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. Clustering models can also be modified in an interactive manner to improve their solutions. 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 (**?**), with no universal consensus and no definition which is appropriate for all applications. Without a ground truth solution clusters must be determined by the relative spatial relationships between points. However, the spatial structure can be less informative for determining clusters in the presence of irrelevant/noisy features, as well as correlations among subsets of features. Such characteristics abound especially in high dimensional applications, and make the clustering problem especially challenging. To accurately cluster such data sets it becomes necessary to identify subspaces which allow a strong separation of clusters. The best subspace to separate clusters will clearly depend on the cluster definition employed, and moreover a single subspace of fixed dimension may not allow a complete separation of all clusters.

A principled approach to finding high quality 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* (**?**). 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. This index is known to be maximised by the eigenvector associated with the largest eigenvalue of the covariance matrix. For most other projection indices no closed form solution is available and these objectives are instead numerically optimised. Although PCA has been successfully applied in numerous clustering problems, there is no guarantee that any number of principal components will be relevant for preserving/exposing cluster structure. This is unsurprising as it is trivial to construct data sets in which clusters are only separated along directions of low data variability. As will be seen in the remainder, when the clustering objective is included in the projection index, substantial improvements can be made.

As far as we are aware the only existing **R** package which combines projection pursuit and clustering is **ProjectionBasedClustering** (**?**). **ProjectionBasedClustering** provides both linear and non-linear dimension reduction techniques, including Independent Component Analysis (**?**, ICA) and t-distributed Stochastic Neighbour Embedding (**?**, t-SNE). None of these incorporates a clustering criterion directly into the dimension reduction formulation. As a result there is no guarantee that the lower dimensional embeddings of the data will exhibit any cluster structure. Moreover different dimension reduction techniques may lead to very different low dimensional embeddings. This is problematic from the user's perspective. Even if the user knows the type of clusters which are of interest it is unclear which dimension reduction technique is most appropriate for their problem. The **subspace** package (**?**) also performs projection based clustering, including methods such as CLIQUE (**?**) and SubClu (**?**). The approach adopted by these methods differs fundamentally from ours in that clusters are defined through grid cells which have high data density when projected onto multiple axes of the input space. There is thus no search for an optimal subspace/projection of the data.

In this paper we present the **R** package **PPCI** which provides implementations of three recently developed projection pursuit methods for clustering. The projection indices underlying these methods are based on three popular clustering objectives, including those underlying k-means; density clustering; and clustering by normalised graph cuts. With no guarantee of the existence of a single subspace

to distinguish all clusters, we instead adopt the approach of repeatedly identifying a single univariate subspace which allows the separation of at least one cluster from the remainder. The separations of clusters are thus induced by hyperplanes in the original space. The overall clustering model has a divisive hierarchical structure, in which each cluster is formed by the intersection of finitely many half-spaces.

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

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

The remaining paper contains a formulation of the general problem of divisive hierarchical clustering using projection pursuit, and a description of the three methods included in **PPCI**. Thereafter we provide instructive examples and important extensions of these approaches.

Projection pursuit, hyperplanes and divisive hierarchical clustering

In this section we introduce a general framework for finding optimal univariate projections for separating clusters. Since the induced cluster boundaries are hyperplanes with normal vector equal to the optimal projection vector, we approach this problem from the point of view of defining optimal hyperplanes to bi-partition the data, and from these derive associated projection indices. We then discuss briefly how to combine such hyperplanes to produce a divisive hierarchical clustering model.

Finding optimal hyperplanes via projection pursuit

To begin with, assume the data set consists of a finite set of vectors in $\mathbb{R}^d$, $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$. For a unit-length vector $\mathbf{v} \in \mathbb{B}^d = \left\{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x}\|_2 = 1\right\}$, we use $\mathbf{v}^\top \mathcal{X} = \{\mathbf{v}^\top \mathbf{x}_i\}_{i=1}^n$ to denote the projection of $\mathcal{X}$ onto $\mathbf{v}$. Now, a hyperplane in $\mathbb{R}^d$ is a translated subspace of co-dimension one, that is parameterised by a pair $(\mathbf{v}, b) \in \mathbb{B}^d \times \mathbb{R}$, 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 $\mathcal{X}$ induces a binary partition based on the half-space to which each observation is allocated,

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

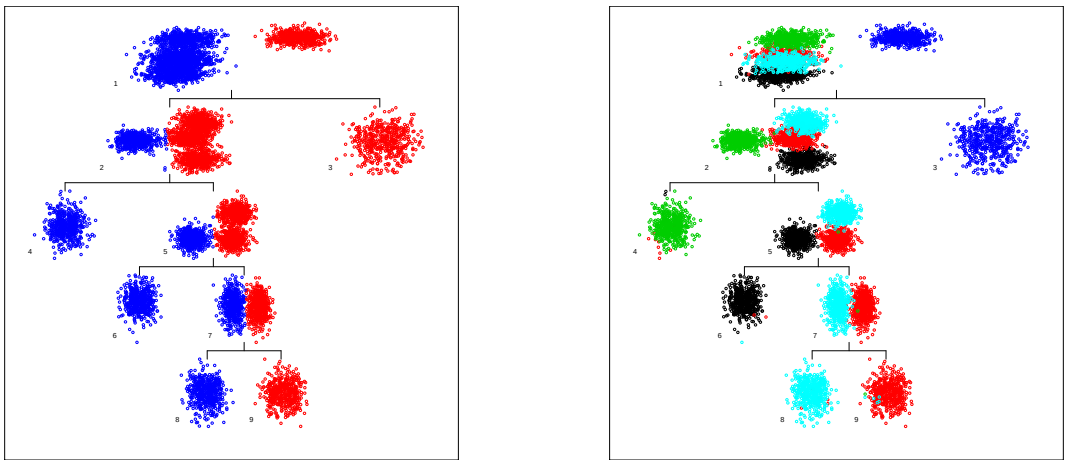
Hyperplanes are linear cluster separators as the boundary between the two clusters is defined by the linear equation $\mathbf{v}^\top \mathbf{x} = b$. It is simple to construct examples where linear cluster separators are too restrictive, however hyperplanes have been very successful in accurately separating clusters in many real world applications, and especially in high dimensional data sets (**????**).

