OPC: Optimal Projections for Clustering

An open source MATLAB and Octave library for Dimensionality Reduction for Clustering

Nicos G. Pavlidis

Contents

1

Preface

Optimal Projections for Clustering (OPC) is an open source MATLAB and Octave library that implements clustering methods that seek the optimal low dimensional subspace to identify (or separate) clusters. This is a recognised issue in high-dimensional data clustering [12], but also occurs whenever irrelevant features, or correlations among subsets of features exist, or when clusters are defined in different subspaces. These characteristics cause the spatial data structure to become less informative about the underlying clusters. To correctly identify clusters algorithms need to simultaneously solve two interrelated problems, (i) identify the subspace in which clusters can be distinguished, and (ii) associate observations to clusters [12].

OPC focuses on methods which seek low dimensional subspaces that are optimal with respect to a given clustering objective. This distinguishes the methods in this library from generic dimensionality reduction techniques, which determine the low dimensional embedding by optimising an objective function that is not related to any clustering criterion [27]. Consequently, although generic dimensionality reduction techniques have been successfully applied in numerous clustering applications, they are not guaranteed to find a subspace that preserves (or reveals) the cluster structure. A further limitation of using generic dimensionality reduction techniques for clustering is that the low dimensional data embedding typically varies substantially across different methods. This is a problem from a user's perspective because there is no principled approach to select one embedding over the others, even if the user knows the type of clusters they are seeking.

Numerous implementations of clustering algorithms and generic dimensionality reduction techniques are available in the MATLAB programming language. The MATLAB toolbox for dimensionality reduction, is the most extensive MATLAB toolbox for this purpose that we are aware of. A number of open source MATLAB implementations of robust PCA (which is not included in the previous toolbox) like fastRPCA, are also available.

To the best of our knowledge OPC is the only library in the MATLAB programming language that implements dimensionality reduction methods explicitly designed for clustering. The methods included optimise clusterability criteria motivated from k-means, normalised graph cut, spectral, and density clustering. Partitioning, as well as divisive hierarchical clustering algorithms are provided. To render the algorithms accessible to non-expert users, all the implemented algorithms require minimal input from the user, and employ parameter settings recommended in the literature. However, all the parameters of each algorithm can be modified by specifying optional arguments.

A very appealing characteristic of clustering algorithms that perform dimensionality reduction internally is that they enable the visualisation of the clustering model. Such visualisations constitute an invaluable component of the process of validating a clustering model. OPC provides a common interface across algorithms to produce such visualisations. The optimal projection matrix is one of the outputs of all partitioning algorithms in the library. This matrix jointly with the cluster assignment vector, can be readily used to visualise the clusters. The divisive hierarchical algorithms return the cluster hierarchy as an object of the cluster tree, `ctree`, class. This class enables the visualisation of the binary partition at each node of the cluster hierarchy (tree). It also enables the modification of the clustering model by pruning subtrees, or partitioning leaves of the current model. The binary partition at any node of the `ctree` object can also be

altered by modifying the arguments of the projection pursuit algorithm used to produce it. An extensive example of validating and modifying the clustering model using these functionalities is provided in Section 6.

To render OPC extensible, we include an interface that allows the user to create divisive hierarchical algorithms that have all the aforementioned features by providing as input only a projection pursuit function for clustering. Section 8 illustrates how to obtain the well known bisecting k-means algorithm [23] through this approach, as well as how to create a divisive hierarchical version of the LDA-k-means algorithm [3].

2

Installation

To install OPC a C and C++ compiler is required. Detailed installation instructions for MATLAB and Octave are provided in the following sections. OPC depends on the following two open source libraries that are included in the library:

1. A cluster tree class, called `ctree`, which is a modification of the `tree` class by Jean-Yves Tinevez.

2. The improved Fast Gauss Transform [14].

In Octave after each restart you also need to load the statistics and optim packages (see last paragraph of Section 2.2 for more details). The `setpath` script also performs this task.

2.1 MATLAB

OPC requires the statistics and optimization MATLAB toolboxes. To use OPC it is also necessary to compile the C++ FIGtree library [14]. The most recent version (0.9.3) is included in the OPC library in the directory `src/libs/figtree-0.9.3/`. For instructions on how to compile the library refer to the `README.md` file within this directory, or equivalently to the FIGtree GitHub page. Effectively, the process involves two steps. First the C++ library needs to be compiled. It is recommended to use the GCC compiler, which is straightforward to install in Linux and Mac.

```
$ # Note this is taking place in the command line not within MATLAB or Octave
$ # Assuming OPC_DIR is the path to the directory where OPC was uncompressed
$ cd (OPC_DIR)/src/libs/figtree-0.9.3/
$ make all
```

In Microsoft Windows GCC is also recommended but there are instructions on how to compile the library using Microsoft Visual Studio. Moreover, the SourceForge FIGtree page contains releases with Windows binaries.

After the successful compilation of the C++ library the MATLAB interface to FIGtree needs to be compiled, as well as two C++ functions that compute one-dimensional Gaussian kernel density estimators and their derivatives. (A C++ compiler is necessary for this step.) The script `install` in the root OPC directory is included to simplify this process. Beware that this script assumes that the path to the FIGtree library is `(OPC_DIR)/src/libs/figtree-0.9.3/`. If the FIGtree is in a different location then edit the `install.m` script to provide the correct path to the corresponding folder. In MATLAB:

```
>> cd('(OPC_DIR)/')
>> setpath
>> install
```

2.2 GNU Octave

OPC uses object oriented programming and contains `classdef` classes. These are only supported after the Octave 4.0 release. OPC is therefore incompatible with previous versions.

OPC requires the statistics package, and the non-linear optimization package, optim. Both packages can be found at the extra packages for Octave repository. Detailed instructions to install and load extra packages in Octave are provided in relevant documentation page. Once these are installed they need to be loaded after every restart of Octave. The `setpath` script located in the root OPC directory loads these packages and adds the OPC directory and its subfolders to the Octave search path.

To complete the installation of OPC it is necessary to compile the FIGtree library, and two C++ functions that compute one-dimensional Gaussian kernel density estimators and their derivatives. Refer to the description of this process in the previous section.

2.3 Documentation

The user guide for OPC is provided at the GitHub OPC repository. The documentation is also available online at this link. A function reference for OPC, created through the M2HTML documentation system, is available at this link.

The file `reproduction_script.m` in the root OPC directory reproduces every example in this document in (the results reported in this document are obtained using MATLAB 2018a). To recreate all the figures (in PNG format) uncomment the lines which call the `print` function, which follow calls to the `plot` and `nplot` functions.

2.3.1 Contributing

The preferred way to contribute to OPC is to fork the main repository on GitHub. Detailed instructions are available through the online Git documentation. Please use GitHub issues to file bug reports and feature requests.

3

Implemented Methods

The OPC library contains implementations of the following clustering algorithms:

1. Principal Direction Divisive Partitioning (PDDP) [2]
2. Density-enhanced Principal Direction Divisive Partitioning (dePDDP) [25]
3. Minimum Density Divisive Clustering (MDDC) [18]
4. Maximum Clusterability Divisive Clustering (MCDC) [6]
5. Linear Discriminant Analysis k-means (LDA-k-means) [3]
6. Dimensionality Reduction for Spectral Clustering (DRSC) [16, 17]
7. Minimum Spectral Connectivity Projection Pursuit (SCPPDC) [7]
8. Minimum Normalised Cut Divisive Clustering (NCUTDC) [5]
9. Bisecting k-means [23] and a projection pursuit version of this algorithm based on LDA-k-means. The implementation of these algorithms is used as an illustration of how the user can extend the library.

We next provide a brief exposition of the implemented methods. Throughout we assume that the dataset consists of n vectors in the d–dimensional Euclidean space, $\mathcal{X} = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$. The rows of the data matrix, $X \in \mathbb{R}^{n\times d}$, correspond to data points. The projection matrix is defined as $V \in V_q(\mathbb{R}^d) = \{A \in \mathbb{R}^{d\times q} \,|\, A^\top A = I_q\}$, where $1 \leqslant q < d$, and I_q is the identity matrix in q dimensions. When there is a single projection vector we use lower case notation, v, to denote vectors that lie on the d–dimensional unit–sphere, $v \in \mathbb{S}^d = \{x \in \mathbb{R}^d \,|\, \|x\|_2 = 1\}$.

3.1 Principal Direction Divisive Partitioning and density-enhanced version

The most widely used approach to combine dimensionality reduction and clustering is to project the d–dimensional dataset onto the first q principal components. Principal Component Analysis (PCA) can be considered the most popular projection pursuit method. PCA can be formulated either as maximising the variance of the projected data, or minimising the squared reconstruction error [10]. PCA is the only projection pursuit method that has a closed form solution. For a given choice of q, the projection index is optimised by the q eigenvectors associated with the q largest eigenvalues of the data covariance matrix. Although there is no guarantee that the subspace spanned by any $q < d$ principal components preserves the cluster structure, this approach has been found to be effective in a plethora of applications [12].

PDDP and dePDDP are two divisive clustering algorithms that recursively project (subsets of) the data onto their first principal component. PDDP bi-partitions the data by splitting at the mean of the projections, while dePDDP constructs a one-dimensional kernel density estimator (KDE) and splits at the lowest local minimum. Effectively, both algorithms bi-partition each cluster using a separating hyperplane,

$$H(v, b) = \{x \in \mathbb{R}^d \,|\, v^\top x = b\},$$

where v is the unit-length vector normal to the separating hyperplane, and $b \in \mathbb{R}$ is the displacement of the separating hyperplane from the origin.

3.2 Minimum Density Divisive Clustering

The Minimum Density Divisive Clustering (MDDC) algorithm recursively bi-partitions the data using Minimum Density Hyperplanes (MDHs) [18]. An MDH avoids intersecting high density clusters by seeking the hyperplane that minimises the integral of the empirical probability density function along its surface,

$$\hat{I}(v, b) = \int_{H(v,b)} \hat{p}(x) \mathrm{d}x.$$

If a density estimator that uses isotropic Gaussian kernels is employed,

$$\hat{p}(x) = \frac{1}{n(2\pi h^2)^{d/2}} \sum_{i=1}^{n} \exp\left\{-\frac{\|x - x_i\|^2}{2h^2}\right\},$$

then $\hat{I}(v, b)$ can be estimated through the one-dimensional density estimator,

$$\hat{I}(v, b) = \frac{1}{n\sqrt{2\pi h^2}} \sum_{i=1}^{n} \exp\left\{-\frac{(b - v^\top x_i)^2}{2h^2}\right\}.$$

The ability to compute $\hat{I}(v, b)$ through the above equation enables us to estimate MDHs. The MDH is the solution to the following optimisation problem,

$$v^\star = \min_{v \in \mathbb{S}^d} f(v), \tag{3.1}$$

$$f(v) = \min_{b \in \mathbb{R}} \left\{\hat{I}(v, b) + c_0 \max\{0, \mu_v - \alpha\sigma_v - b, b - \mu_v - \alpha\sigma_v\}^{1+c_1}\right\}. \tag{3.2}$$

The second term in the last equation, $c_0 \max\{0, \mu_v - \alpha\sigma_v - b, b - \mu_v - \alpha\sigma_v\}^{1+c_1}$, is a penalty function that ensures that the optimal MDH is within α standard deviations, σ_v from the mean of the projected data, μ_v. Such a constraint is necessary since it is always possible to obtain hyperplanes with density arbitrarily close to zero if the hyperplane is sufficiently far from the centre of the data. The terms c_0 and c_1 are constants whose values are fixed (see [18]), while α is adaptively set by the projection pursuit algorithm.

