^2 + \sum_{\mathbf{x}_i \in \mathcal{X}^+_{\mathbf{v},b}} \left(\mathbf{v}^\top \mathbf{x}_i - \mu_{\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}}\right)^2 + \sum_{\mathbf{x}_j \in \mathcal{X}^-_{\mathbf{v},b}} \left(\mathbf{v}^\top \mathbf{x}_j - \mu_{\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}}\right)^2},$$

where $\boldsymbol{\mu}_{\mathbf{v}^\top \mathcal{X}}$, $\boldsymbol{\mu}_{\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}}$, and $\boldsymbol{\mu}_{\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}}$ are respectively the means of the data and each of the clusters formed by the hyperplane $H(\mathbf{v}, b)$, when projected onto $\mathbf{v}$. One of the attractive features of the variance ratio as a projection index is that it is scale invariant. This implies that unlike k-means, whose solution is largely determined by directions in which the data exhibit high variability, the MCDC algorithm is capable of separating clusters whose internal covariance structure dominates the overall variability in the data.

Minimum normalised cut hyperplanes (`ncuth` and `ncutdc`): Arising from graph partitioning algorithms, the normalised cut objective has become popular for data clustering (von Luxburg, 2007). The normalised cut (NCut) associated with a partition of $\mathcal{X}$ into clusters $\mathcal{C}_1, ..., C_k$ is given by (Shi and Malik, 2000),

$$\text{NCut}(\mathcal{C}_1, \ldots, \mathcal{C}_k) = \sum_{m=1}^{k} \frac{\text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m)}{\text{volume}(\mathcal{C}_m)}, \tag{5}$$

$$\text{Cut}(\mathcal{C}, \mathcal{X} \setminus \mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C}, \\ \mathbf{x}_j \notin \mathcal{C}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j), \quad \text{volume}(\mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C} \\ \mathbf{x}_j \in \mathcal{X}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j). \tag{6}$$

By minimising the normalised cut one simultaneously attempts to minimise between cluster similarity and maximise within cluster similarity. Similarities are typically defined via a function of the distances between pairs of points, i.e., $\text{similarity}(\mathbf{x}_i, \mathbf{x}_j) = k(\|\mathbf{x}_i - \mathbf{x}_j\|)$, where $k : \mathbb{R}^+ \to \mathbb{R}^+$ is a decreasing function. When computing similarities between the projected data $\{\mathbf{v}^\top \mathbf{x}_i\}_{i=1}^n$, if the Laplace kernel, $k(x) = \exp(-|x|/\sigma)$, is used then the optimal NCut solution by a hyperplane orthogonal to $\mathbf{v}$ can be determined in $\mathcal{O}(n \log n)$ time (Hofmeyr, 2017). This is a substantial improvement over the $\mathcal{O}(n^2)$ spectral relaxation of the NP-hard NCut problem in the original space. This means that the optimal normalised cut can be used as a projection index within an efficient clustering algorithm. The projection index follows directly from the formulation of the NCut, as

$$\Phi(\mathbf{v}|\mathcal{X}) = \min_b \phi(\mathbf{v}, b|\mathcal{X}), \tag{7}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) := \text{NCut}(\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}). \tag{8}$$

It is important to note that in (Hofmeyr, 2017) the divisive clustering algorithm based on minimum normalised cut hyperplanes is called NCutH. To be consistent with the naming conventions used in this paper we call the divisive algorithm NCUTDC while NCUTH refers to the projection pursuit for a single hyperplane.

Divisive hierarchical clustering using hyperplanes

A divisive hierarchical clustering model is constructed by recursively applying a partitioning method to (subsets of) the data set. Using hyperplanes to recursively bi-partition the data yields a cluster hierarchy with a binary tree structure. The leaves of the tree correspond to the final clusters, with each defined as the intersection of a finite number of half spaces. If the number of clusters is specified then the only requirements for constructing a divisive hierarchical clustering model are the cluster quality measure Q, which leads to the projection index Φ, and an indexing (or selection) strategy that determines which leaf (cluster) of the current hierarchy should be partitioned at each step of the recursion. Examples of indexing strategies include splitting the largest cluster, $i = \arg\max_{j \in \text{leaves}(T)} \{|\mathcal{C}_j|\}$, where T is the cluster hierarchy and $\mathcal{C}_j \subset \mathcal{X}$ is the data subset allocated to leaf j, or splitting the leaf whose bi-partition achieves the maximum clusterability, $i = \arg\max_{j \in \text{leaves}(T)} \Phi(\mathbf{v}_{\text{opt}}|\mathcal{C}_j)$.

A general comment on choosing between the methods in PPCI

In this section we discussed three objectives which can be used to obtain optimal projections for clustering using projection pursuit. While in many cases all three methods will lead to similar

clustering solutions, they can be differentiated by their underlying clustering objectives. Being motivated by the ratio of between to within cluster variances, and its inherent similarity with the k-means objective, the Maximum Clusterability Divisive Clustering method is appropriate for obtaining strong discrimination between the core of clusters which lie close to their means. It tends to lead to subspaces in which the cores of clusters are very compact and well separated. It's limitations are its greater susceptibility to outliers than either of the other two methods included in **PPCI**, and the fact that points lying close to the boundaries between clusters may be less well discriminated. On the other hand, Minimum Density Hyperplanes focus precisely on the boundaries between clusters, seeking to obtain very low density, and therefore sparse regions to separate clusters. It is more robust to outliers than either of the other two methods, but has a higher overall failure rate when there is a high degree of cluster overlap and hence the density between clusters is not particularly low. Finally, Normalised Cut Hyperplanes may be seen as lying somewhere between these. It has been shown that the normalised cut is closely related to the idea of low-density separation, while simultaneously also focusing on within cluster compactness (Trillos et al., 2016; Hofmeyr, 2017, 2019).

Users who are aware of the types of clusters in which they are interested will hopefully find the above discussion useful in guiding their application. Alternatively, the user may consider applying all three approaches to obtain multiple clustering solutions. These can then each be inspected separately through the instructive visualisation tools offered in the package. On the other hand, if the user prefers not to have the dimension reduction guided by a specific clustering objective, then generic dimension reduction and visualisation methods may be more appropriate.

Applying PPCI

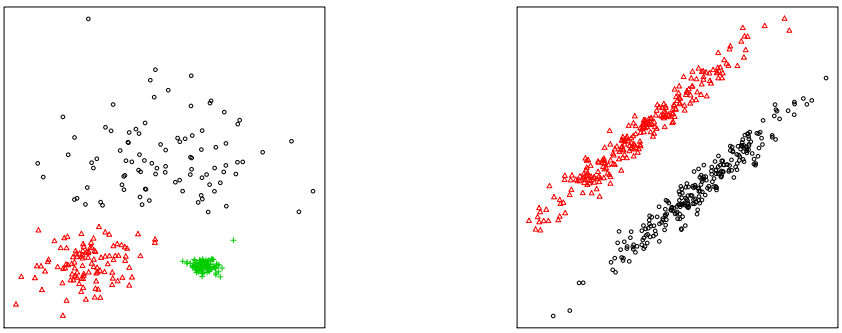
In this section we illustrate the practical application of the main methods provided in the **PPCI** package within R. We first use a few two-dimensional toy examples to illustrate some of the advantages and limitations of the clustering models obtained by the methods presented in **PPCI**. Although the clustering methods in **PPCI** are motivated by problems of at least moderate dimensionality, these two dimensional examples are intended to provide interpretable illustrations of the types of clusters which these methods can, and cannot identify. Next we apply the main clustering algorithms in their default formulation to a popular real image data set taken from the UCI machine learning repository (Lichman, 2013). We go on to discuss in detail how instructive visualisations offered in **PPCI** can be used to validate and, if necessary, modify a clustering solution. Finally we discuss briefly a few extensions to the standard methods in the package.