One of the benefits of employing hyperplanes for clustering is that projecting onto a vector (in this case the vector normal to the hyperplane) achieves the maximum reduction of dimensionality. Furthermore the associated clustering problems within the corresponding one-dimensional subspace become tractable. Now, if the *projection vector*, $\mathbf{v}$, is a combination of features that are not relevant for distinguishing clusters then there will be no hyperplane orthogonal to $\mathbf{v}$ which induces a high quality binary partition of $\mathcal{X}$. If instead $\mathbf{v}$ is a combination of features along which clusters are separable, then there will exist a b for which $H(\mathbf{v}, b)$ induces a high quality binary partition of $\mathcal{X}$.

Let Q be a measure of quality of a binary partition, where quality can be measured either in terms of a high degree of clusterability, or low degree of connectedness between the two components. The quality of a hyperplane $H(\mathbf{v}, b)$ can then be written as,

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

When the number of clusters is large, the objective $\phi(\mathbf{v}, b|\mathcal{X})$ has numerous local optima. As a result optimising $\phi(\mathbf{v}, b|\mathcal{X})$ directly with respect to both parameters, $\mathbf{v}$ and b, can lead to partitions of only moderate quality. To mitigate this we use a projection pursuit formulation in which the projection index, $\Phi(\mathbf{v}|\mathcal{X})$ in Eq. (2), is the quality of the best hyperplane with normal vector $\mathbf{v}$. If Q is a measure of



(a) Without cluster labels

(b) With true cluster labels

Figure 1: Divisive hierarchical clustering model for simulated data set based on Maximum Clusterability Divisive Clustering algorithm

clusterability then the optimal projection vector, $\mathbf{v}_{\text{opt}}$, maximises $\Phi(\mathbf{v}|\mathcal{X})$ over all unit-length vectors,

$$\mathbf{v}_{\text{opt}} = \arg\max_{\mathbf{v}\in\mathbb{B}^d} \Phi(\mathbf{v}|\mathcal{X}), \tag{1}$$

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

If instead Q measures the connectedness between the elements of the binary partition then Eqs. (1) and (2) become minimisation problems. Since Q is arbitrary this provides a framework for deriving projection indices for essentially any clustering objective. By considering only the best hyperplane orthogonal to a candidate projection vector this formulation allows b to change discontinuously, and hence avoids many local optima of $\phi(\mathbf{v}, b|\mathcal{X})$. 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. 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. **?** strongly advocate using BFGS to optimise such non-smooth locally Lipschitz functions, over more computationally expensive methods like gradient sampling (**?**). We therefore use the existing optimised implementation of BFGS provided in **R**'s base **stats** package.

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, $i = \arg\max_{j\in\text{leaves}(T)} \Phi(\mathbf{v}_{\text{opt}}|\mathcal{C}_j)$.

Example 1 *We apply the Maximum Clusterability Divisive Clustering algorithm (**?**, MCDC) to 10 dimensional simulated data arising from a Gaussian mixture containing 5 components, each representing a cluster. The function* `mcdc()` *implements this algorithm and is described in detail later. The resulting hierarchical clustering model is visualised through the* `tree_plot()` *function, and depicted in Figure 1. Details of the arguments of this function are given in Table 1*

Each scatterplot in Figure 1 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 $\mathbf{v}_{\text{opt}}$. If the `labels` *argument is not specified, as in Figure 1(a), 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. When*

Argument	Description
`sol`	A clustering solution from any clustering algorithm in **PPCI**.
`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 the order in which the nodes were added to the hierarchy is indicated in the plot.

Table 1: Arguments accepted by `tree_plot` function.

the `labels` *argument is specified the points in each scatterplot are coloured according to their corresponding labels, as shown in Figure 1(b).*

```
> set.seed(1)
> means <- matrix(rnorm(50), ncol = 10)
> X <- matrix(, 2500, 10)
> for(i in 1:5) X[((i-1)*500+1):(i*500),] <- t(matrix(rnorm(5000, 0, .5), ncol = 500) + means[i,])
> sol <- mcdc(X, 5)
> tree_plot(sol)
> tree_plot(sol, labels = rep(1:5, each = 500))
```

Clustering and Projection Pursuit in PPCI

In this section we describe the projection pursuit and clustering methods provided in **PPCI**, as well as their implementation in **R**. The main functions are described in Table 2. The projection pursuit algorithms find optimal projections and corresponding optimal hyperplanes to bi-partition the data, while the clustering algorithms obtain a complete divisive hierarchical clustering model.

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 **?**.
`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 **?**
`ncuth`	`ncutdc`	Uses hyperplanes with minimum normalised cut measured across them, using the method of **?**. This leads to partitions with low between and high within cluster similarity.

Table 2: Projection pursuit and clustering algorithms in **PPCI**.

The only mandatory argument for the projection pursuit methods (`mdh`,`mch` and `ncuth`) is the data matrix `X`, while the clustering algorithms (`mddc`,`mcdc` and `ncutdc`) require both the data matrix, `X`, and the number of clusters to extract, `K`. In addition all methods accept numerous optional arguments which allow the user to modify the parameters associated with the projection pursuit. When omitted all optional arguments are assigned default values. Table 3 provides a complete list of the arguments for these methods. For full details use the `help()` command within the **R** console, providing the relevant function name as argument, e.g. '`help(mdh)`'.

The output of the projection pursuit algorithms (`mdh`,`mch` and `ncuth`) is an unnamed list of length m, where m is the number of initialisations considered. Each element of the output is a named list containing the details of the corresponding optimal solution. The output of the clustering algorithms (`mddc`,`mcdc` and `ncutdc`) is a named list containing details of the clustering solution. The most important components of the output are the `$cluster` field, which contains the cluster assignment vector, and the `$Nodes` field, which is an unnamed list in which each element contains the details of the corresponding node in the hierarchical clustering model.