The most computationally expensive task in the estimation of MDHs is the minimisation of the one dimensional kernel density estimator, for each v. This is achieved by first evaluating $f(v)$ on a grid of m points, to bracket the location of the minimiser, and then applying bisection to compute the minimiser at the desired accuracy. The Fast Gauss Transform [14] reduces the computational cost of the first step, and by far most computationally expensive step, from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$. Bisection requires $\mathcal{O}(-\log_2 \varepsilon)$ iterations to locate the minimiser with accuracy ε.

3.3 Maximum Clusterability Divisive Clustering

Variance ratio clusterability [32] is a measure of strength or conclusiveness of the cluster structure in a dataset. Let $\{C_i\}_{i=1}^k$ denote a k-way partition of the data set $\mathcal{X}$, and $\mu(\cdot)$ the function that takes as input

a set of data points and returns their mean vector. Then the variance ratio clusterability of $\mathcal{X}$ is defined as,

$$\max_{C_1,\ldots,C_k} \frac{\sum_{j=1}^{k} |C_j| \, \|\mu(\mathcal{X}) - \mu(C_j)\|^2}{\sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu(C_j)\|^2},$$
$$\text{such that: } \mathcal{X} = \bigcup_{i=1}^{k} C_i, \text{ and } C_i \cap C_j = \emptyset, \forall i \neq j.$$

It is easy to show that the partition that maximises the variance ratio clusterability also minimises the k-means objective function,

$$\min_{C_1,\ldots,C_k} \sum_{i=1}^{n} \min_{j=1,\ldots,k} \|x_i - \mu(C_j)\|_2^2,$$
$$\text{such that: } X = \bigcup_{i=1}^{k} C_i, \text{ and } C_i \cap C_j = \emptyset, \forall i \neq j.$$

The Maximum Clusterability Divisive Clustering (MCDC) [6] algorithm recursively bi-partitions the data by identifying the unit-length vector that maximises the variance ratio clusterability index of the projected data. The resulting hyperplane separators are called Maximum Clusterability Hyperplanes (MCHs). For any $C \subsetneq X$ the binary partition of X is given by $\{C, X \setminus C\}$. Let Σ the data covariance matrix, $\Sigma = n^{-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^\top$. Then the projection index for a given v is given by,

$$f(v) = \max_{C \subset X} \frac{|C| \left(v^\top (\mu(C) - \mu(\mathcal{X}))\right)^2 + (n - |C|) \left(v^\top (\mu(\mathcal{X} \setminus C) - \mu(\mathcal{X}))\right)^2}{\frac{n}{n-1} v^\top \Sigma v + \sum_{x_i \in C} \left(v^\top (x_i - \mu(C))\right)^2 + \sum_{x_i \in \mathcal{X} \setminus C} \left(v^\top (x_i - \mu(\mathcal{X} \setminus C)\right)^2}.$$

For the special case of one dimensional data it is possible to identify the set $C^\star$ that optimises the variance ratio clusterability in $\mathcal{O}(n \log n)$ time. This is feasible because in this case the set $C^\star$ satisfies, $C^\star = \{x \in X \,|\, v^\top x < b\}$ for some $b \in \mathbb{R}$.

3.4 Linear Discriminant Analysis k–Means

The LDA-k-means algorithm [3] exploits the connection between Linear Discriminant Analysis (LDA) and k-means clustering. To see this we first need to introduce some notation. Without loss of generality assume that the data are centred at zero. Define the total scatter, the between-cluster scatter, and the within-cluster scatter matrices as,

$$S_t = \sum_{i=1}^{n} x_i x_i^\top,$$
$$S_b = \sum_{j=1}^{k} |C_j| \, \mu(C_j) \mu(C_j)^\top,$$
$$S_w = \sum_{j=1}^{k} \sum_{x_i \in C_j} (x_i - \mu(C_j))(x_i - \mu(C_j))^\top,$$

respectively. Then the k-means objective function can be expressed as,

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu(C_j)\|^2 = \min_{C_1,\ldots,C_k} \operatorname{tr}(S_w), \tag{3.3}$$

where $\mathrm{tr}\,(\cdot)$ is the trace operator. Since $S_t = S_w + S_b$, minimising $\mathrm{tr}\,(S_w)$ is equivalent to maximising $\mathrm{tr}\,(S_b)$. In LDA the cluster assignment (class labels) are assumed known and the algorithm seeks the orthonormal matrix $V \in V_{k-1}(\mathbb{R}^d)$ that,

$$\max_V \frac{\mathrm{tr}\left(V^\top S_b V\right)}{\mathrm{tr}\left(V^\top S_w V\right)}, \quad \text{s.t.} \quad V^\top V = I_{k-1}. \tag{3.4}$$

The fact that both LDA and k-means aim to minimise the within class (cluster) scatter and maximise the between class scatter motivates the LDA-k-means algorithm. LDA-k-means is a two step iterative algorithm. In the first step the $(k-1)$-dimensional linear subspace in which the data are projected is fixed, and the objective is to identify the cluster assignment that maximises the ratio of the between class scatter to the within class scatter of the projected data:

$$\max_{C_1,\ldots,C_k} \frac{\mathrm{tr}\left(V^\top S_b V\right)}{\mathrm{tr}\left(V^\top S_w V\right)} = \max_{C_1,\ldots,C_k} \frac{\mathrm{tr}\left(V^\top (S_t - S_w) V\right)}{\mathrm{tr}\left(V^\top S_w V\right)} = \max_{C_1,\ldots,C_k} \frac{\mathrm{tr}\left(V^\top S_t V\right)}{\mathrm{tr}\left(V^\top S_w V\right)} - 1 \Rightarrow$$

$$\min_{C_1,\ldots,C_k} \mathrm{tr}\left(V^\top S_w V\right) = \min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|V^\top x_i - V^\top \mu(C_j)\|^2$$

The k-means algorithm is applied to solve this problem. In the second step the cluster (class) assignment is considered fixed and the objective is to identify the orthonormal matrix V that maximises the objective function. LDA is used to solve this problem. The only element of the algorithm that remains unspecified is the initial choice of V. In [3] it is recommended to initialise V to the first $(k-1)$ principal components.

The convergence of LDA-k-means algorithm is not discussed in [3]. Since no implementation of k–means is guaranteed to identify the globally optimal solution to the problem in Eq. (3.3) the two-step procedure in LDA-k-means can fail to converge, or even to to monotonically improve the objective function, Eq. (3.4). It is important to note that the are multiple formulations of the LDA problem. We make use of the formulation which maximises the eigenvalues of the matrix $S_t^{-1} S_b$. Although this formulation is not guaranteed to converge, it benefits substantially from the available closed form for the LDA step.

3.5 Minimum Normalised Cut Divisive Clustering

The approach to clustering based on graph theory associates clusters with strongly connected components of a graph, $G = (V, E)$. The set of vertices, V, corresponds to the data set, $V = \mathcal{X}$, while edge weights represent pairwise similarities between them [28]. The graph, G, can be equivalently represented through an adjacency, or similarity, matrix, $A \in \mathbb{R}^{n \times n}$, defined as,

$$A_{ij} = k(x_i, x_j), \tag{3.5}$$

where $k(\cdot)$ is a kernel function (with the Gaussian kernel being the most frequent choice). The degree of a vertex is defined as,

$$d_i = \sum_{j=1}^{N} A_{ij}.$$

For any subset $C \subset X$, the volume of C is defined as:

$$\mathrm{vol}(C) = \sum_{x_i \in C} d_i.$$

A graph cut is a partition of the vertices of a graph into k pairwise disjoint components. The *minimum normalised cut* of a graph is the solution to the following optimisation problem,

$$\min_{C_1,\ldots,C_k} \mathrm{NCut}(C_1, \ldots, C_k) = \min \sum_{j=1}^{k} \frac{\mathrm{cut}\,(C_j, \mathcal{X} \setminus C_j)}{\mathrm{vol}(C_j)}, \tag{3.6}$$

where the cut of binary partition is defined as,

$$\text{cut}\left(C_j, \mathcal{X} \setminus C_j\right) = \frac{1}{2} \sum_{j:\, x_j \in C_j} \sum_{i:\, x_i \in \mathcal{X} \setminus C_j} A_{ji}$$

Dividing the value of the cut for each binary partition with the cluster volume (normalisation) discourages the optimal solution from containing partitions (clusters) with very few vertices (data points), which is undesirable from a clustering perspective. However, it also renders the problem NP-hard.

The minimum normalised cut divisive clustering (NCUTDC) [5] is an algorithm that recursively bi-partitions the data using hyperplanes. NCUTDC relies on the fact that if the data points are projected onto a vector, $v \in \mathbb{S}^d$, and the Laplace kernel is used to compute the pairwise similarities,

$$A_{ij} = \exp\left(-\left|v^\top x_i - v^\top x_j\right| / \sigma\right),$$

then the hyperplane that optimises the minimum normalised cut criterion (which is equivalent to a binary partition of the form, $\mathcal{X} = \{C^\star, \mathcal{X} \setminus C^\star\}$ where $C^\star = \{x \in \mathcal{X} \,|\, v^\top x < b\}$ for some $b \in \mathbb{R}$) can be computed in log-linear time. The projection index for this method is:

$$\begin{aligned} f(v) &= \min_{b \in \mathbb{R}} \text{NCut}\left(C_{v,b}, C^c_{v,b}\right), \text{ where,} \\ C_{v,b} &= \{v^\top x \,|\, v^\top x < b,\ x \in \mathcal{X}\}, \\ C^c_{v,b} &= \{v^\top x \,|\, v^\top x > b,\ x \in \mathcal{X}\}, \end{aligned}$$

It is important to stress 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.

3.6 Dimensionality Reduction for Spectral Clustering

Spectral clustering is a family of methods that solve a continuous relaxation of the original NP-hard graph cut problem in (3.6) [4, 21, 28]. To describe the continuous relaxation of the normalised graph cut problem it is first necessary to define the *graph Laplacian* matrix:

$$L_n = I_n - D^{-1/2} A D^{-1/2},$$

where A is the adjacency matrix defined in Eq. (3.5), $D = \text{diag}(d_1, \ldots, d_n)$ is the degree matrix, and I_n is the identity matrix in n dimensions. For a given k-way partition of $\mathcal{X}$ into $\{C_1, \ldots, C_k\}$, define the $(n \times k)$ matrix T as:

$$T_{ij} = \begin{cases} \sqrt{d_i / \text{vol}(C_j)}, & \text{if } x_i \in C_j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

The minimum normalised cut problem can be formulated as [21]:

$$\min_{C_1, \ldots, C_k} \text{tr}\left(T^\top L_n T\right), \quad \text{subject to: } T^\top T = I_k, \text{ and } T \text{ as defined as in Eq. (3.7)},$$

or equivalently [15]:

$$\max_{C_1, \ldots, C_k} \text{tr}\left(T^\top D^{-1/2} A D^{-1/2} T\right) \quad \text{subject to: } T^\top T = I_k, \text{ and } T \text{ as defined as in Eq. (3.7)}.$$

The continuous relaxation which gives rise to spectral clustering allows T to be any orthonormal matrix in $\mathbb{R}^{n \times k}$ [28]. The normalised cut spectral clustering problem is thus expressed as [21]:

$$\min_{T \in \mathbb{R}^{n \times k}} \text{tr}\left(T^\top L_n T\right), \quad \text{subject to: } T^\top T = I_k.$$

The solution to the above optimisation problem is the matrix T whose columns are the k eigenvectors of L_n that correspond to the k smallest eigenvalues (or equivalently the k eigenvectors that correspond to the k largest eigenvalues of $D^{-1/2}AD^{-1/2}$).