Two-dimensional toy data sets

One of the advantages offered by divisive clustering algorithms, is that when clustering is performed repeatedly on subsets of the data, each such clustering task adapts to the scale of the corresponding data subset. As a result, such approaches are naturally equipped to handle data with clusters defined at varying scales. A simple example is seen in Figure 2(a). To ensure exact reproducibility we first set the seed, before creating the data set. We then apply the `mddc()` function to obtain three clusters, and plot the result.

```
> set.seed(1)
> means <- c(1, 0, -1, 0, 0, sqrt(2))*runif(6)
> X1 <- matrix(means, 300, 2, byrow = T) + .05*rnorm(600)*(1:3)^2
>
> sol1 <- mddc(X1, 3)
>
> plot(X1, col = sol1$cluster, pch = sol1$cluster, xlab = '', ylab = '',
        xaxt = 'n', yaxt = 'n')
```

Figure 2(b) shows a simple case containing very elongated clusters. In such cases the directions of high variability do not contribute to the separation of clusters. Rather the within cluster variability dominates the between cluster separation. In cases like this off-the-shelf dimension reduction methods like PCA will fail to find a good separation of clusters. Indeed in this case the first principal direction does not lead to separation of the clusters at all. Using projection indices which incorporate the clustering objective can lead to far superior results. As there are only two clusters, these can be clustered by a single hyperplane, and we apply the function `mdh()` to the data to obtain the result.



(a) Clusters of different scale (b) Elongated clusters

Figure 2: Simple data sets illustrating clustering problems for which the methods in **PPCI** are most appropriate.

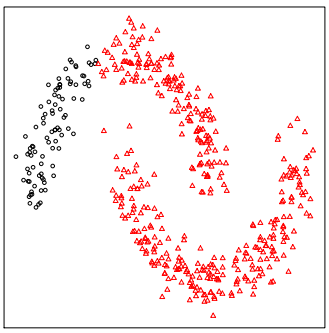
```
> set.seed(1)
> S <- matrix(c(1, .8, .8, 1), 2, 2)
> X2 <- matrix(rnorm(1000), ncol= 2)%*%S
                            + cbind(rep(c(.8,-.8),each=250), rep(c(-.8,.8),each=250))
>
> sol2 <- mdh(X2)
>
> plot(X2, col = sol2$cluster, pch = sol2$cluster, xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
```

Arguably the main limitation of the methods included in **PPCI** is the restriction to linearly separable clusters. That is, clusters which can be accurately separated using hyperplanes. Figure 3 shows a classic example taken from (Jain and Law, 2005). The data set contains two clusters, each in the shape of a half moon, arranged so that they cannot be separated by a hyperplane. The methods in **PPCI** are not appropriate for such problems. Cases such as this are important as there may be apparent cluster structure in the optimal univariate subspace. This can be seen in Figure 3(b), which includes a kernel estimate of the density from the data projected into the optimal subspace. There is evidence from the density plot of multiple modes, which may be interpreted as indicating clusters. However, no partition based only on this univariate representation can accurately separate the clusters. The visualisation tools offered by **PPCI** all include bivariate scatter plots in addition to the density plots from the optimal univariate subspace. These can be used to help identify the occurrence of such problems, but cannot completely eliminate them. We discuss the visualisation methods included in **PPCI** in detail in Section 2.4.3.

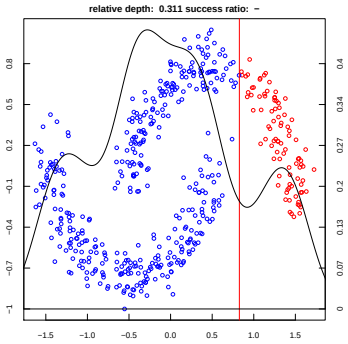
```
> set.seed(1)
> th <- runif(500)*pi + rep(c(pi,0),each=250)
> X3 <- cbind(cos(th)+rep(c(.5,-.5),each=250), sin(th)+rep(c(.2,-.2),each=250))
> X3 <- X3 + .1*rnorm(1000)
>
> sol3 <- mdh(X3)
>
> plot(X3, col = sol3$cluster, pch = sol3$cluster, xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
>
> plot(sol3)
```

Note that it is not the shape of the clusters themselves which makes the methods inappropriate, but rather their arrangement. In particular, if the convex hulls of the clusters overlap then it is not possible to accurately separate them using hyperplanes. In Figure 4 we show the same clusters as in Figure 3, but arranged so that their convex hulls do not overlap, and hence they can be separated by a hyperplane. In this case the methods in **PPCI** are capable of separating the clusters. Notice also that in such cases the univariate density plot from the optimal subspace will tend to show stronger bimodality than in cases where the clusters are not being separated by a hyperplane.

```
> set.seed(1)
> th <- runif(500)*pi + rep(c(pi,0),each=250)
> X4 <- cbind(cos(th)+rep(c(.5,-.5),each=250), sin(th)-rep(c(.3,-.2),each=250))
```

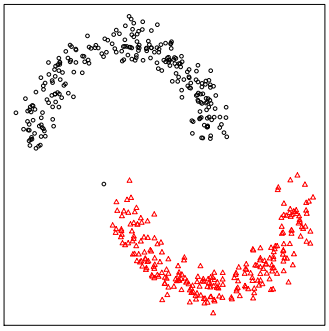


(a) Non-linear clusters with overlapping convex hulls

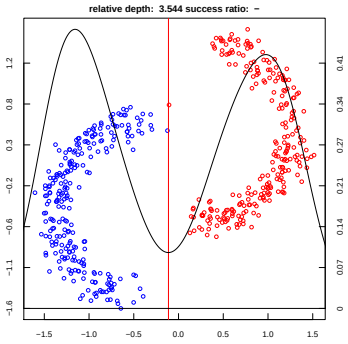


(b) Univariate density plot from the optimal projection of the data based on the intergrated density on the hyperplane.

Figure 3: Non-linear clusters with overlapping convex hulls , which cannot be separabed by hyperplanes. Despite this, the univariate density still shows evidence of multiple modes.



(a) Non-linear clusters with disjoint convex hulls



(b) Univariate density plot from the optimal projection of the data based on the intergrated density on the hyperplane.

Figure 4: Non-linear clusters with non-overlapping convex hulls, which can be separated by a hyperplane. In this case the multi-modality of the univariate density is much stronger.

```
> X4 <- X4 + .1*rnorm(1000)
>
> sol4 <- mdh(X4)
>
> plot(X4, col = sol4$cluster, pch = sol4$cluster, xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
>
> plot(sol4)
```

In a later example we briefly discuss how data containing clusters which are not linearly separable can first be transformed using a non-linear transformation so that the methods in **PPCI** can be made better equipped to cluster them.

Divisive hierarchical clustering of image data

We will illustrate the use the projection pursuit based clustering algorithms on a popular example from image clustering, including handwritten images of numerical digits. To cluster image data each image is first vectorised by stacking either the rows or columns of the matrix of pixel intensities. Image data are frequently characterised by one or more of (i) high dimensionality, (ii) presence of outliers, (iii) correlated features, (iv) noise dimensions, (v) elongated cluster shapes. This example was selected as it allows us to illustrate the effect which (some of) these challenging characteristics can have on a clustering algorithm, and at the same time how projection pursuit based methods can be made well

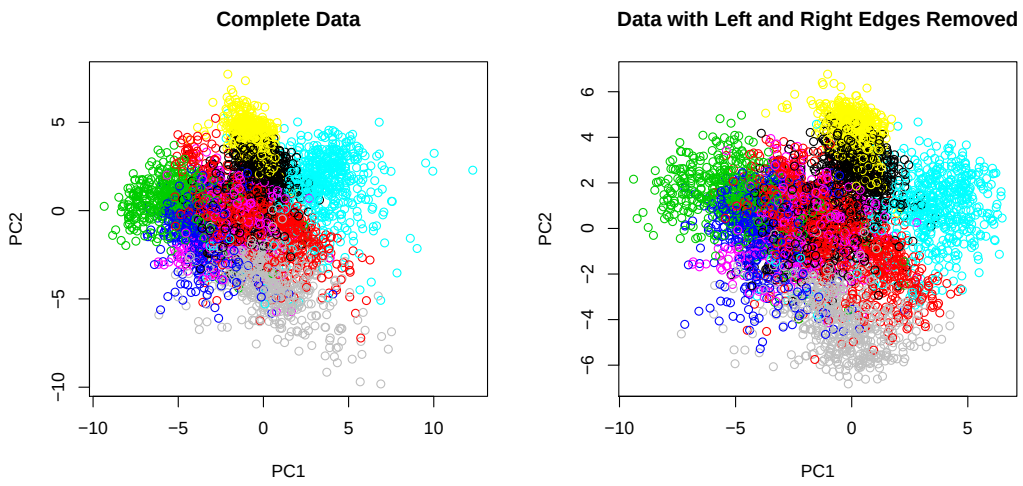


Figure 5: Principal component projections of optidigits data set. Complete data set (left) and with removal of left and right edges (right)

suited to handling them.