In the following subsections we provide a brief description of each of the methods implemented in **PPCI**, as well as examples illustrating the usage 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. (5)).
`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. (5)).
`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 3: Arguments accepted by clustering and projection pursuit functions.

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. Clusters are defined either as "regions of attraction" of each mode of p (**???**); or as maximally connected components of the level sets $\left\{\mathbf{x} \in \mathbb{R}^d \mid p(\mathbf{x}) > \lambda\right\}$, for a suitable choice of the level parameter λ (**???**). An immediate consequence of this definition is that cluster boundaries traverse regions of low probability density, known as the *low density separation* assumption (**?**). A number of influential methods for clustering and semi-supervised classification use low density hyperplanes, including maximum margin clustering (**?**) and semi-supervised support vector machines (**??**). Although all these methods use maximum margin hyperplanes to approximate low density separators, the consistency of this approach remains an open question (**?**).

The Minimum Density Hyperplane (**?**, MDH) directly estimates the hyperplane with minimum density based on an estimated density $\hat{p}$. Following **?** the estimated density on $H(\mathbf{v}, b)$, $\hat{I}(\mathbf{v}, b)$ in Eq. (3), is defined as the surface integral of $\hat{p}$ on $H(\mathbf{v}, b)$. By estimating p through a Gaussian kernel mixture $\hat{I}(\mathbf{v}, b)$ can be computed exactly using only the projections of $\mathcal{X}$ onto $\mathbf{v}$,

$$\hat{p}(\mathbf{x}) = \frac{1}{n(2\pi h^2)^{d/2}} \sum_{i=1}^{n} e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2h^2}} \Rightarrow \hat{I}(\mathbf{v}, b) := \int_{H(\mathbf{v},b)} \hat{p}(\mathbf{x}) d\mathbf{x} = \frac{1}{n\sqrt{2\pi h^2}} \sum_{i=1}^{n} e^{-\frac{(b-\mathbf{v}^\top \mathbf{x}_i)^2}{2h^2}}. \quad (3)$$

It is clear that for any vector $\mathbf{v}$ one has $\lim_{b \to \pm\infty} \hat{I}(\mathbf{v}, b) = 0$, that is, hyperplanes passing through

Argument	Description
`sol`	A solution from any projection pursuit algorithm in **PPCI**.
`X`	The data matrix used to obtain `sol`.
`labels`	(optional) A vector of class labels. Points with the same label are plotted with the same colour.

Table 4: Arguments accepted by `hp_plot` function.

the tail of the estimated density can be made to have arbitrarily small density. To avoid such solutions which are not meaningful for clustering, a penalty term is added to $\hat{I}(\mathbf{v}, b)$ to constrain the distance of the optimal hyperplane from the mean of the data (which without loss of generality can be assumed to be zero). Specifically, the Minimum Density Projection Pursuit (**?**, MDP^2) algorithm is based on the following optimisation problem,

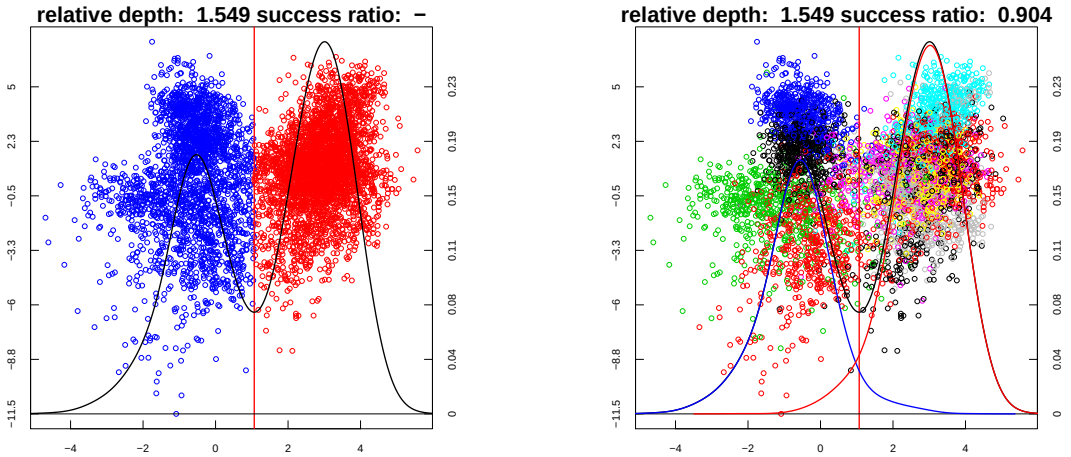
$$\min_{\mathbf{v}} \Phi(\mathbf{v}|\mathcal{X}) = \min_{b \in \mathbb{R}} \phi(\mathbf{v}, b|\mathcal{X}), \tag{4}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) = \hat{I}(\mathbf{v}, b) + C \max\{0, -\alpha\sigma_{\mathbf{v}} - b, b - \alpha\sigma_{\mathbf{v}}\}^{1+\epsilon}, \tag{5}$$

for any $\epsilon \in (0, 1)$, and a constant C. The term $\sigma_{\mathbf{v}}$ denotes the standard deviation of the projected data $\mathbf{v}^\top \mathcal{X}$. The value of α controls the distance of the optimal solution to the mean of the data, and is dynamically adjusted during the execution of the algorithm.

Example 2 *In this example we first apply* `mdh()` *and then* `mddc()` *to obtain a binary partition, and then a complete clustering of the optical recognition of handwritten digits (optidigits) data set from the UCI repository (**?**). The data set we consider combines the training and test set data sets in UCI and contains 5620 observations. Observations correspond to vectorised images of handwritten digits 0–9. The images have been compressed to 8×8 pixels resulting in a 64 dimensional data set.*

```
> data(optidigits)
> sol <- mdh(optidigits$x)
> hp_plot(sol, optidigits$x)
> hp_plot(sol, optidigits$x, labels = optidigits$c)
> success_ratio(sol[[1]]$cluster, optidigits$c)
 [1] 0.9040637
```



(a) Labels not specified (b) Labels specified

Figure 2: Two-dimensional visualisation of binary partition of the optical handwritten digits recognition data set through `mdh()`.