The Dimension Reduction Spectral Clustering (DRSC) [16] algorithm aims to identify the projection matrix, V, that maximises the sum of the k largest eigenvalues of the $D^{-1/2}AD^{-1/2}$, where both the similarity and degree matrices, A and D respectively, are computed from the projected data, $XV \in \mathbb{R}^{n\times q}$. The sum of the k largest eigenvalues of $D^{-1/2}AD^{-1/2}$ is a measure of spectral connectivity, with larger values indicating more separable clusters.

$$\max_{U,V} \quad \text{trace}(U^\top D^{-1/2}AD^{-1/2}U), \tag{3.8}$$

$$\text{s.t.} \quad U^\top U = I, \tag{3.9}$$

$$A_{ij} = k(\|V^\top x_i - V^\top x_j\|), \tag{3.10}$$

$$V^\top V = I. \tag{3.11}$$

The above formulation is equivalent to minimising the sum of the k smallest eigenvalues of L_n. It is clear that for a given V the matrix U that maximises the trace in (3.8) has columns given by the k eigenvectors associated with the k largest eigenvalues of $D^{-1/2}AD^{-1/2}$ (or equivalently the k smallest eigenvalues of L_n).

DRSC employs a two-stage algorithm to solve the above optimisation problem. In the first stage V is fixed and the k eigenvectors of $D^{-1/2}AD^{-1/2}$ are computed to obtain U. In the second stage, U is fixed and for each column of V gradient ascent is performed to maximise the trace in (3.8). The projection index for this method can therefore be written as:

$$\begin{aligned} f(V) = \sum_{n-k+1}^{n} \lambda_j \left(D^{-1/2}AD^{-1/2}\right) = \sum_{j=1}^{k} \lambda_j \left(L_n\right), \\ \text{s.t. } A_{ij} = k(\|V^\top x_i - V^\top x_j\|), \\ V^\top V = I, \end{aligned}$$

where $\lambda_j(\cdot)$ denotes the j-th smallest eigenvalue of its matrix argument. However, in the computation of the gradient of $f(V)$ in the second stage of the algorithm, the degree matrix D is considered fixed, and therefore the influence of V on the degrees of the vertices is ignored. The gradient estimated by assuming D is fixed is not the gradient of the overall objective function. There is furthermore no guarantee that this direction estimated through this approach is even an ascent direction for the objective function. This can cause DRSC to fail to converge.

3.7 Spectral Connectivity Projection Pursuit Divisive Clustering

Spectral Connectivity Projection Pursuit Divisive Clustering (SCPPDC) [7] recursively bi-partitions the data by projecting onto the linear subspace that minimises the second smallest eigenvalue of the normalised graph Laplacian, L_n. In more detail, for each projection matrix, V, the data are first projected onto V, and then the similarity matrix, A, the degree matrix, D, and the normalised graph Laplacian, L_n, are computed from the projected data. The projection index employed by this method is the value of the second smallest eigenvalue of $L_n(V)$:

$$f(V) = \lambda_2 \left(L_n(V)\right) + \omega V^\top V,$$

where the second term, $\omega V^\top V$, with $\omega \geqslant 1$, is a penalty function included to ensure that the columns of V are orthogonal. SCPPDC therefore employs the same objective function as DRSC for the case when $k = 2$. However, unlike DRSC, the projection pursuit algorithm employed by SCPPDC is globally convergent because the algorithm uses the gradient of the overall objective function. Furthermore in each gradient descent step SCPPDC updates the entire projection matrix, rather than optimising each column of V separately as in DRSC.

3.8 Bisecting k–means

Bisecting k-means [23] is a well known divisive clustering algorithm originally proposed for document clustering, that recursively applies 2-means. Since 2-means can be viewed as separating the clusters by first projecting the data onto the vector connecting the two centroids and then assigning observations to clusters based on their location relative to the mid-point between the two centroids, this algorithm can be considered as performing dimensionality reduction.

This algorithm is included in the library as an example of how a user can extend the functionality of the library by specifying a function that performs projection pursuit for clustering (see Section 8). A projection pursuit version of bisection k-means based on the LDA-k-means algorithm is also discussed in Section 8.

4

General Information

The algorithms in OPC are designed to require minimum input from the user, so that non-expert users can directly apply the implemented methods. However all the parameters of each algorithm can be controlled by specifying the optional parameters through name, value pairs. All clustering algorithms (except `drsc`) require only the specification of the data matrix and the number of clusters. (`drsc` also requires the specification of the scaling parameter of the Gaussian kernel used to compute similarities, because the cross–validation approach recommended in [16] is computationally very expensive.)
Complete documentation for each function is available using `help`, or through the HTML function reference (see Section **??**).

Output Arguments

The partitioning algorithms, `ldakmeans` and `drsc`, return as the first three output arguments:

1. `idx`: A vector of cluster assignments, `idx` $\in \{1, \ldots, k\}^n$
2. `V`: The projection matrix
3. `fval`: The value of the projection index at termination

Both `ldakmeans` and `drsc` return additional output arguments that are specific to each algorithm.
All divisive clustering algorithms in OPC return the following two output arguments:

1. `idx`: A vector of cluster assignments, `idx` $\in \{1, \ldots, k\}^n$
2. `t`: A `ctree` object that represents the cluster hierarchy in the form of a binary tree

Cluster Hierarchy class `ctree`

The class `ctree` implements a generic interface to interact with cluster hierarchies that enables the visualisation, and modification of the hierarchy. The properties of the `ctree` are:

1. `Node`: A cell array each element of which stores a binary separator (node) of the cluster hierarchy. Different clustering algorithms can return objects of different classes to represent the binary separator.
2. `Parent`: A vector containing the index of the parent of each node. In particular, if `j = Parent(i)` then `Node{j}` is the parent of `Node{i}`.
3. `method`: String specifying the clustering algorithm used to create the `ctree` object.
4. `cluster`: A vector containing the cluster assignment from this clustering model.
5. `data`: The data matrix on which the clustering model has been estimated

6. `params`: A structured array containing all the parameters employed by the algorithm.

The first two properties are necessary to describe a cluster hierarchy, while the cluster assignment, `cluster` is the main output from any clustering algorithm. The last two properties `data` and `params` are useful to visualise and modify the clustering model.

The main methods for the `ctree` class are:

1. `get(id)`: Returns the binary separator at `Node{id}`. The structure of these will be discussed next.
2. `findleaves()`: Returns the indices of the leaves of the `ctree` object (that is the locations in the `Node` cell array). By construction the leaves correspond to the final clusters.
3. `nnodes()`: Returns the number of nodes in the tree.
4. `depth()`: Returns the depth of a tree.
5. `perf(labels)`: This function takes as input the actual cluster assignment and computes the quality of the binary partition at each node of the `ctree` object. For internal nodes quality is measured in terms of the success ratio [18], while on leaves it is measured in terms of purity [35]. The function returns a new `ctree` object.
6. `plot(labels,colours)`: This function produces a visualisation of the entire cluster hierarchy. The user can provide as optional input the actual cluster assignment, `labels`. In this case the observations in the figure produced are coloured according to the actual cluster membership. The user can also determine the colour used for each cluster, by setting the last optional argument `colours`. (`colours` must be a $k\times 3$ matrix specifying an RGB triplet for each one of the k clusters.)
7. `nplot(id, labels,colours)`: This function produces a visualisation of the binary separator in `Node{id}`. The interpretation of the optional arguments, `labels` and `colours` is the same as for the `plot()` function described above.
8. `prune(id)`: Prunes the subtree rooted at `Node{id}`. (After this operation `Node{id}` is a leaf node.) The function returns a new `ctree` object.
9. `split(id,varargin)`: Splits a leaf node of the tree using the algorithm used to originally create this `ctree` object with the same parameter settings. The user can modify any of the settings of the clustering algorithm by specifying the optional arguments. The syntax is identical to that used by the divisive algorithm used to create the specific `ctree` object initially. The function returns a new `ctree` object.

The last two functions, `prune` and `split` are extensively discussed in the section on model validation and modification (Section 6). Visualisation of clustering models obtained by different algorithms are discussed in almost every example in Section 5. We next describe the objects that describe binary separators (elements of the `Node` cell array in the `ctree` class.

Binary separators

Different clustering algorithms produce objects of different classes to describe the binary separator employed. The properties discussed below are shared by all classes of binary separators in OPC, and suffice to understand how to extract all the meaningful information from such objects.

1. `v`: Stores the optimal projection matrix/ vector. The data (subset) is first projected onto `v` and then bi-partitioned.
2. `fval`: Value of the projection index for associated projection pursuit algorithm. For instance for an MDH the projection index is the value of the density on the optimal split point. For NCUTDC `fval` is the value of the minimum normalised cut. If `fval` is $\pm\infty$ this signifies that the projection pursuit algorithm has failed to identify a valid separator for this dataset (see examples in Section 7.1).

3. `params`: A structured array containing all the parameter values for the projection pursuit algorithm employed to estimate this binary separator.

4. `idx`: A vector containing the row-numbers (indices) of the observations allocated to this binary separator in the `ctree` object. Observations allocated to the left child have a negative sign while observations allocated to the right child have a positive sign.

5. `tree_params`: A structured array containing information that is only used when displaying a `ctree` object on the terminal.

In the next code snippet we provide a very simple example of how to extract information from `ctree` object, making use of all the above information. We first compute a cluster hierarchy with PDDP. (Each algorithm will be discussed in much greater detail in the next section.)

```
>> % load an example dataset
>> load('datasets/optidigits.mat');
>> % Estimate a divisive hierarchical model using PDDP
>> [idx,t] = pddp(X,10);
>> % Produce visualisation (output not shown presently)
>> plot(t);
>>
>> % Get separator at left child of root node
>> node2 = t.get(2);
>> % Identify which observations were allocated Node 2
>> obs = abs(node2.idx);
>> % Identify how many observations were allocated to Node 2
>> length(node2.idx);
>> % Projection matrix for this node (1st PC for this data subset)
>> node2.v
>> % Value of projection index: PDDP uses total_scatter by default
>> node2.fval
```

Overview of Main Optional Arguments

The following list of optional input arguments is not exhaustive but is meant to illustrate the main parameters that are common across most of algorithms. For a comprehensive list of all the optional arguments for each algorithm use `help` from the command line interface, or refer to the HTML function reference.

verb All algorithms (except `gppdc`) accept the optional input argument `verb`. If `verb` is non-zero then progress information is displayed during the clustering process. Algorithms that perform projection pursuit illustrate a two-dimensional visualisation of the current candidate solution at each step. This slows down the progress of the algorithm considerably, especially for divisive clustering algorithms, and especially on Octave (the MATLAB `pca` function is much faster than the Octave `princomp` function). On the other hand such visualisation enable the user to obtain important insights as to whether or not the algorithm locates meaningful partitions. We therefore recommend setting `verb=0` (which is the default) when estimating a cluster hierarchy, and enabling this option when more insight is required for particular splits in the tree (see Section 6).

labels, colours The `labels` optional argument allows the user to define the actual cluster assignment. This information is used only to assess performance and to improve the visualisation (see `verb` option above), by assigning colours to observations based on the actual cluster. The argument `colours` allows the user to specify the colour used to display observations from each cluster. These two options are very useful when performing exploratory data analysis.

split_index This optional argument is relevant for divisive algorithms only. It must be set to a handle associated with a function that takes three inputs:

1. `v`: the currently optimal projection matrix/ vector,
2. `X`: a data matrix containing the observations from the cluster that is currently split,
3. `par`: a structured array that contains all the parameters of the algorithm,

and returns the value of a splitting criterion. At each step all divisive clustering algorithms split the current cluster that has the highest `split_index` value. OPC functions that can be used as split indices are:

1. `reldepth`: Relative depth of separating hyperplane [18].
2. `total_scatter`: Total scatter [2].

v0 This optional argument determines the initial projection matrix used to perform projection pursuit. For divisive clustering algorithms it must defined as a handle associated with a function that takes two inputs:

1. `X`: a data matrix containing the observations from the cluster that is currently split,
2. `par`: a structured array that contains all the parameters of the algorithm,

and returns the value a projection matrix/ vector. OPC functions that can be used as to set the initial projection matrix/ vector are:

1. `pcacomp`: Returns principal component vectors in the order specified by the user.
2. `mc_v0`: Returns vector connecting centroids of 2-means algorithm.