Example 1 In this example we consider the optical recognition of handwritten digits data set obtained from the UCI machine learning repository (Lichman, 2013). Each image contains a handwritten instance of one of the digits 0–9, resulting in ten clusters. The images have been compressed to 8×8 so that, once vectorised, the observations contain 64 dimensions. This data set is characterised by the presence of noise dimensions and outliers. Many of the noise dimensions correspond to pixels on the left and right edges of the images. In addition, many of the images which may be considered outliers from their clusters are so precisely because the digit extends to the left or right edge. There is thus a relationship between these two challenging characteristics.

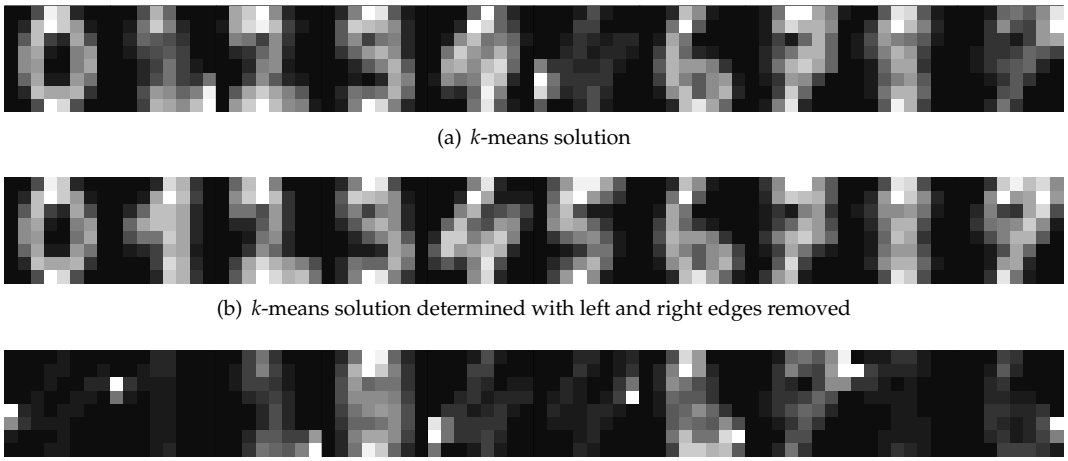
We first visualise the data in their vector form using principal component projections to illustrate the presence and potential effect of outliers. We also provide the same visualisation but with the removal of the dimensions corresponding to the left and right edges of the images. These visualisations are shown in Figure 5.

```
> data(optidigits)
> par(mfrow = c(1, 2))
> plot(optidigits$x%*%eigen(cov(optidigits$x))$vectors, col = optidigits$c+1,
    xlab = 'PC1', ylab = 'PC2', main = 'Complete Data')
> edges <- c((1:8)*8-7, (1:8)*8)
> plot(optidigits$x[,-edges]%*%eigen(cov(optidigits$x[,-edges]))$vectors,
    col = optidigits$c+1, xlab = 'PC1', ylab = 'PC2',
    main = 'Data with Left and Right Edges Removed')
```

The main structure of the data in their principal components is similar in both cases, however there is clear evidence of points lying far from the bulk of the data which are not present with the edges removed. Furthermore the separation of individual clusters is more substantial in the case where edges have been removed. Although the number of outliers is relatively few, their influence on a clustering solution can be substantial. To illustrate the problematic nature of these outliers and noise dimensions for clustering, we begin by applying the popular k-means algorithm to the data[1]. We also apply k-means separately to the left and right edges and to the middle six columns of the images (i.e. with left and right edges removed). We use the same seed for random number generation in each case to ensure the points used as initial centroids are the same in all cases. We then plot the means of the extracted clusters, recast as 8×8 images, using the function `optidigits_mean_images()`. This function takes a single vector of cluster assignments and produces an image representing the cluster means. The outputs of this function are shown in Figure 6.

```
> set.seed(1)
> km_sol <- kmeans(optidigits$x, 10, nstart = 10)
```

[1]We also considered spectral clustering and multiple agglomerative hierarchical clustering algorithms, all of which rendered sub-optimal performance on the whole data set. In the interest of brevity we report only the results from k-means.



(a) *k*-means solution

(b) *k*-means solution determined with left and right edges removed

(c) *k*-means solution determined using only left and right edges of images

Figure 6: Cluster means of optidigits data from *k*-means

```
> set.seed(1)
> km_sol_edges <- kmeans(optidigits$x[,edges], 10, nstart = 10)
> set.seed(1)
> km_sol_middle <- kmeans(optidigits$x[,-edges], 10, nstart = 10)
>
> optidigits_mean_images(km_sol$cluster)
> optidigits_mean_images(km_sol_middle$cluster)
> optidigits_mean_images(km_sol_edges$cluster)
```

The *k*-means solution from the complete images, Figure 6(a), shows two clusters (where digits 5 and 9 should be) which are characterised primarily by a few edge pixels. This can be seen by the presence of bright edge pixels with the remainder of the image being indistinct as a result of averaging images containing multiple different digits. The cluster for the digit 1 is also less distinct than many others and appears to be a combination of 1's and 2's. Similarly the cluster for digit 3 is a combination of 3's and 9's. There are no well defined clusters for digits 5 and 9. By removing the left and right edges, Figure 6(b), the result is substantially improved. Each digit is well distinguished in its own cluster, with the possible exception of the digit 3 which here contains a mix of digits 3 and 9. The clusters determined using only the edge pixels, Figure 6(c), show how the edge pixels can be misleading for distinguishing digits, and that many of the inaccuracies from the *k*-means solution could be caused by the edges containing information which is counter-informative to the true clusters in the data. The majority of the clusters from the edges are determined by images which happen to share common edge pixels. For the most part these do not contribute to the identification of different digits.

This ad-hoc investigation allowed us to identify potential causes for why accurate clustering of the optidigits data set is challenging, and to obtain a modest improvement by a manual reduction of dimensions. However, this required the ability to visualise the cluster means as images, and also we had domain specific knowledge (that the images are of the digits 0–9) which will not be available in many real world applications. Naturally it is preferable if these challenges can be circumvented automatically by the clustering algorithm. We next apply the clustering algorithms implemented in **PPCI**, and similarly investigate the resulting cluster means. Note that both `mddc()` and `ncutdc()` in their default implementations do not contain any random components, however `mcdc()` uses *k*-means (with ten restarts) to initialise the projection pursuit. The output of `mcdc()` (and similarly `mch()`) is thus subject to variation. Although this variation has not been explicitly quantified, our experience is that it is considerably less than in the output of the standard *k*-means algorithm itself. Still, to ensure exact reproducibility we set the seed as before.

```
> set.seed(1)
>
> mddc_sol <- mddc(optidigits$x, 10)
> ncutdc_sol <- ncutdc(optidigits$x, 10)
> mcdc_sol <- mcdc(optidigits$x, 10)
>
> optidigits_mean_images(mddc_sol$cluster)
> optidigits_mean_images(ncutdc_sol$cluster)
> optidigits_mean_images(mcdc_sol$cluster)
```

(a) MDDC solution

(b) NCUTDC solution

(c) MCDC solution

Figure 7: Cluster means of optidigits data from **PPCI** algorithms

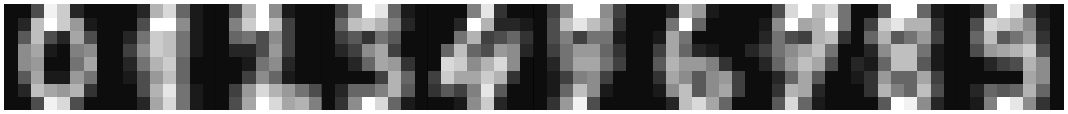


Figure 8: Cluster means of optidigits data from MCDC algorithm, with minimum cluster size set to 50

Clustering algorithms based on projection pursuit should in general be robust to the presence of noise dimensions. In addition the MDDC algorithm focuses on finding low density hyperplanes which pass between clusters, and so is not very susceptible to outliers. Similarly, the NCUTDC algorithm finds solutions with low between cluster similarity and is fairly robust to the presence of outliers. We expect these two methods to work well on these types of data. On the other hand, the *k*-means objective is susceptible to such outliers, and the MCDC algorithm may not be able to accurately identify all clusters. Figure 7 shows the cluster means produced using the clustering algorithms in **PPCI**. Both MDDC and NCUTDC have produced solutions of substantially better quality than *k*-means. With the exception of 8, each digit is well distinguished from the others. On the other hand MCDC has assigned one cluster to images which coincide in a few edge pixels, and so lacks a cluster assigned to the digit 5. Images of the digits 3 and 9 have also been combined in a single cluster (in place of cluster 3). The remaining images of digit 9 represent a different writing style.