The function `hp_plot` *provides a visualisation of hyperplane separators obtained from any projection pursuit algorithm in* **PPCI**. *The arguments for this function are described in Table 4. The output of the first call to this function is depicted in Figure 2(a). If the* `labels` *argument is not specified the points are coloured according to the half space to which they belong. The projected density is depicted with a solid black line, with its scale depicted on the right vertical axis. If the* `labels` *argument is specified, as in the second call to* `hp_plot` *above, the colours of the projected points represent the true cluster assignments, as shown in Figure 2(b). In this case*

*each true cluster can be assigned to the half space that contains the majority of its observations. The projected density can then be seen as a two component mixture density arising from the clusters assigned to the two half spaces. The red and blue solid lines illustrate these two component densities. The quality of a binary partition of a data set containing more than two clusters can be evaluated through the success ratio (**?**). The success ratio is reported in the title of Figure 2(b), and can be computed through the* `success_ratio(clusters,labels)` *function.*

Next we use the `mddc()` *algorithm to obtain a complete clustering of the optidigits data set.*

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

The visualisation of the cluster hierarchy through the `tree_plot` *function has been discussed in Example 1). The* `cluster_performance(clusters,labels)` *function implements four external cluster validity measures: purity (**?**), normalised mutual information (NMI) (**?**), adjusted Rand index (**?**), and V-measure (**?**).*

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 (**?**). The *variance ratio* clusterability is defined as (**?**),

$$\max_{\substack{\mathcal{X}=\mathcal{C}_1\cup\cdots\cup\mathcal{C}_k \\ \mathcal{C}_i\cap\mathcal{C}_j=\emptyset, i\neq j}} \frac{\sum_{i=1}^{k} |\mathcal{C}_i| \, \|\boldsymbol{\mu}_{\mathcal{C}_i} - \boldsymbol{\mu}\|^2}{\sum_{i=1}^{k} \sum_{j:\mathbf{x}_j\in\mathcal{C}_i} \|\mathbf{x}_j - \boldsymbol{\mu}_{\mathcal{C}_i}\|^2}, \tag{6}$$

where $\boldsymbol{\mu}_{\mathcal{C}_i} = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x}_j\in\mathcal{C}_i} \mathbf{x}_j$ is the mean of the i-th cluster and $\boldsymbol{\mu} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$ is the mean of the data. This represents the ratio of the between and within cluster scatter values, 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. This renders clusterability in the general setting primarily of theoretical interest. 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. This is important as the values of the variance ratio and of the k-means objective are largely determined by directions in which the data exhibit high variability. This is problematic when the clusters are not separable along these directions. The Maximum Clusterability Divisive Clustering algorithm (MCDC) recursively partitions a data set by identifying the univariate subspace that maximises the variance ratio of the projected data. Since the variance ratio measure is scale invariant, this objective is able to overcome situations where directions which separate clusters are of relatively low variability.

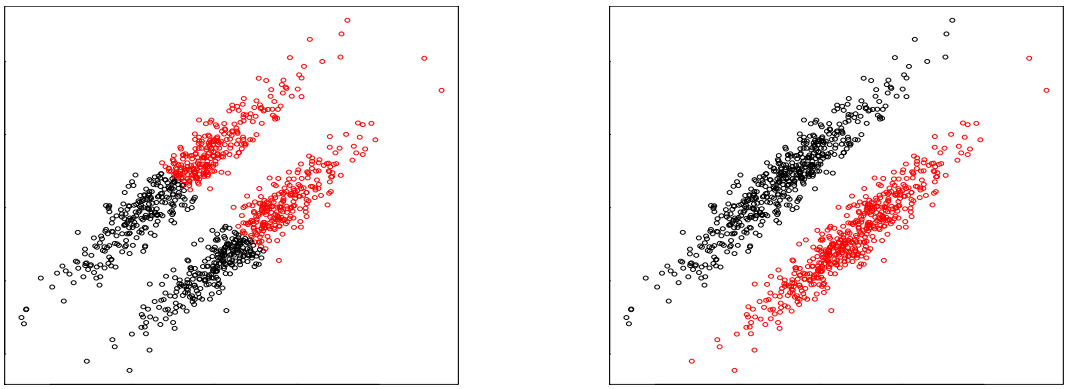
It is straightforward to see that the optimal partition of $\mathbf{v}^\top\mathcal{X}$ based on the variance ratio splits the projected data at a point. The projection index can thus be written as

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

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

The term $\frac{n}{n-1}\sum_{i=1}^{n}\left(\mathbf{v}^\top\mathbf{x}_i - \mu_{\mathbf{v}^\top\mathcal{X}}\right)^2$ in the denominator is added to ensure that $\Phi(\mathbf{v}, b|\mathcal{X})$ is bounded, without affecting the optimal solution.

Example 3 *We consider a simple two-dimensional example, seen in Figure 3, to illustrate the effectiveness of using variance ratio as a projection index. This example contains two well separated and easily distinguished clusters. However, the elongation of the clusters along their principal axes means that the variability of the data along this direction dominates the variability in the direction which separates the clusters. The k-means solution, Figure 3(a), is severely influenced by this fact. On the other hand by maximising the variance ratio of the projected data the true clusters can be recovered, Figure 3(b).*