For partitioning algorithms, like `drsc`, and for algorithms that estimate a single separating hyperplane, like `mdh`, `mch` and `ncuth`, `v0` can be defined as fixed projection matrix/ vector.

minsize Most algorithms enable the user to specify the minimum number of observations in a cluster.

maxit, ftol Algorithms that use numerical methods to identify the optimal projection matrix accept as arguments `maxit`: the maximum number of iterations, and `ftol` which specifies the function tolerance (that is the minimum change in the value of the objective function value in two consecutive iterations, required to allow the numerical algorithm to continue).

5

Using OPC

In this section we present examples of using the algorithms implemented in OPC. Our aim is to illustrate not only the syntax and the main options for each algorithm, but also to highlight how a user can obtain insights about the clustering models produced.

We use the optical recognition of handwritten digits (`optidigits`) dataset from the UCI repository [13]. This dataset is included in the library and combines the training and test datasets specified in UCI. Observations correspond to vectorised images of handwritten digits 0–9. The images have been compressed to 8×8 pixels resulting in a 64 dimensional dataset. The data are preprocessed by removing two features which have the same value for all observations, and standardising the remaining features. The resulting dataset consists of 5620 data points in 62 dimensions. The data matrix and the true cluster assignments are stored in `optidigits.mat`.

```
>> % Load optiditigs dataset: X := data matrix, labels := true cluster assignment
>> load('datasets/optidigits.mat');
```

5.1 Clustering without Dimensionality Reduction

We first apply k-means and spectral clustering to the full dimensional data. We use the built-in `kmeans` function in MATLAB and Octave, and the OPC function `spclust` that implements the spectral clustering algorithm of Ng. et al. [15]. The function `spclust` by default performs spectral clustering using the Gaussian kernel to estimate the similarity (kernel) matrix. If the scale parameter is not specified then `spclust` uses the local scaling rule recommended in [31]. We have found that this rule is effective in a plethora of applications and hence incorporate it as the default option in `spclust`. (MATLAB code for the complete self–tuning spectral clustering algorithm [31], which also estimates the number of clusters, can be found here.)

Note that the default initialisation of `kmeans` in both MATLAB and Octave is through the k-means++ algorithm [1], and therefore the output of the `kmeans` function is not deterministic. This also affects spectral clustering because the final cluster assignment is performed through `kmeans`. To ensure that results are reproducible we set the random number seed. All the results reported in this document were obtained on MATLAB 2018a.

```
>> % set random number seed to ensure reproducibility (MATLAB only)
>> % in Octave use rand('seed',x) and randn('seed',x)
>> rng(987)
>> % k-means clustering
>> km = kmeans(X,10);
>> % Spectral clustering
>> % (In Octave the pdist function can fail for this dataset size)
>> sc = spclust(X,10);
```

5.1.1 Assessing Clustering Performance

The `cluster_performance` function implements four well-established external cluster evaluation measures: Purity [35], Normalised Mutual Information [24], adjusted Rand index [8], and V-measure [19].

```
>> % Evaluate cluster performance
>> % (performance will vary on consecutive kmeans executions due to
>> % random initialisation. Performance is reported for illustration)
>> cluster_performance(km, labels)

ans =

  struct with fields:

      Purity: 0.4683
         NMI: 0.5145
     AdjRand: 0.3371
    Vmeasure: 0.5101

>> cluster_performance(sc, labels)

ans =

  struct with fields:

      Purity: 0.6521
         NMI: 0.6191
     AdjRand: 0.4817
    Vmeasure: 0.6189
```

5.2 Dimensionality Reduction as pre-processing step prior to Clustering

The most common approach to handle datasets containing irrelevant or correlated features is to first perform dimensionality reduction, and then apply a clustering algorithm. Although this approach has frequently produced satisfactory results, there is no guarantee that this will be the case as highlighted in the influential review paper on high–dimensional clustering [12]. The dimensionality reduction MATLAB toolbox contains a large number of generic dimensionality reduction methods. At present we will use PCA, which is most widely used in practice and for which a built-in implementation is available in both MATLAB and Octave. The function `pcacomp` in OPC takes as input a data matrix and a vector of integers that specifies which principal components should be returned. The first output argument of `pcacomp` is a matrix whose columns are the principal component vectors in the order these are specified in the second argument to the function. We first choose to project the data onto the first 9 principal components. This choice is made because for a k-means clustering $(k-1)$ dimensions are sufficient.

```
>> % Estimate the first 9 PCs
>> V = pcacomp(X,[1:9]);
>> % Perform clustering
>> kmPCA = kmeans(X*V,10);
>>
>> % (On this lower dimensional dataset the pdist function in
>> % Octave can work even if it failed on the original dataset)
>> scPCA = spclust(X*V,10);
>> % Assess performance
>> cluster_performance(kmPCA, labels)

ans =

  struct with fields:
```

```
      Purity: 0.6842
         NMI: 0.6444
     AdjRand: 0.5201
    Vmeasure: 0.6442


>> cluster_performance(scPCA, labels)

ans =

  struct with fields:

      Purity: 0.6381
         NMI: 0.6472
     AdjRand: 0.4695
    Vmeasure: 0.6466
```

Another common approach is to use the first q principal components that retain a pre–specified proportion of the variance. Different values for this proportion have been suggested. In the present dataset preserving a proportion very close to one requires retaining all 62 dimensions. We therefore use a threshold of 0.9 which is attained by projecting onto the first 33 principal components.

```
>> % Estimate the first 33 PCs
>> V = pcacomp(X,[1:33]);
>> % Perform clustering
>> kmPCA = kmeans(X*V,10);
>> scPCA = spclust(X*V,10);
>> % Assess performance
>> cluster_performance(kmPCA, labels)

ans =

  struct with fields:

      Purity: 0.6295
         NMI: 0.6106
     AdjRand: 0.4566
    Vmeasure: 0.6100

>> cluster_performance(scPCA, labels)

ans =

  struct with fields:

      Purity: 0.6290
         NMI: 0.6307
     AdjRand: 0.4442
    Vmeasure: 0.6289
```

Projecting onto the first $k-1$ principal components allows k-means to substantially improve its performance with respect to all evaluation measures, but for spectral clustering some performance measures improve while others do not. If the dimensionality of the projection subspace is instead 33 then k-means still performs better than on the full dimensional data but worse than when q was set to nine. The performance of spectral clustering is also worse when 33 rather than 9 principal components are used.

5.3 Linear Discriminant Analysis k–means

The first algorithm we discuss in which dimensionality reduction is an integral part of the clustering process is LDA–k–means [3]. The function `ldakmeans` implements this algorithm. As discussed in Section 3.4, LDA–k–means is a partitioning algorithm that aims to identify the optimal low-dimensional projection, and cluster assignment through a two-stage iterative algorithm. In the first stage, the projection matrix is fixed

and k-means is applied on the projected data to identify the optimal cluster/ class assignment. In the second stage, the labels are treated as given and Linear Discriminant Analysis (LDA) is used to identify the optimal linear subspace. The inability to identify the global optimum of the k-means objective function, and the variability due to the random initialisation of the centroids in the `kmeans` function, cause the algorithm to not be guaranteed to converge, or even to improve the value of the objective function over consecutive iterations.

We apply `ldakmeans` twice to illustrate this point. The first execution yields a very good performance on this dataset. The performance is substantially lower in the second application of the algorithm. Enabling visualisation also illustrates that the algorithm does not monotonically improve the projection index, which is defined as the ratio of the trace of the between-cluster and within-cluster scatter matrices.

```
>> % Setting verb=1 enables visualisation
>> [ldakm,U,fval] = ldakmeans(X,10,'verb',1);
>> cluster_performance(ldakm, labels)

ans =

  struct with fields:

      Purity: 0.8096
         NMI: 0.7808
     AdjRand: 0.6772
    Vmeasure: 0.7807

>> % Visualisation using actual labels
>> [ldakm,U,fval] = ldakmeans(X,10,'labels',labels,'verb',1);
>> cluster_performance(ldakm, labels)

ans =

  struct with fields:

      Purity: 0.6916
         NMI: 0.6663
     AdjRand: 0.5509
    Vmeasure: 0.6661
```

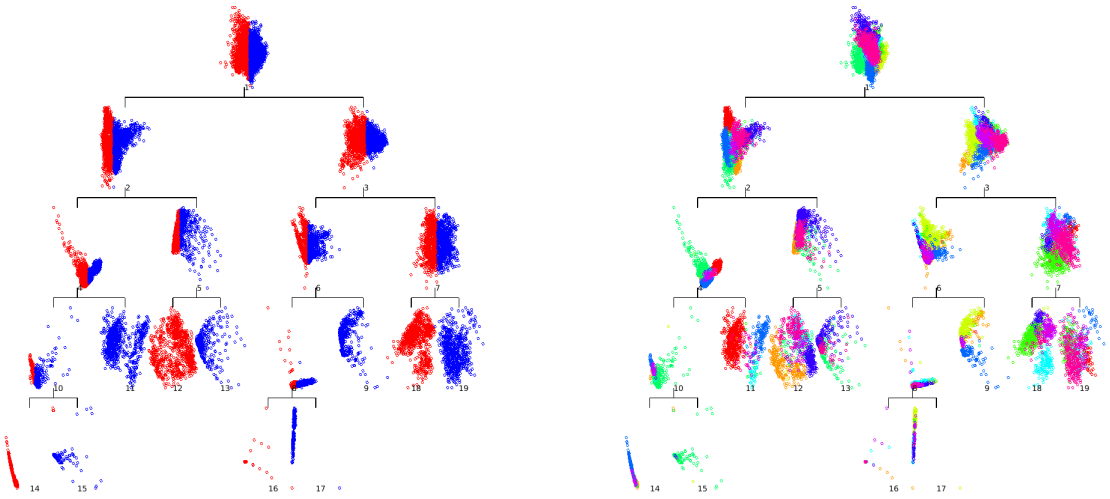
5.4 Divisive Clustering based on Principal Components

We next proceed to discuss divisive hierarchical clustering algorithms. In this section we discuss PDDP and dePDDP; two methods that recursively project the data (subset) onto their first principal component, and split at a point along the projection vector. Note that this approach enables these algorithms to identify clusters defined in different subspaces. This is not feasible in the standard approach of first performing dimensionality reduction, and then clustering.