All clustering and projection pursuit algorithms implemented in **PPCI** offer a simple means for mitigating the effect of outliers. This is achieved through the optional argument `minsize`, which specifies the minimum number of points which can be assigned to a cluster. Since this constraint is applied during optimisation, the subspaces obtained through projection pursuit are suitable for separating clusters of at least `minsize` points. We use this simple modification in the MCDC algorithm by setting the mininmum cluster size to 50, and again visualise the resulting cluster means as images. In the next subsection we discuss in more detail the modification and validation of clustering models.

```
> set.seed(1)
> mcdc_sol <- mcdc(optidigits$x, 10, minsize = 50)
>
> optidigits_mean_images(mcdc_sol$cluster)
```

The results are shown in Figure 8, where the solution is substantially improved. The digits 5 and 7 have not been separated well from one another, however all other clusters are clearly separated and all digits are represented in their own clusters.

In this example we saw how clustering algorithms using projection pursuit are capable of obtaining high quality clustering results in the presence of noise dimensions and outliers, without any need for domain knowledge or specialised visualisations. We assessed the performance in relation to the cluster means and their representation as images. A very simple and intuitive modification to the input parameters allowed us to substantially improve upon the default solution for the MCDC algorithm, based on its observed susceptibility to outliers, which it inherits from the *k*-means objective. In the next example we discuss in detail a more structured approach to validating and further improving clustering solutions using the projection pursuit framework provided in **PPCI**.

Modifying and validating a clustering solution using instructive visualisations

If the data are assumed to arise from a mixture of simple parametric distributions, then well established model selection methods can be used to validate a clustering model. In the absence of such strong assumptions, however, the validation of clustering models remains a challenging problem. Low-dimensional visualisations that reveal the cluster structure present in multivariate data can therefore be critical in determining an appropriate model. In this section we discuss how clustering models obtained through one of the hierarchical algorithms implemented in **PPCI** can be validated through visualisations, and modified interactively. The projection pursuit algorithms discussed in this paper identify vectors that maximise the clusterability of a data set, and are thus natural candidates for creating such visualisations. Because of the hierarchical structure of the clustering models obtained, modification of a part of the model does not require a complete reconstruction of the solution. In fact two simple operations are sufficient to obtain an extremely versatile framework for model modification. These may be referred to as *pruning* and *extending*.

- Pruning of a hierarchical model refers to the removal of a sub-hierarchy rooted at a specific node in the model. Pruning may be desirable if it is evident that part of the hierarchical model does not contribute to the identification of additional clusters. In addition, it may be that an internal node of the model contains a suboptimal splitting of the data assigned to it. The entire model can thus be improved by first pruning the sub-hierarchy rooted at the defecting node, and then rebuilding this sub-hierarchy by varying the parameters used in the projection pursuit/clustering, thereby improving its quality. A subtree may be removed using the function `tree_prune()`.
- It will be desirable to extend a hierarchical model when it is apparent that a leaf node contains more than one cluster. Extension of a model is performed incrementally, splitting one leaf at a time, using the function `tree_split()`.

The arguments accepted by the functions `tree_prune()` and `tree_split()` are shown in Table 3. Both

Argument	Description
`sol`	A clustering solution arising from one of `mddc`, `mcdc` or `ncutdc`.
`node`	In `tree_prune` the node at which to prune the clustering model. In `tree_split` the node at which to extend the model by further splitting the data at the node. Only leaf nodes may be split, and an error will be thrown if a non-leaf is specified.
`...`	(optional. `tree_split` only) Any collection of optional arguments associated with the clustering algorithm which produced the solution `sol`.

Table 3: Arguments accepted by `tree_prune()` and `tree_split()` functions.

functions return an object of the same type as the original solution. The function `tree_split()` accepts all optional arguments associated with the clustering algorithm which produced the solution `sol`. This allows the user to refine the solution at the node by modifying the way in which the projection pursuit is conducted.

In this framework an initial hierarchical model can be built using an estimate of the number of clusters in the data. One may then first prune away parts of the model which result in individual clusters being over-partitioned, or where the partition at an internal node appears to produce a suboptimal split of the clusters. Any leaf nodes which appear to contain more than one cluster can then be recursively split. If necessary, this process can be iterated, until the model is deemed valid. A solution may be seen to be valid if there are no internal nodes at which a single cluster is divided by the binary partition at that node, and there are no leaf nodes which contain multiple clusters. We will illustrate the modification of a clustering hierarchy by means of a detailed example.

Example 2 We revisit the optical recognition of handwritten digits data set (Lichman, 2013). For illustrative purposes, we do not assume as we did previously that the number of clusters is known. Nor do we assume any domain specific knowledge, which in the prior example allowed us to visualise the clusters as images. We initially partition the data set into seven clusters using minimum density hyperplanes. The resulting hierarchical clustering model is visualised through the `plot.ppci_cluster_solution()` function, which is accessed by applying the base `plot()` function to any clustering solution obtained using **PPCI**. This visualisation is shown in Figure 9.

```
> data(optidigits)
> sol <- mddc(optidigits$x, 7)
> plot(sol)
```

Argument	Description
`sol`	A clustering solution from any clustering algorithm in **PPCI**.
`node`	(optional) If a detailed visualisation of a single node is desired, instead of the complete model, this is achieved by specifying the node number (see argument `node.numbers` below).
`labels`	(optional) A vector of class labels. Points with the same label are plotted with the same colour.
`node.numbers`	(optional) Logical. If `node.numbers==TRUE` then each node is given a number, based on the order in which they were added to the hierarchy, and indicated in the plot.
`transparency`	(optional) if omitted then points in scatter plots are shown as opaque. If a value in (0, 1) is provided then points are shown as transparent, with smaller values corresponding to a greater degree of transparency. Using transparency can be useful in capturing densities, as multiple overlapping transparent points (which occur in data dense regions) appear darker.

Table 4: Arguments accepted by `plot.ppci_cluster_solution()` function.

Each scatterplot in the output of `plot.ppci_cluster_solution()` depicts the data subset assigned to the corresponding node in the hierarchy projected into a two-dimensional subspace. The horizontal axis corresponds to the optimal projection vector, $\mathbf{v}_{\text{opt}}$ in Eq. (1), while the vertical axis is the direction of maximum variance orthogonal to $\mathbf{v}_{\text{opt}}$. Colours indicate the binary partitions induced by the corresponding hyperplane separator. Since leaf nodes correspond to clusters the projected data are assigned a single colour.

Details of the arguments accepted by the function `plot.ppci_cluster_solution()` are given in Table 4. The first thing to notice in Figure 9 is the partition at the node numbered 4. There is a clear

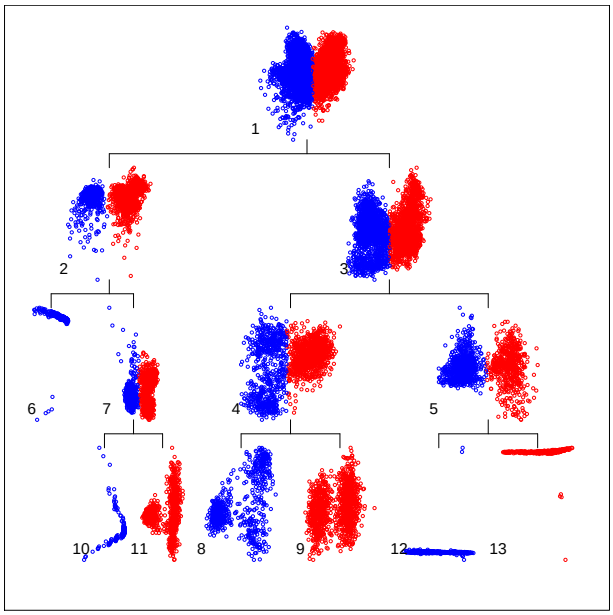
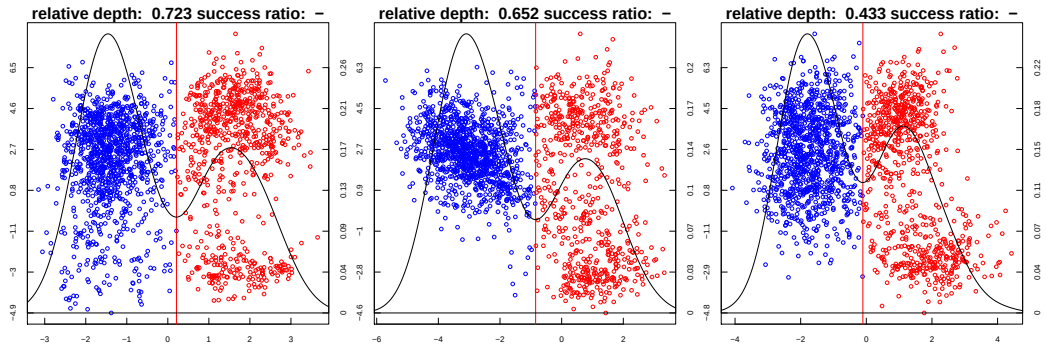