(a) *k*-means solution (b) MCDC solution

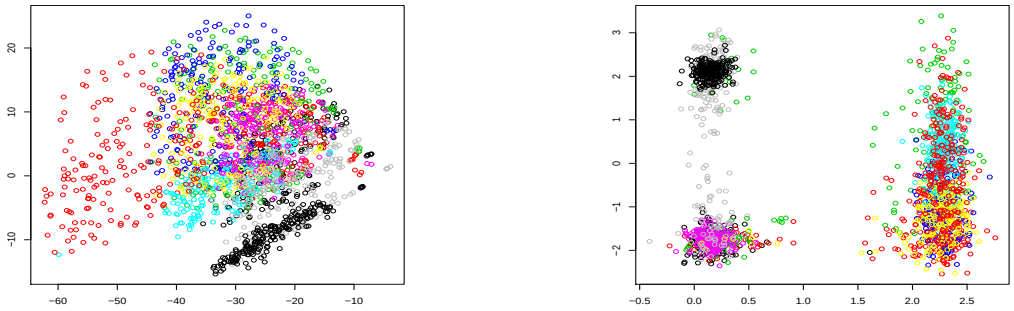
Figure 3: Data set containing two well distinguished clusters. High within cluster variability along the principal axis dominates the between cluster variability causing *k*-means to fail, (a). MCDC on the other hand is able to recover the true solution, (b).

```
> set.seed(1)
> S <- matrix(c(1, .7, .7, 1), 2, 2)
> X <- matrix(rnorm(2000), ncol = 2) %*% S
> X[,1] <- X[,1] + rep(c(.8, -.8), each = 500)
> X[,2] <- X[,2] + rep(c(-.8, .8), each = 500)
>
> km <- kmeans(X, 2, nstart = 10)
> sol <- mch(X)
>
> par(mfrow = c(1, 2))
> plot(X, col = km$cluster, main = "kmeans solution")
> plot(X, col = sol[[1]]$cluster, main = "MCH solution")
```

Example 4 *We next consider a face recognition task with ten clusters in which centroid based algorithms like k-means perform suboptimally because the within cluster covariance structure dominates the between cluster separation. We apply both k-means and MCDC to a subset of the Yale face database B (**?**), containing 2000 portrait images of 10 different human subjects in different lighting conditions and with different orientations. The images have been compressed to* 30×20 *pixels and vectorised, resulting in 600 dimensional data.*

```
> require(rARPACK)
> set.seed(1)
> data(yale)
> km <- kmeans(yale$x, 10, nstart = 10)
> sol <- mcdc(yale$x, 10)
>
> pc <- rARPACK::eigs_sym(cov(yale$x), 2)$vectors
> par(mfrow = c(1, 2))
> plot(yale$x%*%pc, xlab = "PC1", ylab = "PC2",
     main = "Principal component projection", col = yale$c)
> plot(yale$x%*%cbind(sol$Nodes[[1]]$v, sol$Nodes[[2]]$v), xlab = "VR1",
     ylab = "VR2", main = "Projection based on variance ratio", col = yale$c)
>
> cluster_performance(sol$cluster, yale$c)
 adj.rand    purity v.measure       nmi
 0.7206458 0.8220000 0.8309754 0.8309877
> cluster_performance(km$cluster, yale$c)
 adj.rand    purity v.measure       nmi
 0.5006819 0.6675000 0.6798313 0.6800078
```

Figure 4 shows two dimensional projections of these data. In Figure 4(a) the data are projected along the first two principal components, while in Figure 4(b) the projection vectors are taken as the optimal projection vectors from the first two nodes in the MCDC model. Although there is evidence of the presence of multiple clusters in the principal component projections, these high variance projections do not admit a separation of any clusters



(a) Principal component projection

(b) Projection based on variance ratio

Figure 4: Two dimensional projections of vectorised images taken from the Yale faces database B

from the rest. In contrast the projections based on variance ratio show very strong evidence of at least three clusters. The remaining clusters are identified further down the hierarchical model. The presence of elongated cluster shapes, evident in Figure 4(a), prevents k-means from achieving performance as high as that of MCDC.

Minimum normalised cut hyperplanes (`ncuth` and `ncutdc`)

Arising from graph partitioning algorithms, the normalised cut objective has become popular for data clustering (**?**). The normalised cut (NCut) associated with a partition of $\mathcal{X}$ into clusters $\mathcal{C}_1, ..., C_k$ is given by (**?**),

$$\text{NCut}(\mathcal{C}_1, \ldots, \mathcal{C}_k) = \sum_{m=1}^{k} \frac{\text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m)}{\text{volume}(\mathcal{C}_m)}, \tag{7}$$

$$\text{Cut}(\mathcal{C}, \mathcal{X} \setminus \mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C}, \\ \mathbf{x}_j \notin \mathcal{C}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j), \quad \text{volume}(\mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C} \\ \mathbf{x}_j \in \mathcal{X}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j). \tag{8}$$

By minimising the normalised cut one tends to find solutions for which $\text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m)$ is small and $\text{volume}(\mathcal{C}_m)$ is large, for all m. Now, since $\text{volume}(\mathcal{C}_m) = \text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m) + \sum_{i,j:\mathbf{x}_i,\mathbf{x}_j \in \mathcal{C}_m} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j)$, where the latter term is the total internal similarity of points in $\mathcal{C}_m$, this implies that the similarity within clusters is high but the similarity between clusters is low. This makes NCut an attractive objective for clustering. However, the NCut problem is NP-hard, and unlike the k-means objective it is not straightforward to obtain a high quality locally optimal solution efficiently. Instead it is common to consider a continuous relaxation of the problem known as spectral clustering (**??**). This reduces the complexity to $\mathcal{O}(kn^2)$, but this approach remains applicable only to problems of moderate size. In the linear cluster separation framework utilised here substantial improvements can be achieved.

The most critical choice in clustering algorithms based on graph cuts is the determination of pairwise similarities. Similarities are typically defined as 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 $\mathbf{v}^\top \mathcal{X}$, 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 (**?**). The associated Normalised Cut Divisive Clustering (NCUTDC) algorithm is a computationally efficient divisive clustering algorithm relying on hyperplane separators, arising from the objectives,

$$\Phi(\mathbf{v}|\mathcal{X}) = \min_b \phi(\mathbf{v}, b|\mathcal{X}) \tag{9}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) := \text{NCut}(\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}). \tag{10}$$

As in the minimum density hyperplane case, the projection pursuit problem is more conveniently formulated as a minimisation problem. It is important to note that in (**?**) this algorithm 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 algorithm for a single hyperplane.