5.4.1 Principal Direction Divisive Partitioning

The `pddp` function implements the PDDP algorithm.

```
>> % Perform PDDP with default settings
>> [idx1,t1] = pddp(X,10);
Clusters: 1 2 3 4 5 6 7 8 9 10
>> % Evaluate cluster performance
>> cluster_performance(idx1, labels)

ans =

  struct with fields:

      Purity: 0.5171
```



(a) Labels unspecified

(b) Labels specified

Figure 5.1: Visualisation of clustering model in a `ctree` object without and with the specification of the true cluster labels (left and right respectively). In the latter case the default, `hsv`, MATLAB `colormap` is used.

```
          NMI: 0.4324
     AdjRand: 0.2842
    Vmeasure: 0.4324
```

The second output argument is an object of the `ctree` class that stores the cluster hierarchy. The predicted clusters can also be inferred from this `ctree` object using the method `tree2clusters`:

```
>> sum(idx1 ~= tree2clusters(t1))

ans =

     0
```

5.4.2 Visualisation of Clustering Models in OPC

A very appealing characteristic of OPC is that it enables the user to visualise the clustering model produced by any divisive hierarchical algorithm in the library. The user can plot both the entire hierarchy, or specific nodes. The `plot` function in the `ctree` class produces visualisations of the entire hierarchical model:

```
>> % Plot cluster hierarchy
>> plot(t1);
>> % The optional argument 'labels' causes observations
>> % to be coloured according to the actual cluster label
>> plot(t1,labels);
>> % The user can further specify which colours will be used for each cluster
>> plot(t1,labels, lines(10));
```

The output of the first two calls to the `plot()` function is illustrated in Figure 5.1. Each scatterplot in the figure depicts the data subset assigned to the corresponding node in the hierarchy projected into a two-dimensional subspace. In the case of algorithms that utilise one-dimensional projections the horizontal axis corresponds to the optimal projection vector, $v^\star$, while the vertical axis is the direction of maximum

variance orthogonal to $v^\star$. If the projection matrix contains two or more columns the `plot` function selects the two vectors along which the projected data exhibit the highest variance. In the present case, because PDDP projects each data subset onto the first principal component the vertical axis corresponds to the second principal component.

If the optional argument `labels` is not specified, red and blue colours are used to indicate the observations assigned to the left and right child of the node, as shown in the left sub-figure in Figure 5.1. If this argument is specified then the cluster label determines the colour of each point in the figure. The user can also define the colour for each cluster, by specifying a $(k \times 3)$ matrix, whose i–th row is the RGB triplet used to colour the i–th cluster.

Clearly the visualisation which uses the actual cluster labels to colour observations is very informative about how the actual clusters are partitioned at each level of the hierarchy. However, what is more important for unsupervised learning is that the visualisation of the cluster hierarchy without using the actual labels also allows the user to draw valuable insights about the validity of the binary partitions at each node. Such insights are crucial in the process of validating a clustering model, unless very strong assumptions about the types of distributions that give rise to the data are imposed. This issue will be explored further in Section 6, but for now we provide a few examples of insights that can be obtained from Figure 5.1:

1. The splits at depth 0, 1 and 2 of the cluster hierarchy appear to be intersecting dense regions. This could be an artefact of the visualisation because `plot` allocates the same space for the visualisation of each binary partition, and the number of observations allocated to these nodes is higher, compared to nodes closer to the leaves. Therefore closer inspection is required.

2. The leaf nodes (clusters) 11 and 18 appear to contain at least two dense clusters that are well separated by a sparse region. In leaf 11 the first principal component appears to be a direction which allows the separation of the two dense clusters. For the cluster at node 18 this does not appear to be the case.

3. The split at node 8 appears to separate a part of a large cluster and a few outliers from the rest of the data. In its right child, leaf node 17, the first principal component is largely affected by the presence of an outlier.

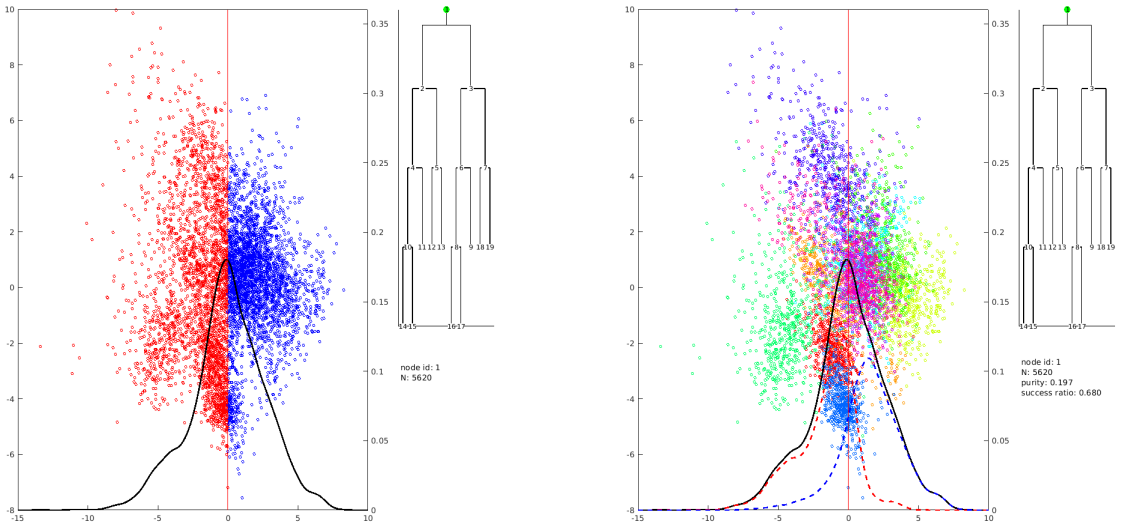
4. The principal component vector (direction of maximum variance) in leaf node number 17 is effectively determined by an outlier.

The `nplot` method of the `ctree` class enables the visualisation of the binary partition at any node of the cluster hierarchy. This function takes as mandatory inputs an object of class `ctree`, and the number of the node to be visualised. It also accepts as optional arguments `labels` and `colours`. The data are projected in a two-dimensional space using the same approach as in the `plot` function described previously. Let's first consider the root node of the tree.

```
>> % Plot partition at root node of the cluster hierarchy
>> nplot(t1,1);
>> % Plot partition at root node of the cluster hierarchy using actual labels
>> nplot(t1,1,labels);
```

Figure 5.2 (even without the labels) illustrates that the first split of the data intersects a very dense region. The premise that underlies PDDP, that if there are two well separated clusters in the data then these will be separable along the first principal component, does not appear to be valid in this example.

When the clustering algorithm uses a hyperplane to bi-partition the data at each node, the `nplot` function also illustrates a kernel density estimator constructed from the projection of the data onto the vector orthogonal to the separating hyperplane (black line). The scale of this function is illustrated on the right vertical axis. If the true clusters labels are specified, as in the second sub-figure in Figure 5.2, then each cluster is assigned to the halfspace that contains the majority of its observations. The overall density can then be seen as a two component mixture arising from the clusters assigned to the two halfspaces. The red and blue dashed lines illustrate these two component densities. This visualisation is intended to highlight the extend to which the separating hyperplane splits clusters in the data. The fact that in Figure 5.2 the two densities overlap considerably is a clear indicator that this hyperplane is splitting clusters.



(a) Labels unspecified

(b) Labels specified

Figure 5.2: Visualisation of binary partition at the root node of the cluster hierarchy without and with the specification of the true cluster labels (left and right respectively). The black line is the kernel density estimator of the data projected onto the vector that is orthogonal to the separating hyperplane

5.4.3 Quality of a Binary Partition

The *success ratio* is a measure of how effectively a binary partition separates at least one cluster from the rest, in datasets that contain two or more clusters [18]. This measure is always between zero and one, with higher values indicating better performance. To compute the success ratio for a specific binary partition we will use the information from the `ctree` object that stores the cluster hierarchy. To access a node of this object we can use the method `get` of the `ctree` class. Each node of the tree is a structured array. The `idx` element of this array contains the indices (row numbers in the data matrix `X`) of the observations allocated to this node. Observations allocated to the left child of a node have negative sign while those allocated to the right child have a positive sign. The code snippet below uses these properties to compute the success ratio at the root node.

```
>> % Get root node of cluster hierarchy
>> node1 = t1.get(1);
>> % Assess quality of binary partition through success ratio
>> success_ratio(sign(node1.idx), labels)
```

It is possible to evaluate the success ratio at each node of the cluster hierarchy, using the method `perf` of the `ctree` class.

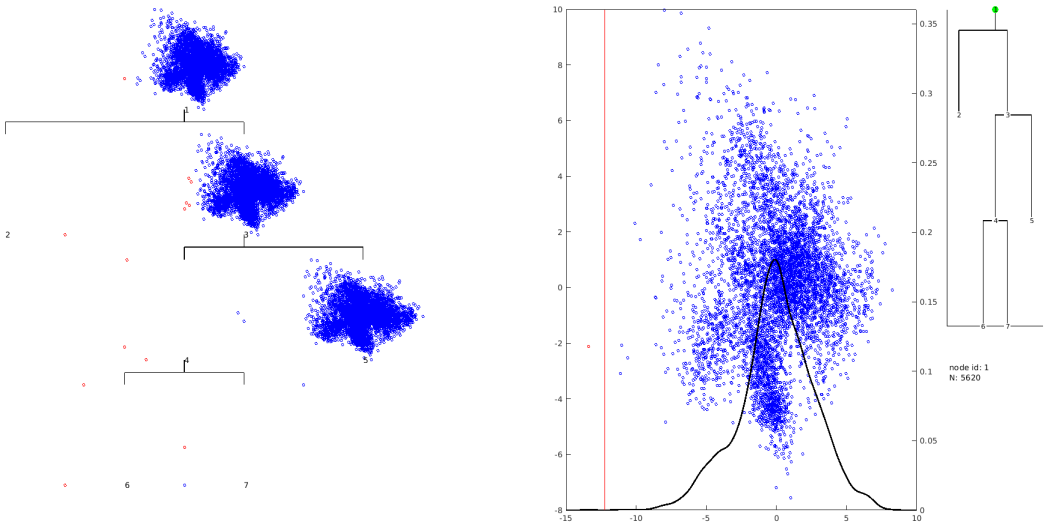
```
>> % Evaluate performance at each node of t1
>> t1 = perf(t1, labels)
```

The output of the above function is displayed on the terminal (but is too large to be presently included).

5.4.4 density-enhanced PDDP

We next consider whether the splitting criterion used by dePDDP will improve the clustering result.

```
>> [idx2, t2] = depddp(X,10);
```



(a) Cluster hierarchy

(b) First binary partition

Figure 5.3: Visualisation of cluster hierarchy and first binary partition performed by dePDDP

```
Clusters: 1 2 3 4 Divisive process terminated because no cluster can be split further
Clusters Identified: 4
```

The output above indicates that dePDDP did not identify 10 clusters. The algorithm fails to partition a cluster when the one-dimensional kernel density estimator constructed from the projection of the data onto the first principal component is unimodal. This is interpreted as an indication that the data belong to a single cluster.

Figure 5.3 illustrates the clustering model created by this algorithm, as well as the binary partition at the root node. From this figure we see why the algorithm failed to estimate 10 clusters and why the clustering model produced is not valid for this dataset. The one-dimensional data projection onto the first principal component always contains one or a few observations that are very distant from the rest. The estimated density has a very small mode around these observations and as a result at each bi-partition dePDDP splits one or a few outliers from the rest of the data. Around the bulk of the data the density estimator is unimodal.

A naive (and ineffective as we will see) approach to overcome this issue would be to impose a constraint on the minimum number of points in a cluster, by specifying the `minsize` option (available in all divisive hierarchical clustering algorithms in OPC).

```
>> [idx2, t2] = depddp(X,10,'minsize',10);
Clusters: 1 Divisive process terminated because no cluster can be split further
Clusters Identified: 1
```

The above message indicates that with a minimum cluster size of 10 the data cannot be split. This should not be surprising as we can see from Figure 5.3 that the density estimator has no local minima apart from the one at its left tail.