Figure 9: Initial clustering of optidigits data set through MDDC with 7 clusters.

presence of multiple clusters, however the separating hyperplane appears to pass through regions of relatively high density, and may not be optimally separating the clusters. We therefore investigate the possibility of improving the partition at this node. This can be achieved by modifying the parameters used to obtain the partition at this node, and include any of the optional parameters accepted by the function which produced the current solution, in this case `mddc()`. In the previous example we saw that one of the clusters consisted of only a few outlying points, and as a result the way was clear in terms of a sensible modification to improve the solution. When no immediate cause for the apparently sub-optimal solution is obvious, we find that considering alternative initialisations for the projection pursuit frequently presents improved solutions. The default initialisation which gave rise to this projection was the first principal component of the data assigned to the node.



(a) Best solution (b) First alternative (c) Second alternative

Figure 10: Three potential binary partitions for node 4 based on three different initialisations.

We isolate the data assigned to this node to see if a better solution can be obtained by considering additional initialisations (we now consider the first three principal components). Each node in the model is stored as a list containing the field `$ixs` which specifies the indices of the observations allocated to the node. We use the optional argument `v0` to investigate alternative initialisations. This argument must be a function which takes as its only argument a data matrix, and returns a matrix, each column of which is considered as an initialisation. The number of rows of the output of `v0` must therefore match the number of columns of its input. When multiple initialisations are considered within one of the projection pursuit algorithms, the best solution is returned. In the case of `mdh()` the solution with the greatest relative depth is returned, as recommended by Pavlidis et al. (2016), whereas in `mch()` and `ncuth()` the best solution is determined by the value of the objective function. The solutions from the other initialisations are stored in the field `$alternatives`. The usefulness of having available the alternative options is that, despite clustering objectives being consistent with our intuition for how clusters should be defined, it is possible that the objective may not adequately capture all potential nuances in the data. A visual comparison, made available by the function `plot.ppci_hyperplane_solution()`, could lead to a preferable solution despite its being less optimal in terms of the objective function. We therefore inspect all potential solutions obtained for this node. The `plot.ppci_hyperplane_solution()` function is accessed by applying the base `plot()` function to the output of any of the projection pursuit algorithms implemented in **PPCI**.

```
> node4_x <- optidigits$x[sol$Nodes[[4]]$ixs,]
> v0 <- function(X) rARPACK::eigs_sym(cov(X), 3)$vectors
> node4_alt <- mdh(node4_x, v0 = v0)
> plot(node4_alt)
> plot(node4_alt$alternatives[[1]])
> plot(node4_alt$alternatives[[2]])
```

The outputs can be seen in Figure 10. The current solution is shown in Figure 10(b), as the first alternative in this case. The proposed best solution, Figure 10(a), also appears visually superior to the current solution as it does not pass through any relatively high density regions. The clustering solution is thus first pruned at node 4, and then extended with a new initialisation. The function `tree_prune()` removes the sub-hierarchy rooted at a specific node. The `tree_split()` function can then be used to extend a clustering model by splitting a leaf node further. In the example below we set the initial projection vector to the best solution obtained above, by using the optional argument `v0`. So as not to deviate from this solution as a result of the penalty term in the MDH formulation, the initial value of α is set to the value from the best solution above. This is achieved through the optional argument `alphamin`. It is important to note that the numbers assigned to the nodes in a model reflect the order in which they were added to the model. As such, pruning a model may alter the numbers of multiple nodes, including potentially those on different branches in the hierarchy. The modified clustering solution is shown in Figure 11.

```
> sol <- tree_prune(sol, 4)
> v0 <- function(X) matrix(node4_alt$v, ncol = 1)
> alpha <- node4_alt$params$alpha
> sol <- tree_split(sol, 4, v0 = v0, alphamin = alpha)
> plot(sol)
```

At this stage the internal nodes appear to produce satisfactory partitions of their data. However, there is evidence of multiple leaf nodes which each contain more than one cluster. A single node can be

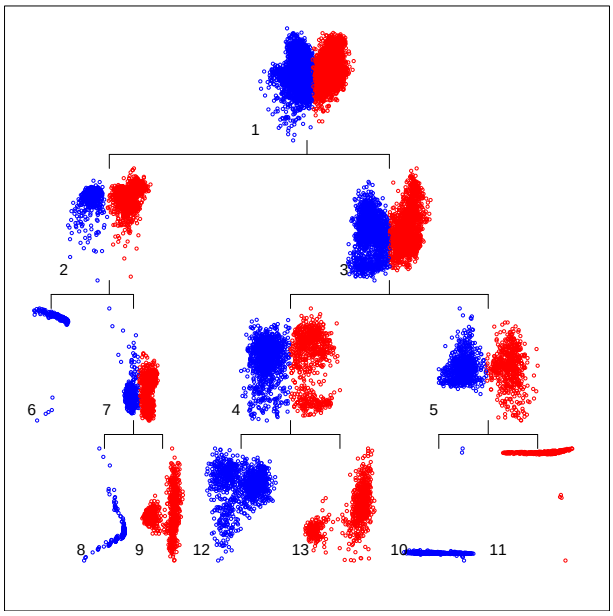
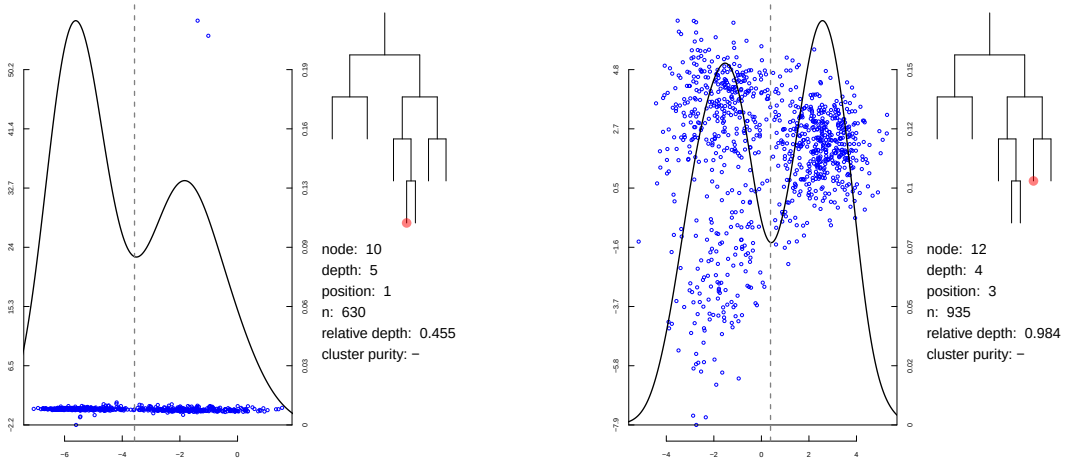


Figure 11: Clustering solution after the modification of the partition at node 4.