Example 5 *We return to the optidigits data set, and apply both spectral clustering and NCUTDC. We use the implementation of spectral clustering in the* **R** *package* **kernlab** *(***?***).*

```
> require(kernlab)
> data(optidigits)
>
> system.time(sol1 <- ncutdc(optidigits$x, 10))
> system.time(sol2 <- kernlab::specc(optidigits$x, 10))
> cluster_performance(sol1$cluster, optidigits$c)
   adj.rand    purity v.measure       nmi
  0.6554853 0.7870107 0.7171148 0.7171234
> cluster_performance(sol2, optidigits$c)
   adj.rand    purity v.measure       nmi
  0.3551385 0.4967972 0.4658001 0.4659070
```

NCUTDC runs orders of magnitude faster than spectral clustering, and achieves substantially higher performance.

It is important to note that because the projection index underlying NCUTDC uses similarities computed on the projected data, this algorithm cannot operate on an arbitrary graph or similarity matrix.

Modifying and validating a clustering solution

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 the validation of a clustering model 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. In previous sections we looked briefly at visualisations of both hyperplane partitions, as well as entire clustering models. Here we use these visualisations to identify important modifications and validate clustering solutions. A hierarchical clustering model can be modified either by pruning parts of the model when it is apparent that a single cluster was divided by the partition at an internal node, or by extending the model by splitting leaves which contain more than one cluster. A model can also be refined by changing the partition at an internal node via the modification of the parameters of the projection pursuit algorithm. Since this affects the entire sub-hierarchy extending from that node, the model is first pruned and then iteratively extended 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 6 *In this example we again revisit the optical recognition of handwritten digits data set (**?**). For illustrative purposes, we do not assume as we did in Example 2 that the number of clusters is known. We initially partition the data set into seven clusters using minimum density hyperplanes, and then visualise the solution using the* `tree_plot()` *function. This visualisation is shown in Figure 5.*

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

The first thing to notice is the partition at the node numbered 4. There is a clear 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. The default initialisation which gave rise to this projection was the first principal component of the data assigned to the node. 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 additional initialisations. When multiple initialisations are considered within one of the projection pursuit algorithms, all the resulting solutions are stored for inspection. Within a hierarchical clustering algorithm only the best solution is included in the model.*

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

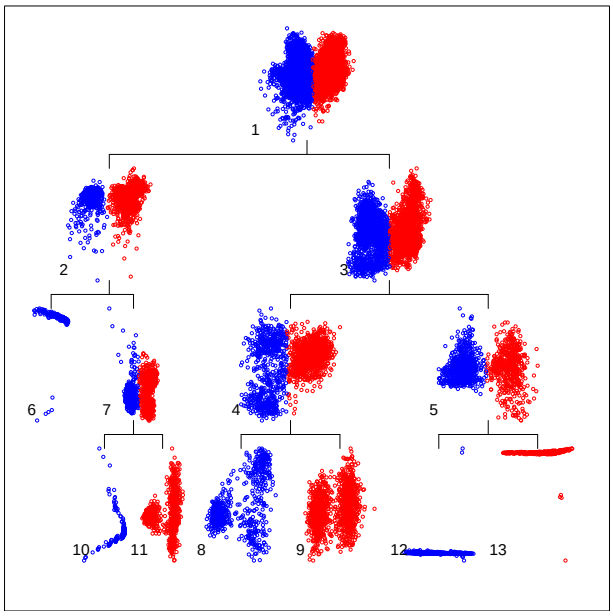
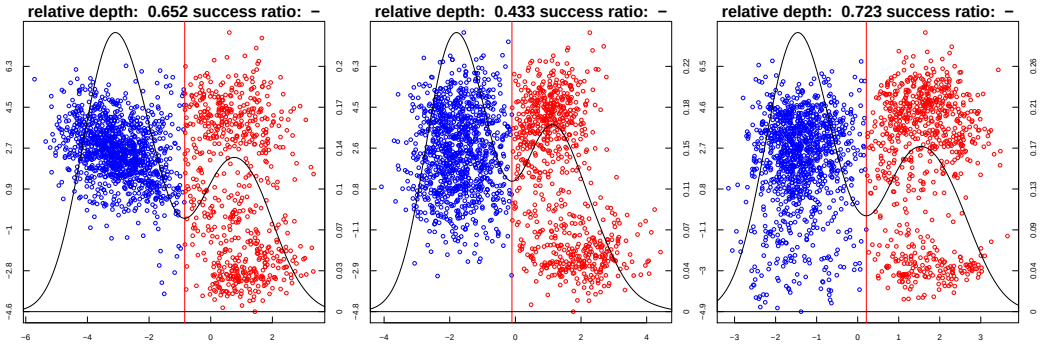


Figure 5: Initial clustering of optidigits data set through MDDC with 7 clusters.



(a) First principal component (b) Second principal component (c) Third principal component

Figure 6: Three potential binary partitions for node 4 based on three different initialisations.

```
> hp_plot(node4_alt[[1]], node4_x)
> hp_plot(node4_alt[[2]], node4_x)
> hp_plot(node4_alt[[3]], node4_x)
```

The outputs can be seen in Figure 6. The solution obtained by initialising on the third principal component (Figure 6(c)) appears superior to the current solution (Figure 6(a)). 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. The arguments accepted by both functions are shown in Table 5. Both 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 the example below we set the initial projection vector to the third principal component by using the optional argument* `v0`*. 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 7.*

```
> sol <- tree_prune(sol, 4)
> v0 <- function(X) matrix(eigen(cov(X))$vectors[,3], ncol = 1)
> sol <- tree_split(sol, 4, v0 = v0)
> tree_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 closely inspected using

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.
`...`	(optional. `tree_split` only) Any collection of optional arguments associated with the clustering algorithm which produced the solution `sol`.

Table 5: Arguments accepted by `tree_prune` and `tree_split` functions.

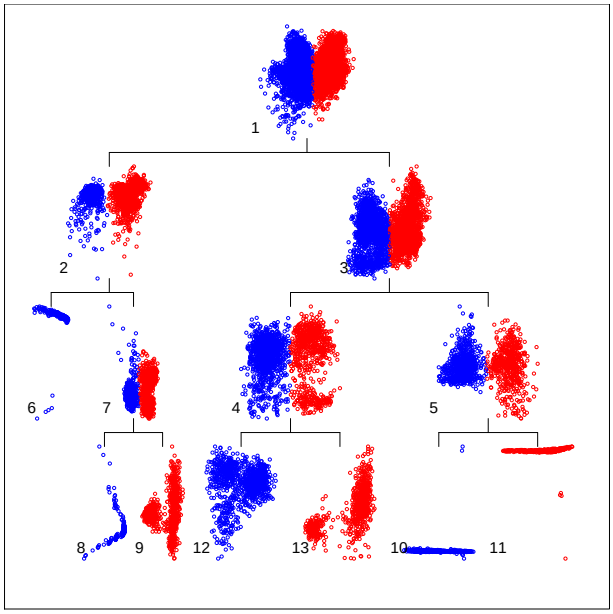


Figure 7: Clustering solution after the modification of the partition at node 4.

the function `node_plot(sol,node)`*. This produces an output similar to the* `hp_plot` *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 8.*