An important parameter in dePDDP is the bandwidth used in the density estimator. By default dePDDP estimates the bandwidth parameter, h, through the rule suggested by Silverman [22] for the approximation of multimodal univariate densities,

$$h = 0.9n^{-1/5}\hat{\sigma},$$

where $\hat{\sigma}$ is the standard deviation of the projected data. The user can modify this by providing as input a function handle. The associated function must take as input a data matrix and a structured array containing

the algorithm's parameters, and return a positive scalar. This function is used to estimate the value of the bandwidth parameter for each cluster. In the next code snippet we halve the value of the bandwidth.

```
>> % Define function handle for bandwidth selection (note 2nd input argument not used)
>> fh = @(x,p)(0.45*size(x,1)^(-0.2)*std(x* pcacomp(x,1)));
>> [idx2, t2] = depddp(X,10,'minsize',10, 'bandwidth', fh);
Clusters: 1 2 3 4 5 6 7 8 9 10
```

dePDDP with these settings can estimate 10 clusters, but a visualisation of the clustering model produced (which is omitted for brevity) illustrates that this model does not overcome any of the previously identified problems.

The inspection of the clustering models produced by PDDP and dePDDP strongly suggests that for this dataset dimensionality reduction by projecting onto the first principal component is not appropriate. In the next section we explore algorithms that attempt to identify the optimal projection vector with respect to explicit clustering criteria.

5.5 Divisive Clustering based on Optimal Hyperplanes

First we consider divisive clustering algorithms obtained by recursively bi-partitioning the data through hyperplanes that are estimated by identifying the optimal one-dimensional data projection. Three such methods will be discussed. The Maximum Clusterability Divisive Clustering (MCDC) algorithm attempts to identify the one-dimensional projection that optimises variance ratio clusterability; a criterion connected to the k–means objective function. Normalised Cut Divisive Clustering (NCUTDC) recursively bi-partitions the data using hyperplanes that minimise the value of the normalised graph-cut criterion. The last method, Minimum Density Divisive Clustering (MDDC) attempts to identify the hyperplane along which the integral of the estimated density is minimised.

For methods based on separating hyperplanes, OPC also includes a separate implementation of the underlying binary segmentation algorithm. To distinguish between the two, the names of functions that produce a complete divisive clustering model end in `dc`, while those that perform a binary partition using a hyperplane as a cluster separator, end in `h`. For example, the MCDC algorithm is implemented in the `mcdc` function, while the `mch` function estimates maximum clusterability hyperplane(s).

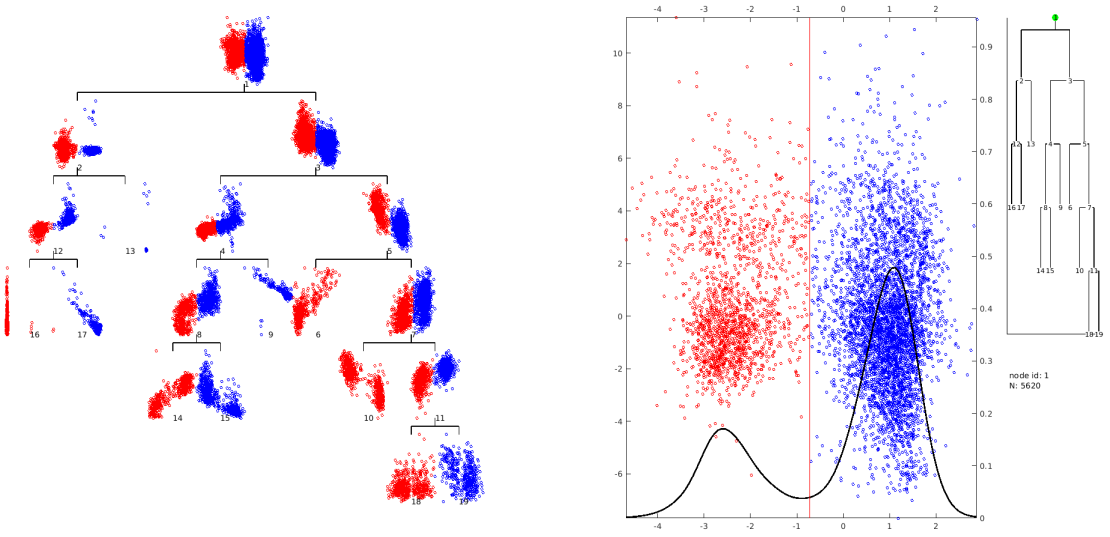
5.5.1 Maximum Clusterability Divisive Clustering

The MCDC algorithm is implemented in the function `mcdc`.

```
>> [idx3,t3] = mcdc(X,10);
>> cluster_performance(idx3,labels)

ans =

  struct with fields:

      Purity: 0.7710
         NMI: 0.7174
     AdjRand: 0.6371
    Vmeasure: 0.7174

>> plot(t3);
>> nplot(t3,1);
```

The clustering performance of this algorithm improves that of k–means applied on the full dimensional data, as well as that of k–means applied on the data projected onto the first 9 or 33 principal components.

It is important to note that the initialisation of the projection pursuit algorithm in MCDC is performed by applying 2-means on the data (subset) and computing the vector that connects the cluster centroids. Since by default `kmeans` in MATLAB and Octave uses the k–means++ algorithm [1], the output `mcdc` can differ in consecutive executions with identical input parameters.



(a) Cluster hierarchy

(b) First binary partition

Figure 5.4: Visualisation of cluster hierarchy and first binary partition performed by MCDC

The plot of the model hierarchy, and the first binary partition is provided in Figure 5.4. The figure shows that MCDC performs a much more sensible clustering, compared to PDDP and dePDDP. The visualisation of the binary partition at the root of the tree clearly shows that the projection pursuit algorithm has managed to identify a hyperplane that traverses a relatively sparse region and separates two dense clusters. This provides clear evidence of a clustering structure in the data which was not visible through PDDP and dePDDP (Figures 5.2 and 5.3, respectively). Inspecting the visualisation for the entire clustering model one observes that the majority of binary partitions at internal nodes appear to be separating effectively dense clusters. A number of leaves appear to contain more than one cluster and are natural candidates for further splitting. An extensive example of using the methods of the `ctree` class to validate and modify a clustering model is provided in Section 6.

We can examine whether an alternative initialisation would improve the final clustering model. In the example below we set the initial projection vector to be the first principal component. This is achieved by specifying the optional argument `v0`. For divisive hierarchical clustering algorithms this input argument must be set to a function handle. The associated function must accept two inputs: the data matrix, and a structured array containing the parameters of the algorithm.

```
>> % Function handle returns 1st PC
>> fh = @(x,p)(pcacomp(x,1));
>> [idx4,t4] = mcdc(X,10,'v0', fh);
>>
>> % Assess clustering performance
>> cluster_performance(idx4,labels)

ans =

  struct with fields:

      Purity: 0.7466
         NMI: 0.6727
     AdjRand: 0.5760
    Vmeasure: 0.6727
```

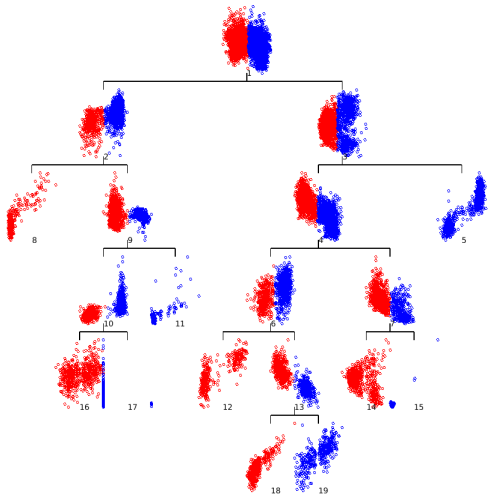


Figure 5.5: MCDC cluster hierarchy using first principal component to initialise projection pursuit algorithm

This initialisation causes a small deterioration in clustering performance. The visualisation of the resulting clustering model in Figure 5.5 reveals that the resulting clusters are more balanced, but there still remain clear indications that some leaves of the hierarchy contain more than one cluster, and should therefore be split.

5.5.2 Minimum Normalised Cut Divisive Clustering

We next produce a clustering model using the minimum Normalised Cut Divisive Clustering (NCUTDC) algorithm.

```
>> [idx5,t5] = ncutdc(X,10);
>>
>> % Assess clustering performance
>> cluster_performance(idx5,labels)

ans =

  struct with fields:

      Purity: 0.7810
         NMI: 0.6950
     AdjRand: 0.6200
    Vmeasure: 0.6949
```

NCUTDC performs better than spectral clustering applied on the full–dimensional data, and the datasets obtained after projection onto the first 9 and 33 principal components. This indicates that the dimensionality reduction approach is enabling the algorithm to identify better quality clusters. The resulting cluster hierarchy is illustrated in Figure 5.6.

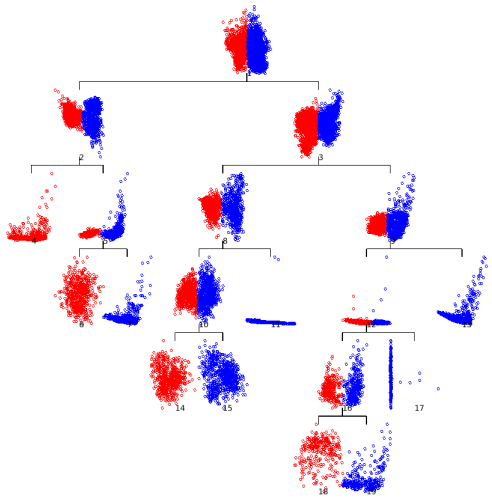


Figure 5.6: Cluster hierarchy obtained from NCUTDC using default parameter settings

5.5.3 Minimum Density Divisive Clustering

The interface of the `mddc` function is identical to the previous two functions, `mcdc`, and `ncutdc`. We thus thus directly execute the algorithm with the default settings

```
>> [idx6,t6] = mddc(X,10);
>>
>> % Assess clustering performance
>> cluster_performance(idx6,labels)

ans =

  struct with fields:

      Purity: 0.8110
         NMI: 0.7716
     AdjRand: 0.7054
    Vmeasure: 0.7716

>> nplot(t6,2);
```

Figure 5.7 provides a visualisation of the binary partition at node 2 of the cluster hierarchy. The dashed blue line in this figure corresponds to the projection index for MDH, $f(v)$ defined in Eq. (3.2). The scale of this function is depicted on the right vertical axis. Notice that $f(v)$ is identical to the kernel density estimator (black solid line) within the range of α standard deviations around the mean, and increases abruptly outside this range. The definition of the projection index in Eq. (3.2) ensures that the minimiser is always within α standard deviations of the mean of the data. A more extensive discussion of `mddc` is postponed until Section 6.

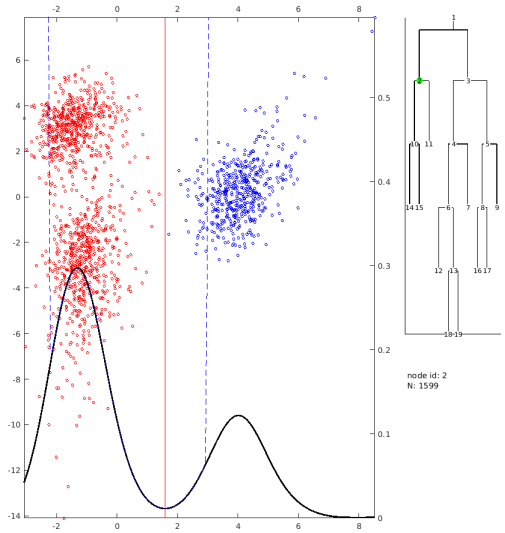


Figure 5.7: Binary partition at node 2 of cluster hierarchy produced by MDDC algorithm.