(a) Visualisation of node 10

(b) Visualisation of node 12

Figure 12: Visualisations of individual nodes in cluster hierarchy.

closely inspected using the function `plot.ppci_cluster_solution()` by specifying the node which is to be inspected. Again this function is accessed via the base `plot()` function. This produces an output similar to the `plot.ppci_hyperplane_solution()` function, but includes information about the position of the node within the hierarchical clustering model as well. All leaf nodes can therefore be inspected separately. Examples including nodes numbered 10 and 12 are seen in Figure 12.

```
> plot(sol, node = 10)
> plot(sol, node = 12)
```

Both nodes show strong evidence of multiple clusters, through their highly bimodal projected densities. Moreover, the potential partition at the leaf node numbered 12 does not appear to offer an optimal partition of the clusters. Alternative initialisations can be considered as above for node 4, leading to the conclusion that initialisation on the second principal component produces a better partition. The importance of closer inspection of individual nodes is also apparent in the case of node 10. The high variance projection orthogonal to $\mathbf{v}_{\text{opt}}$, depicted in the vertical axis in Figure 12(a), is heavily influenced by a few outlying points. This makes the two dimensional plot less relevant for visualising the cluster structure in the data. After inspecting all leaves, the nodes numbered 9, 10, 12 and 13 are further partitioned, with node 12 using the alternative initialisation discussed above.

```
> sol <- tree_split(sol, 9)
> sol <- tree_split(sol, 10)
```

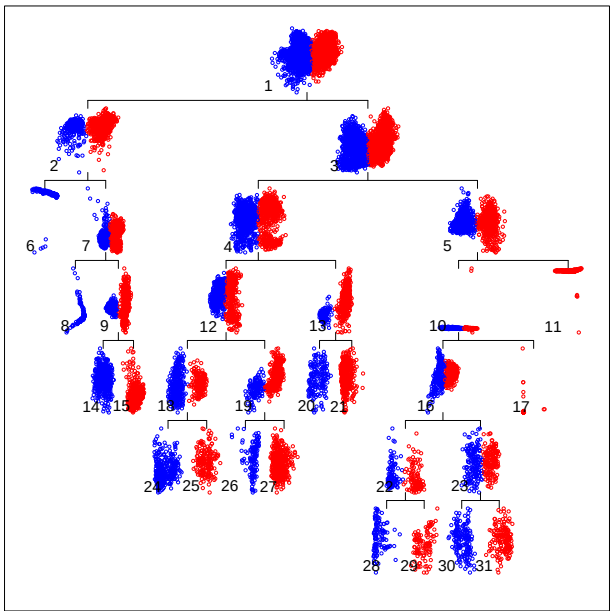


Figure 13: Final clustering solution after modifications. There is no evidence of individual clusters being split, and no evidence of clusters not identified in the model.

```
> node12_x <- optidigits$x[sol$Nodes[[12]]$ixs,]
> v0 <- function(X) rARPACK::eigs_sym(cov(X), 3)$vectors
> node12_alt <- mdh(node12_x, v0 = v0)
> v0 <- function(X) matrix(node12_alt$v, ncol = 1)
> alpha <- node12_alt$params$alpha
> sol <- tree_split(sol, 12, v0 = v0, alphamin = alpha)
> sol <- tree_split(sol, 13)
```

Inspecting the newly produced leaf nodes shows that leaves numbered 16, 18 and 19 likely contain multiple clusters, and so are further partitioned.

```
> sol <- tree_split(sol, 16)
> sol <- tree_split(sol, 18)
> sol <- tree_split(sol, 19)
```

Once again the resulting leaf nodes are inspected, and nodes 22 and 23 are partitioned.

```
> sol <- tree_split(sol, 22)
> sol <- tree_split(sol, 23)
```

The complete solution at this stage can be seen in Figure 13. The solution is valid in the sense that there appear to be no internal nodes at which a single cluster is split, and there are no leaf nodes which show evidence of multiple clusters. Finally, we compare the performance of this model with the solution assuming the number of clusters is known. We evaluate the performance using four popular external cluster evaluation metrics, namely purity (Zhao and Karypis, 2004), normalised mutual information (NMI) (Strehl and Ghosh, 2002), adjusted Rand index (Hubert and Arabie, 1985), and V-measure (Rosenberg and Hirschberg, 2007). The `cluster_performance(clusters,labels)` function computes these metrics from a vector of cluster assignments, `clusters`, and a vector of true cluster labels, `labels`.

```
> sol2 <- mddc(optidigits$x, 10)
> cluster_performance(sol$cluster, optidigits$c)
 adj.rand    purity v.measure       nmi
0.7150910 0.8989324 0.7641322 0.7656866
> cluster_performance(sol2$cluster, optidigits$c)
 adj.rand    purity v.measure       nmi
0.6451702 0.7919929 0.7202152 0.7202187
```

The solution obtained by means of modifying the initial model overestimates the number of clusters by 60%. Despite this it substantially improves the overall agreement between the cluster assignment and the true class labels when compared with the solution assuming the number of clusters is known. We were able to improve the solution substantially without assuming any domain specific knowledge,

and using only an intuitive interpretation of how clusters should appear within low dimensional visualisations.

Extensions

Dimension reduction: Although we prefer the approach of obtaining a divisive hierarchical clustering model through recursive univariate projection, it is interesting to consider using the projection indices in **PPCI** for the purpose of obtaining a single multivariate projection of a data set. For this purpose we provide the functions `mddr()`, `mcdr()` and `ncutdr()`; with the common suffix `dr` denoting "dimension reduction". These functions take only two mandatory arguments, the data matrix, `X`, and the number of dimensions of the projection, `p`. As is the case with the clustering and projection pursuit functions, optional arguments allow the user to modify the parameters used in projection pursuit. To obtain a multivariate projection, we apply the standard approach of recursively obtaining a univariate projection using the data projected into the null space of the projections determined so far (Huber, 1985). The output of `mddr()`, `mcdr()` and `ncutdr()` is a named list of class `ppci_projection_solution`. The most important fields are `$projection`, the projection matrix, and `$fitted`, the projected data.

To evaluate the quality of dimension reduction for clustering, we can use internal cluster validity metrics applied to the true clusters. That is, if the true clusters display a high degree of validity within the reduced space then this indicates that the reduced representation has captured well the true cluster structure in the data.

Example 3 We return again to the optical recognition of handwritten digits data set, and apply PCA as well as the dimension reduction methods provided in **PPCI**. We consider the approach of obtaining a $k-1$ dimensional projection, where k is the number of clusters, as recommended by Ding and Li (2007); Niu et al. (2011). To assess the performance of the dimension reduction, we compute the average silhouette width (Kaufman and Rousseeuw, 2009) of the points in the reduced space, with respect to the true clusters. We use the implementation of silhouette width computations given in the **cluster** package (Maechler et al., 2015). For interest we also compute the average silhouette width of the original data.

```
> require(cluster)
> set.seed(1)
> data(optidigits)
> PCA_sol <- optidigits$x%*%eigen(cov(optidigits$x))$vectors[,1:9]
> mddr_sol <- mddr(optidigits$x, 9)
> mcdr_sol <- mcdr(optidigits$x, 9)
> ncutdr_sol <- ncutdr(optidigits$x, 9)
>
> mean(silhouette(optidigits$c, dist(optidigits$x))[,3])
[1] 0.1082235
> mean(silhouette(optidigits$c, dist(PCA_sol))[,3])
[1] 0.1731441
> mean(silhouette(optidigits$c, dist(mddr_sol$fitted))[,3])
[1] 0.2414112
> mean(silhouette(optidigits$c, dist(mcdr_sol$fitted))[,3])
[1] 0.2654211
> mean(silhouette(optidigits$c, dist(ncutdr_sol$fitted))[,3])
[1] 0.2224043
```

All dimension reduction methods improved the clusterability of the optidigits data set, with all three methods included in **PPCI** providing a substantial improvement over PCA.

Maximum margin hyperplanes for clustering: Maximum Margin Clustering (MMC) (Xu et al., 2004), seeks the maximum margin hyperplane to perform a bi-partition of unlabelled data. MMC can be equivalently viewed as identifying the binary labelling of $\mathcal{X}$ that will maximise the margin of a Support Vector Machine (SVM) classifier estimated on the labelled data set. Unlike the supervised SVM problem, which corresponds to a convex optimisation problem that can be efficiently solved, estimating MMC involves an integer optimisation problem. Exact methods for this problem are applicable only to very small data sets. Existing MMC algorithms that can handle reasonably sized data sets are not guaranteed to identify the globally optimal solution (Zhang et al., 2009).

The minimum density hyperplane and the minimum normalised cut hyperplane converge to the maximum hard margin hyperplane through the data, as the bandwidth (scaling) parameter is reduced

towards zero (Pavlidis et al., 2016; Hofmeyr, 2017). A simple and effective approach to obtain large margin hyperplanes for clustering is to apply either of the above methods for a decreasing sequence of bandwidth/scaling parameters.

Example 4 In this example we attempt to separate digits 3 and 9 from the optidigits data set. This is one of the most difficult binary classification benchmarks considered in Zhang et al. (2009). We only use the test set of the optidigits data set to render our results directly comparable to those reported by Zhang et al. (2009). The final 1797 data in the complete optidigits data set provided in **PPCI** correspond with the test cases. These are the indices 3824 to 5620.