```
> node_plot(sol, 10)
> node_plot(sol, 12)
```

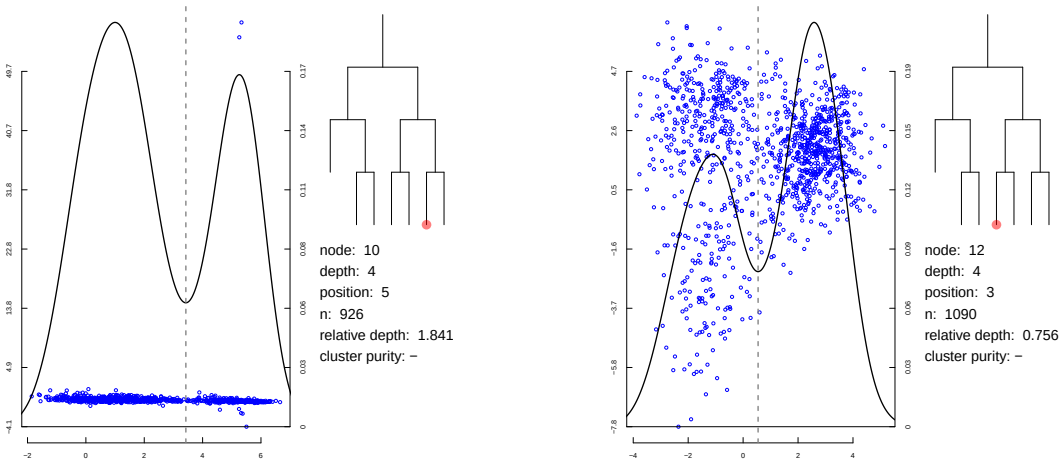
Both nodes show strong evidence of multiple clusters, through their highly bimodal projected densities. Moreover, the potential partition at the leaf node numbered 12 does not appear to offer an optimal partition of the clusters. Alternative initialisations can be considered as above for node 4, leading 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, where the high variance projection orthogonal to $\mathbf{v}_{\text{opt}}$*, depicted in the vertical axis in Figure 8(a), is influenced by a few outlying points, making 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)
> v0 <- function(X) matrix(eigen(cov(X))$vectors[,2], ncol = 1)
> sol <- tree_split(sol, 12, v0 = v0)
> 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.



(a) Visualisation of node 10 (b) Visualisation of node 12

Figure 8: Visualisations of individual nodes in cluster hierarchy.

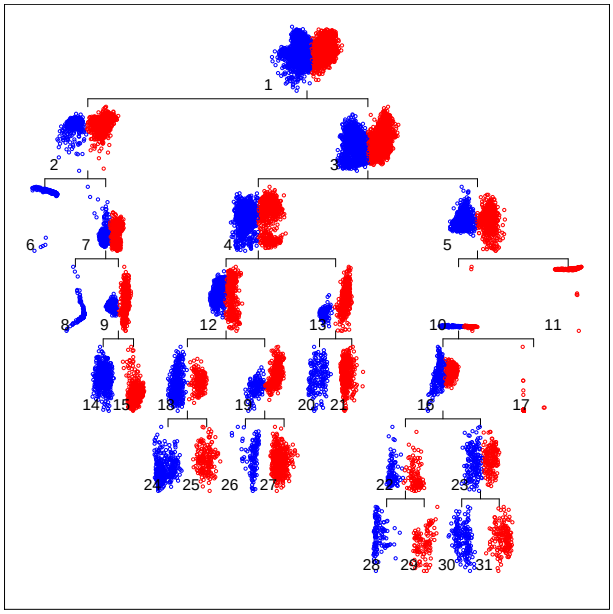


Figure 9: Final clustering solution after modifications. There is no evidence of individual clusters being split, and no evidence of clusters not identified in the model.

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

The complete solution at this stage can be seen in Figure 9. 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.

```
> 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.

Extensions

In this section we consider two extensions of the projection pursuit methods discussed in the previous sections. First we discuss how to estimate large margin hyperplanes for clustering through either `mdh()` or `ncuth()`, and subsequently consider the binary partition of data sets whose clusters cannot be linearly separated.

Maximum margin hyperplanes for clustering

Maximum Margin Clustering (MMC) (**?**), 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 (**?**).