5.6 Spectral Clustering

In this section we describe two algorithms that attempt to identify the optimal linear subspace to perform spectral clustering: Dimensionality Reduction for Spectral Clustering (DRSC), and Spectral Clustering Projection Pursuit (which to retain the naming conventions in OPC we call Spectral Clustering Projection Pursuit Divisive Clustering, SCPPDC). Both are gradient-based algorithms that aim to minimise eigenvalue(s) of the normalised graph Laplacian. Unlike the divisive clustering algorithms discussed in previous sections the two methods presented in this section are not limited to one-dimensional projections and can be applied for any choice of the dimensionality of the projection space, q. This enables these methods to identify clusters whose convex hulls are not linearly separable.

Both DRSC and SCPPDC are computationally very expensive, because each function evaluation involves the computation of the similarity matrix, $\mathcal{O}(n^2)$, and its eigen-decomposition, $\mathcal{O}(n^3)$. The gradient ascent scheme for DRSC optimises each column vector of V separately, which increases the computational cost by a factor of q.

5.6.1 Dimensionality Reduction for Spectral Clustering

We will pre-process the data by micro-clustering before applying DRSC so that the algorithm terminates within a reasonable time. Micro-clustering [34] is effectively k-means clustering using a value of k that is much larger than the expected number of clusters in the data. We will use 200 micro-clusters as recommended for SCPPDC to render the results of the two algorithms comparable. In [16] it is suggested to select σ through 10-fold cross–validation using as performance measure the mean-squared error from the k-means step (last step of spectral clustering). This approach is computationally very expensive. Instead at present we use the bandwidth selection rule suggested in [7], and implemented in the function `scpp_def_sigma`. Finally, the dimensionality of the projection space, q, needs to be set. Following [16], the default setting for q in `drsc` is $q = k - 1$. We initialise DRSC using the first $k - 1$ principal component vectors, rather than a set of random orthonormal vectors. We have found that this initialisation improves performance considerably in the vast majority of cases. The user can modify this parameter specifying the optional argument `v0`. Since DRSC

is a partitioning algorithm, `v0` can be specified either as a function handle (as before), or as a projection matrix (which is not allowed for divisive algorithms). When the optional argument `v0` is set q is determined by the number of columns of the initial projection matrix.

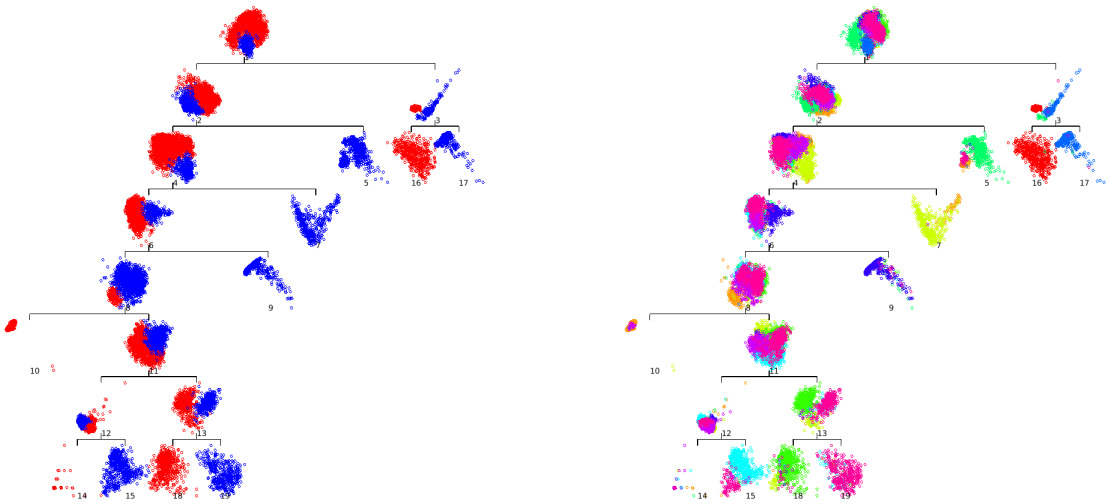
```
>> % the next algorithm is very slow. set seed to ensure reproducibility
>> % without having to run all the code up to this point
>> % In Octave use randn('seed',x); and rand('seed',x)
>> rng(56789);
>> % micro-clustering
>> [d2c,centroids] = kmeans(X,200);
>>
>> % Set scale parameter for Gaussian kernel to SCPPDC recommendation
>> sigma = scpp_def_sigma(X);
>>
>> % Execute drsc
>> [idx7,W,fval,sumD] = drsc(centroids,10,sigma);
>>
>> % Assign cluster labels to observations using
>> % the cluster labels of the micro-clusters
>> idx7 = reverse_assign(idx7, d2c);
>>
>> cluster_performance(idx7, labels)

ans =

  struct with fields:

      Purity: 0.1181
         NMI: 0.0585
     AdjRand: 4.2323e-04
    Vmeasure: 0.0274
```

DRSC terminates after exhausting the maximum number of iterations (the default value for which is 50) without converging. A plot of the projection index after each complete pass over all the columns of V (contained in the third output argument `fval`) indicates that the algorithm is not monotonically improving the objective function. The clustering quality of the identified solution is also poor.

5.6.2 Spectral Clustering Projection Pursuit Divisive Clustering

Next we apply the SCPPDC algorithm on the `optidigits` dataset. The syntax for this divisive hierarchical algorithm is identical to the previous algorithms. The function `scppdc` uses all the default values recommended in [7], and the user need only specify the data matrix and the number of clusters. A critical parameter for this algorithm (as for all spectral clustering methods) is the choice of the scale parameter, σ, of the Gaussian kernel that is used to compute pairwise similarities. By default this parameter is set through the function `scpp_def_sigma`. The user can modify this by setting the optional argument `sigma` to a function handle. The associated function must accept as arguments a data matrix and a structured array containing the parameters of the algorithm. Also very influential is the choice of the initial projection matrix. In `scppdc` the projection matrix is initialised to the first two principal components of the cluster being split. The user can modify this by specifying the optional argument `v0`. As in all divisive hierarchical algorithms, `v0` must be a handle to a function that returns a projection matrix and accepts as inputs the data matrix and a structured array containing all the parameters of the algorithm.

```
>> rng(1098765);
>> [idx8, t8] = scppdc(X,10);
Clusters: 1 2 3 4 5 6 7 8 9 10
>> % Visualisation
>> plot(t8)
>> plot(t8,labels)
>> % Performance assessment
>> cluster_performance(idx8,labels)
```



(a) Labels unspecified

(b) Labels specified

Figure 5.8: Cluster hierarchy produced by `scppdc` without and with the specification of labels

```
ans =

  struct with fields:

      Purity: 0.8384
         NMI: 0.7676
     AdjRand: 0.7007
    Vmeasure: 0.7676
```

Figure 5.8 illustrates the cluster hierarchy produced by `scppdc`. The only leaf node (cluster) that appears to contain more than one clusters is the node with number 4. The second sub-figure in Figure 5.8 verifies that this impression is correct. It is also clear that the ability to separate non-convex clusters in the two-dimensional projection space enhances the performance of this algorithm. This is verified by the cluster performance measures. SCPPDC achieves the highest performance out of all the considered methods on this dataset, and importantly improves substantially over spectral clustering applied on the full dimensional dataset, as well as on data projected onto the first q principal components, and DRSC.

6

Model Validation and Modification

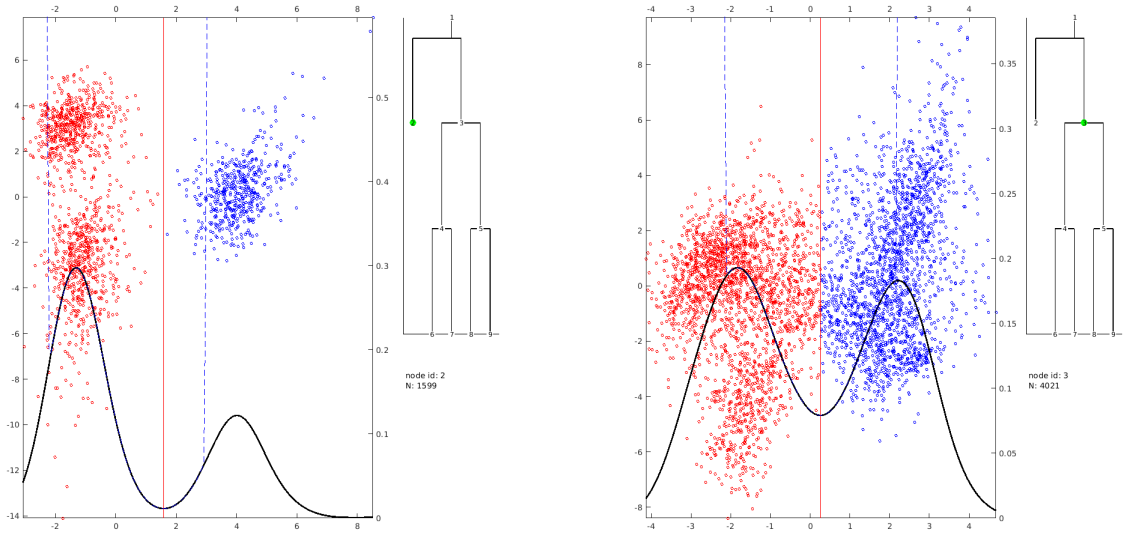
In this section we illustrate how to validate and modify a clustering model produced by any divisive hierarchical clustering algorithm in OPC. For partitioning algorithms, visualisation is straightforward using the projection matrix, and the in-build function `plot` and `scatter` in MATLAB and Octave. To modify the clustering model the user can set the optional input arguments, as discussed in the previous section.