To obtain large margin hyperplane separators we apply the `mdh()` function recursively. At each iteration after the first, the initial projection vector is set to the optimal vector identified in the previous iteration, while the bandwidth parameter is multiplied by 0.9. The bandwidth can be specified with the optional argument `bandwidth`. As before, so as not to affect the initialisation through the penalty term in the MDH formulation, the initial value of α is set to the final value from the previous solution (using the optional argument `alphamin`). This and the previous bandwidth value may be extracted from the `$params` field of the solution. The outputs of all of the binary-partitioning hyperplane algorithms in **PPCI** contain the field `$params`, a named list storing the parameters used in the projection pursuit.

```
> ids <- (3824:5620)[which(optidigits$c[3824:5620]%in%c(3, 9))]
>
> x39 <- optidigits$x[ids,]
> hp0 <- mdh(x39)
>
> hp <- hp0
>
> repeat{
>   hp_new <- mdh(x39, v0 = hp$v, bandwidth = hp$params$h*0.9,
        alphamin = hp$params$alpha)
>   if(hp$v%*%hp_new$v>(1-1e-10)) break
>   else hp <- hp_new
> }
>
> plot(hp0)
> plot(hp)
>
> 1-cluster_performance(hp$cluster, optidigits$c[ids])[2]
      purity
  0.02479339
```

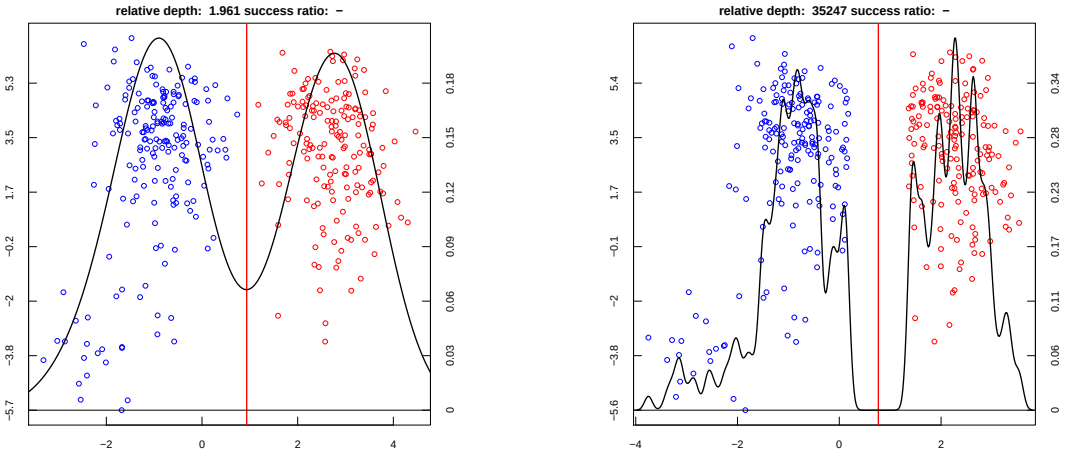
The first and last MDHs obtained from the above example are illustrated in Figure 14. As the figure shows, the optimal MDH for the smallest bandwidth achieves a large margin, unlike the MDH for the default setting of h. Notice that in Figure 14(b) the very small value of the bandwidth causes the density on the separating hyperplane to be very close to zero, and consequently the relative depth is extremely large.

For this data set (Zhang et al., 2009, Table IV) report the clustering error of k-means, kernel k-means, normalised cut spectral clustering (Shi and Malik, 2000), generalised maximum margin clustering (Valizadegan and Jin, 2006), and three versions of the iterative support vector machine (regression) they propose. The best performance is achieved by generalised maximum margin clustering with an error of 0.083. Both versions of k-means, and spectral clustering achieve an error around 0.2. The more recent cutting plane maximum margin clustering algorithm (Wang et al., 2010) achieves a much larger error of 0.3609 on this data set.[2] The large margin hyperplane obtained through repeatedly reducing the bandwidth parameter of the MDH algorithm achieves an error that is more than three times smaller than that of the best competing algorithm.

Non-linear cluster separators: Because the equation which defines a hyperplane is a linear one, the boundary between clusters separated by a hyperplane is linear. Hyperplanes have been very successful in accurately separating clusters in many real world applications, and especially in high dimensional data sets (Boley, 1998; Tasoulis et al., 2010, 2012; Zhang et al., 2009), however it is simple to construct examples where linear cluster separators are too restrictive. Specifically, if the convex hulls of clusters overlap, then they cannot be perfectly separated by hyperplanes.

Non-linear embedding of a data set into a high-dimensional *feature space* is a common technique used to separate clusters with overlapping convex hulls using hyperplanes. In the most well known

[2]MATLAB code for this method is available at https://sites.google.com/site/binzhao02/



(a) MDH for default bandwidth

(b) MDH for smallest bandwidth considered

Figure 14: Large margin hyperplane separator for the optdigits 3-9 problem obtained by applying `mdh()` for a decreasing sequence of bandwidths

context of Support Vector Machines (Boser et al., 1992, SVM), this embedding is only implicitly performed using the so-called "kernel trick". Here we show through a simple example how to achieve this non-linear separation using the methods in **PPCI**, by first pre-processing the data using kernel Principal Component Analayis (Schölkopf et al., 1998, KPCA). We use the implementation of KPCA provided in the package **kernlab** (Karatzoglou et al., 2004). We selected the hyperparameter KPCA using trial and error; choosing a value for which the cluster sizes were reasonably well balanced. The setting of kernel hyperparameters remains a challenging problem in the unsupervised setting.

Example 5 We revisit the popular "two-moons" toy data set of (Jain and Law, 2005). We first use KPCA to embed the data in a high-dimensional feature space in which the clusters are linearly separable before applying a hyperplane based clustering algorithm, namely NCUTDC.

```
> require(kernlab)
> set.seed(1)
> th <- runif(500)*pi + rep(c(pi,0),each=250)
>
> x <- cbind(cos(th) + rep(c(0.5,-0.5),each=250), sin(th)+ rep(c(0.2,-0.2),each=250))
>  x <- x + matrix(0.1*rnorm(1000), ncol=2)
> x2 <- kernlab::kpca(x, kernel = "rbfdot", kpar = list(sigma = 3))@rotated
> sol1 <- ncuth(x)
> sol2 <- ncuth(x2)
> par(mfrow = c(1, 2))
> par(mar = c(2.5, 2.5, 3, 2.5))
> plot(x, col = sol1$cluster, main = "Linear Separator")
> plot(x, col = sol2$cluster, main = "Non-linear Separator")
```

In this example we combined the non-linear embedding of KPCA with the linear separation framework used in **PPCI**, effectively borrowing much of the added flexibility offered by non-linear dimension reduction techniques. However, this approach does not mitigate the increased computational cost associated with non-linear methods. When the number of data is large, we prefer the simple and common approaches of constructing the KPCA embedding using either a random subset of the data, or using a compressed version of the data obtained from a simple clustering technique, such as *k*-means.

Conclusions

This paper discusses three recently proposed projection pursuit methods for clustering, and the implementation in the R package **PPCI**. The projection pursuit methods seek univariate projections that maximise the clusterability of the binary partition of the projected data, or alternatively minimise the connectedness between the elements of this bi-partition. Identifying projection directions that explicitly optimise a clustering criterion can lead to significant improvements over PCA and other

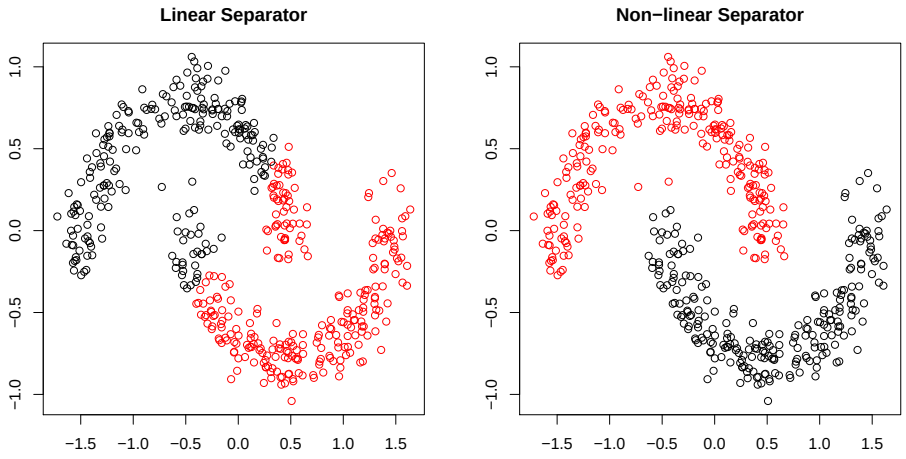


Figure 15: Nonlinear cluster separation after preprocessing the data through Kernel PCA

classical dimension reduction techniques. The cluster boundaries induced by these methods are hyperplanes, which can be combined to produce divisive hierarchical clustering models capable of identifying clusters defined in different subspaces and with different scales. An important advantage of using hyperplanes for clustering is that projecting onto the vector normal to the hyperplane provides the maximum dimensionality reduction, in that the projected data are one dimensional. Visualising the data through such projections allows the user to obtain insights concerning the validity of the clustering model. We discuss how kernel PCA can be combined with the implemented methods to handle clusters which cannot be linearly separated. Two of the implemented projection pursuit algorithms are asymptotically equivalent to the maximum hard margin separator as their smoothing parameters are reduced to zero. We illustrate how large margin hyperplanes for clustering can be obtained using these methods.