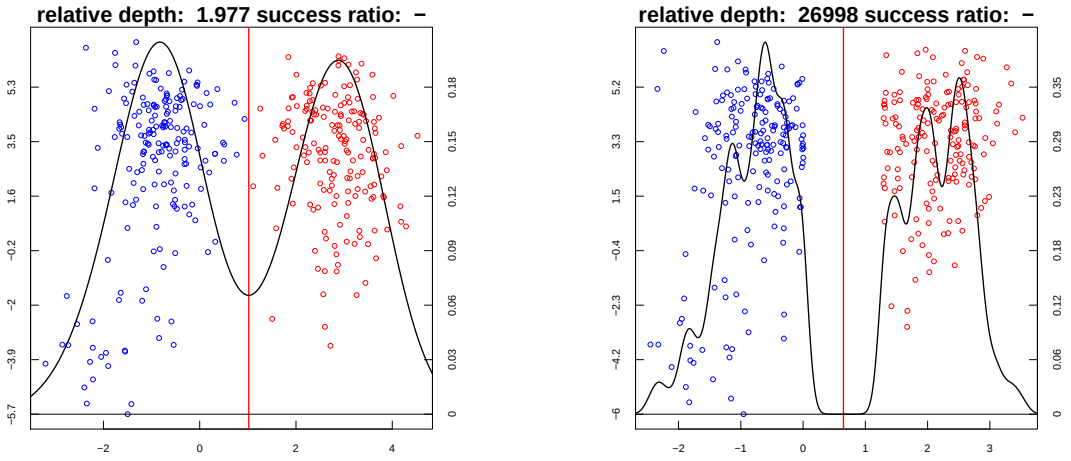
The minimum density hyperplane and the minimum normalised cut hyperplane converge to the maximum hard margin hyperplane through the data, as the bandwidth (scaling) parameter is reduced towards zero (**??**). A simple and effective approach to obtain large margin hyperplanes for clustering is to apply either of the above methods for a decreasing sequence of bandwidth/scaling parameters.

Example 7 *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* **?**. *We only use the test set of the optidigits data set to render our results directly comparable to those reported by* **?**. *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`. *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[[1]]$v, bandwidth = hp[[1]]$params$h*0.9,
        alphamin = hp[[1]]$params$alpha)
>   if(hp[[1]]$v%*%hp_new[[1]]$v>(1-1e-10)) break
>   else hp <- hp_new
> }
>
> hp_plot(hp0[[1]], x39)
> hp_plot(hp[[1]], x39)
>
> 1-cluster_performance(hp[[1]]$cluster, optidigits$c[ids])[2]
      purity
  0.02479339
```

The first and last MDHs obtained from the above example are illustrated in Figure 10. 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 10(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 (***?***, Table IV) report the clustering error of k-means, kernel k-means, normalised cut spectral clustering (***?***), generalised maximum margin clustering (***?***), 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 (***?***) achieves a much larger error of 0.3609*



(a) MDH for default bandwidth

(b) MDH for smallest bandwidth considered

Figure 10: Large margin hyperplane separator for the optdigits 3-9 problem obtained by applying `mdh()` for a decreasing sequence of bandwidths

on this data set.[1] *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 separators using Kernel PCA

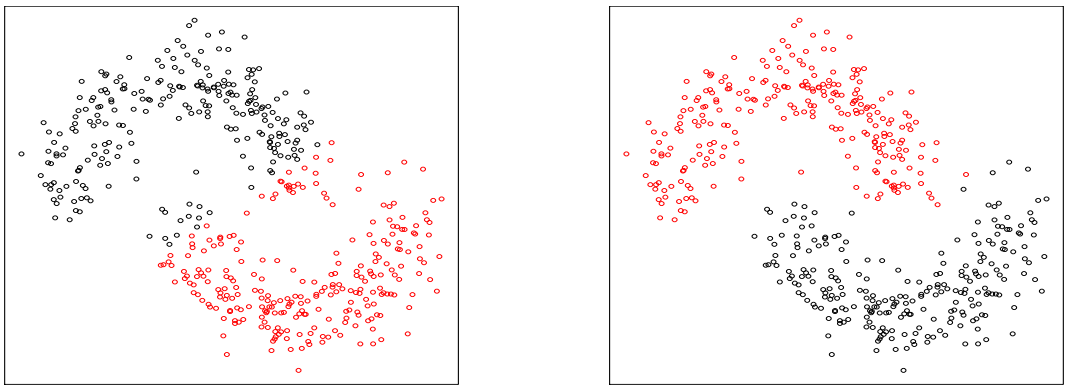
The main limitation of linear cluster separators is their inability to accurately separate clusters whose convex hulls overlap. Although many real world applications involve data in which clusters can be separated well by hyperplanes, this is not always the case, especially in lower dimensional examples. Using Kernel Principal Components Analysis (KPCA) (**?**) the data can be embedded in a high-dimensional feature space in which clusters are linearly separable. Hyperplanes in the feature space correspond to nonlinear cluster boundaries in the original space. Below we illustrate how combining KPCA with a linear cluster separator enables the separation of clusters which cannot be linearly separated within the input space. We then apply this approach to the application of identifying different hand-written digits based on pen trajectories. We selected the hyperparameter for the kernel embedding 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 8 *We employ a data set containing two clusters, each in the shape of a half moon, arranged so that they cannot be separated by a hyperplane (**?**). We use KPCA to embed the data in a high-dimensional feature space in which the clusters are linearly separable.*

```
> 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))
> x <- x + matrix(0.18*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[[1]]$cluster, main = "Linear Separator")
> plot(x, col = sol2[[1]]$cluster, main = "Non-linear Separator")
```

Example 9 *We next apply the KPCA embedding to the Pen Based Recogntion of Handwritten Digits data set (**?**). Due to the size of the data set, which includes 10992 observations, we build the kernel embedding using a compressed version of the data set obtained through k-means.*

[1] Code for this method is available at https://sites.google.com/site/binzhao02/



(a) Linear Separator (b) Nonlinear Separator

Figure 11: Nonlinear cluster separation after preprocessing the data through Kernel PCA

```
> require(kernlab)
> set.seed(1)
> data(pendigits)
> kp <- kernlab::kpca(kmeans(pendigits$x, 200)$centers, kernel = "rbfdot",
        kpar = list(sigma = 0.1))
> x2 <- predict(kp, pendigits$x)
> sol1 <- ncutdc(pendigits$x, 10)
> sol2 <- ncutdc(x2, 10)
> cluster_performance(sol1$cluster, pendigits$c)
   adj.rand    purity v.measure       nmi
  0.6180774 0.7801128 0.7097384 0.7097736
> cluster_performance(sol2$cluster, pendigits$c)
   adj.rand    purity v.measure       nmi
  0.7130409 0.8265102 0.7940943 0.7941021
```

The performance is substantially improved by first embedding the data in the feature space and then applying NCUTDC.

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 enables significant improvements over PCA and other 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. 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.

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