Bibliography

M. Ackerman and S. Ben-David. Clusterability: A theoretical study. In D. van Dyk and M. Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 1–8, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009. PMLR. URL http://proceedings.mlr.press/v5/ackerman09a.html. [p6]

F. R. Bach and M. I. Jordan. Learning spectral clustering, with application to speech separation. *Journal of Machine Learning Research*, 7:1963–2001, 2006. [p1]

D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998. ISSN 1573-756X. URL https://doi.org/10.1023/a:1009740529316. [p1, 20]

R. J. Bolton and W. J. Krzanowski. Projection pursuit clustering for exploratory data analysis. *Journal of Computational and Graphical Statistics*, 12(1):121–142, 2003. URL https://doi.org/10.1198/1061860031374. [p1]

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992. [p21]

J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15(3):751–779, 2005. URL https://doi.org/10.1137/030601296. [p6]

D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. ISSN 0162-8828. URL https://doi.org/10.1109/34.1000236. [p6]

C. Ding and X. He. K-means clustering via principal component analysis. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 29. ACM, 2004. [p1]

C. Ding and T. Li. Adaptive dimension reduction using discriminant analysis and k-means clustering. In *Proceedings of the 24th International Conference on Machine Learning*, pages 521–528. ACM, 2007. [p2, 19]

J. A. Hartigan. *Clustering Algorithms*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1975. [p6]

D. Hofmeyr and N. Pavlidis. Maximum clusterability divisive clustering. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 780–786, 2015. URL https://doi.org/10.1109/ssci.2015.116. [p1, 3, 6]

D. P. Hofmeyr. Clustering by minimum cut hyperplanes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1547–1560, 2017. ISSN 0162-8828. URL https://doi.org/10.1109/tpami.2016.2609929. [p1, 3, 7, 8, 20]

D. P. Hofmeyr. Improving spectral clustering using the asymptotic value of the normalised cut. *Journal of Computational and Graphical Statistics*, (just-accepted):1–25, 2019. [p8]

D. P. Hofmeyr, N. G. Pavlidis, and I. A. Eckley. Minimum spectral connectivity projection pursuit. *Statistics and Computing*, 29(2):391–414, 2019. ISSN 1573-1375. URL https://doi.org/10.1007/s11222-018-9814-6. [p1]

P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985. [p1, 5, 19]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. URL https://doi.org/10.1007/bf01908075. [p18]

A. K. Jain and M. H. C. Law. Data clustering: A user's dilemma. In S. K. Pal, S. Bandyopadhyay, and S. Biswas, editors, *Pattern Recognition and Machine Intelligence: First International Conference, PReMI 2005, Kolkata, India, December 20-22, 2005. Proceedings*, pages 1–10, Berlin, Heidelberg, 2005. Springer-Verlag. URL https://doi.org/10.1007/11590316_1. [p9, 21]

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999. URL https://doi.org/10.1145/331499.331504. [p1]

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. Kernlab - an S4 package for kernel methods in R. *Journal of Statistical Software, Articles*, 11(9):1–20, 2004. URL https://doi.org/10.18637/jss.v011.i09. [p21]

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*, volume 344. John Wiley & Sons, 2009. [p19]

A. Krause and V. Liebscher. Multimodal projection pursuit using the dip statistic. *Preprint-Reihe Mathematik*, 13, 2005. [p1]

H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data*, 3(1):1–58, 2009. [p1]

A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141(1):135–163, 2013. URL https://doi.org/10.1007/s10107-012-0514-2. [p6]

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. [p8, 11, 14]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *Cluster: Cluster Analysis Basics and Extensions*, 2015. R package version 2.0.3 — For new features, see the 'Changelog' file (in the package source). [p19]

D. Niu, J. G. Dy, and M. I. Jordan. Dimensionality reduction for spectral clustering. In *International Conference on Artificial Intelligence and Statistics*, pages 552–560, 2011. [p1, 2, 19]

D. Niu, J. G. Dy, and M. I. Jordan. Iterative discovery of multiple alternative clustering views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1340–1353, 2014. [p1]

N. G. Pavlidis, D. P. Hofmeyr, and S. K. Tasoulis. Minimum density hyperplanes. *Journal of Machine Learning Research*, 17(156):1–33, 2016. URL http://jmlr.org/papers/v17/15-307.html. [p1, 3, 6, 16, 20]

D. Peña and F. J. Prieto. Cluster identification using projections. *Journal of the American Statistical Association*, 96(456):1433–1445, 2001. URL https://doi.org/10.1198/016214501753382345. [p1]

A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, volume 7, pages 410–420, 2007. URL http://www1.cs.columbia.edu/~amaxwell/pubs/v_measure-emnlp07.pdf. [p18]

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. URL https://doi.org/10.1162/089976698300017467. [p21]

J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000. URL https://doi.org/10.1109/34.868688. [p7, 20]

A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002. URL http://www.jmlr.org/papers/v3/strehl02a.html. [p18]

S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos. Enhancing principal direction divisive clustering. *Pattern Recognition*, 43(10):3391–3411, 2010. URL https://doi.org/10.1016/j.patcog.2010.05.025. [p1, 20]

S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos. Random direction divisive clustering. *Pattern Recognition Letters*, 2012. URL https://doi.org/10.1016/j.patrec.2012.09.008. [p20]

M. Thrun, F. Lerch, and F. Pape. *ProjectionBasedClustering: Projection Based Clustering*, 2018. URL https://CRAN.R-project.org/package=ProjectionBasedClustering. R package version 1.0.6. [p2]

M. C. Thrun. *Projection-Based Clustering through Self-Organization and Swarm Intelligence*. Springer-Verlag, 2018. [p2]

N. G. Trillos, D. Slepvcev, J. Von Brecht, T. Laurent, and X. Bresson. Consistency of cheeger and ratio graph cuts. *The Journal of Machine Learning Research*, 17(1):6268–6313, 2016. [p8]

A. Ultsch and M. C. Thrun. Credible visualizations for planar projections. In *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 1–5. IEEE, 2017. [p2]

H. Valizadegan and R. Jin. Generalized maximum margin clustering and unsupervised kernel learning. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1417–1424. MIT Press, 2006. URL http://papers.nips.cc/paper/3072-generalized-maximum-margin-clustering-and-unsupervised-kernel-learning.pdf. [p20]

U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. ISSN 0960-3174. URL https://doi.org/10.1007/s11222-007-9033-z. [p7]

F. Wang, B. Zhao, and C. Zhang. Linear time maximum margin clustering. *IEEE Transactions on Neural Networks*, 21(2):319–332, 2010. URL https://doi.org/10.1109/tnn.2009.2036998. [p20]

L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1537–1544. MIT Press, 2004. URL http://papers.nips.cc/paper/2602-maximum-margin-clustering.pdf. [p19]

B. Zhang. Dependence of clustering algorithm performance on clustered-ness of data. *HP Labs Technical Report HPL-2001-91*, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.285&rep=rep1&type=pdf. [p6]

K. Zhang, I. W. Tsang, and J. T. Kwok. Maximum margin clustering made practical. *IEEE Transactions on Neural Networks*, 20(4):583–596, 2009. URL https://doi.org/10.1109/tnn.2008.2010620. [p19, 20]

Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004. URL https://doi.org/10.1023/b:mach.0000027785.44527.d6. [p18]

David P. Hofmeyr
Stellenbosch University
Stellenbosch
South Africa
dhofmeyr@sun.ac.za

Nicos G. Pavlidis
Lancaster University
Lancaster
United Kingdom
n.pavlidis@lancaster.ac.uk