The interactive validation and modification of divisive hierarchical models is more challenging, and constitutes the focus in the remainder of this section. In the present example we will use the `optidigits` dataset used in the previous section, but assume that the actual number of clusters is unknown, and start with an initial guess of $k = 5$. We use the `mddc` function to construct the initial cluster hierarchy, but the interface we describe relies on methods of the `ctree` class and is hence applicable to all divisive hierarchical clustering methods in OPC. The functions for visualisation, `plot` and `nplot` have been discussed extensively in the previous section. The two main functions for the modification of a `ctree` object are `prune` and `split`.

```
>> rng(201800630);
>> load('datasets/optidigits.mat');
>>
>> % Cluster optidigits datasets assuming k=5
>> [idx,t] = mddc(X,5);
>> Visualise cluster hierarchy (plot not shown)
>> plot(t);
>> % Evidence of 3 well separated clusters
>> nplot(t,2);
>> % Hyperplane intersects dense area
>> nplot(t,3);
```

An inspection of the cluster hierarchy (not shown for brevity), and the hyperplane separators in nodes 2 and 3 (shown in Figure 6.1), suggests that the leaf node 2, appears to contain 3 well separated dense clusters, while the hyperplane in node 3 intersects a dense region. The hyperplane separator for the cluster at node 2 appears to be effectively separating one of the three dense clusters. We can therefore use this hyperplane to split the cluster at node 2. This is achieved by calling the `split` function (method of the class `ctree`) with no optional arguments. If no optional arguments are specified, the `split` function partitions a leaf node using the binary separator estimated with the parameter settings used to create the first instance of the `ctree` object (in this example `t`). An inspection of the new leaf node with number 10 (omitted for brevity), illustrates that this cluster contains two of the three dense clusters identified in node 2, and that the MDH for this node separates these clusters effectively. We therefore split node 10 by calling the `split` function without specifying any arguments.

```
>> % Split cluster at leaf node 2 using estimated MDH
>> t1 = split(t,2);
>> % Plot omitted for brevity
>> nplot(t1,10);
>> % Split cluster at leaf node 10 using estimated MDH
>> t1 = split(t1,10);
>> % Visualise cluster hierarchy
```



(a) Node 2

(b) Node 3

Figure 6.1: Binary partitions at nodes 2 and 3 of initial clustering hierarchy

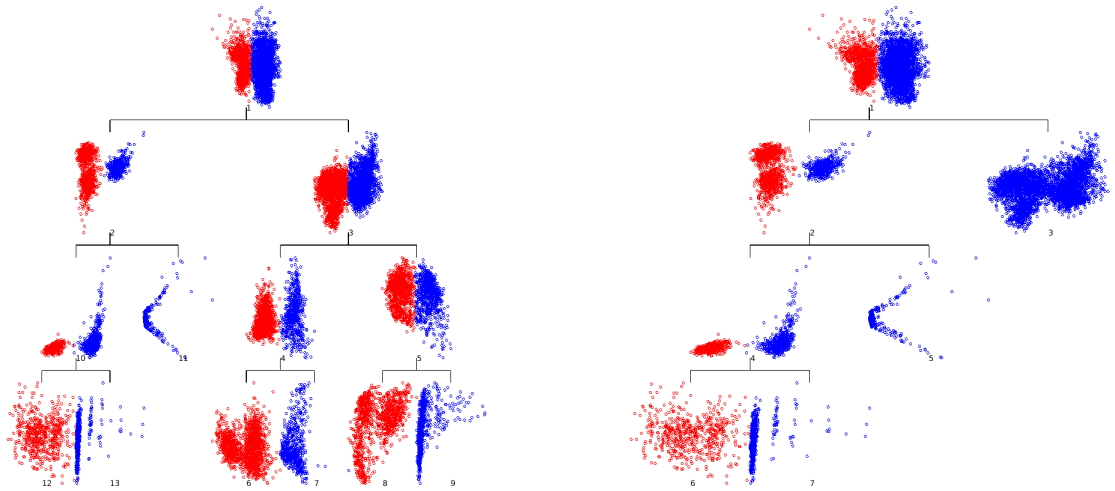
```
>> plot(t1);
```

We next explore whether the binary partition at node 3 can be improved. To consider alternative hyperplane separators, we first prune the subtree rooted at this node (the `split` function can only be applied to a leaf node). Note that after the tree is pruned the numbers of different nodes can change, as shown in Figure 6.2. Node numbers in `ctree` objects are always consecutive, and represent the order in which nodes were added to the model.

```
>> % Prune tree at node 3
>> t1 = prune(t1,3);
```

Having pruned the tree we next explore whether the cluster at node 3 can be partitioned more effectively. If we apply the `split` function on the cluster at node 3 we will obtain the same hyperplane as before. The `split` function accepts all the optional arguments (using the same syntax) of the divisive hierarchical clustering algorithm that created the first instance of the `ctree` object. In this example, we consider MDHs obtained by initialising the projection vector at the 2nd and 3rd principal component. To this end we need to set the `v0` argument. In the code below we also set the `verb` argument to one to enable the visualisation of the progress of the algorithm. Monitoring the progress of the projection pursuit algorithm we observe that the most promising MDH is obtained for the initialisation at the third principal component. Using the default parameter settings the algorithm rejects this solution because it is outside the default range around the mean of the data. To overcome this we increase the maximum range by setting `alphamax` to 1.2 (from a default of one). Note in the second sub-figure in Figure 6.3 that the local minimiser along the projected density is very close to the boundary of the range of admissible MDHs (evident by the blue dashed line that illustrates the projection index, Eq. (3.2)).

```
>> % Consider alternative parameter settings for PP algorithm:
>> % initialise at 2nd PC
>> split(t1,3,'v0',@(x,p)(pcacomp(x,2)), 'verb',1);
>> % initialise at 3rd PC, and increase range of admissible MDHs
>> split(t1,3,'v0',@(x,p)(pcacomp(x,3)), 'alphamax',1.2, 'verb',1);
```



(a) Tree after splitting node 2

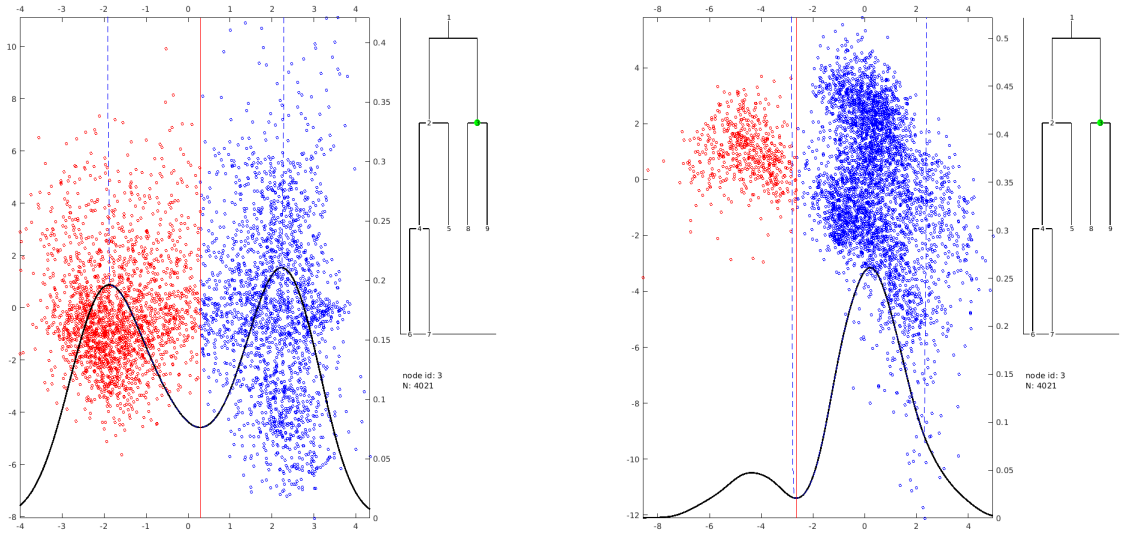
(b) Tree after pruning node 3

Figure 6.2: Cluster hierarchy after splitting node 2 and its children, and then after pruning node 3. Note how the numbers of nodes have changed after pruning the tree: node numbers represent the order in which nodes are added to the tree

```
>> % Update cluster hierarchy
>> t1 = split(t1,3,'v0',@(x,p)(pcacomp(x,3)), 'alphamax',1.2);
```

An inspection of the outcome of the revised clustering model indicates that both clusters arising from the split of node 3 are likely to contain more than one clusters. We first split the right child of node 3 and continue splitting clusters until no leaf exhibits evidence of containing more than one dense clusters. In the following, the default parameter values for the projection pursuit algorithm appear appropriate, so they are not modified. For brevity we omit the visualisations of the individual binary partitions in the following code snippet.

```
>> % Visualise and then split leaves that
>> % appear to contain more than one clusters
>> nplot(t1,9);
>> t1 = split(t1,9);
>> nplot(t1,11);
>> t1 = split(t1,11);
>> nplot(t1,12);
>> t1 = split(t1,12);
>> nplot(t1,15);
>> t1 = split(t1,15);
>> nplot(t1,10);
>> t1 = split(t1,10);
>>
>> % Assess performance
>> idx = tree2clusters(t1);
>> cluster_performance(idx, labels)

ans =

  struct with fields:
```



(a) Initialisation at 2nd PC

(b) Initialisation at 3rd PC

Figure 6.3: Candidate MDH for cluster at node 3 obtained by initialising at the second and third principal component respectively

```
      Purity: 0.8475
         NMI: 0.7842
     AdjRand: 0.7263
    Vmeasure: 0.7842

>> % Visualise clustering model
>> plot(t1,labels)
```

The performance of the final model is higher than that of the model obtained by `mddc` using the default settings and the correct number of clusters. The visualisation of the cluster hierarchy in Figure 6.4 verifies that this is a reasonable clustering model for this dataset.

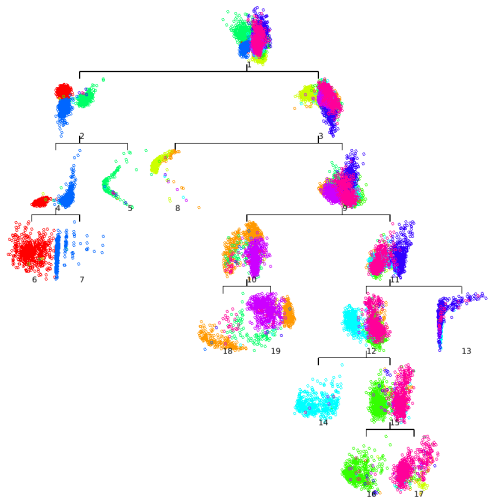


Figure 6.4: Final clustering model

7

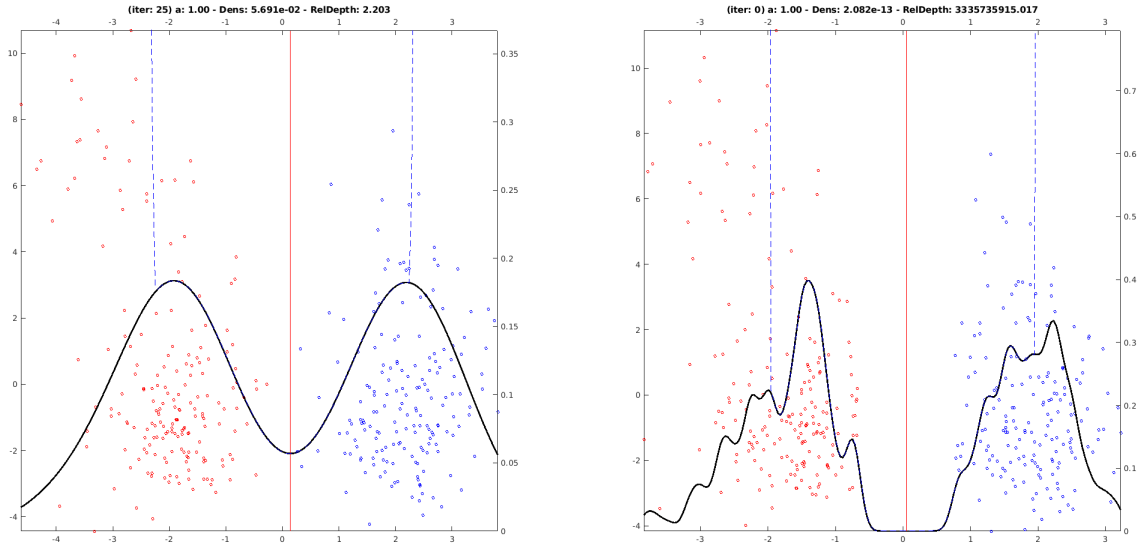
Extensions

7.1 Maximum Hard Margin Clustering

Maximum Margin Clustering (MMC) [30], extends the maximum margin principle, which has been very successful in supervised and semi-supervised classification, to clustering. The MMC problem can be expressed as identifying the binary labelling of $\mathcal{X}$ that will maximise the margin of a Support Vector Machine classifier estimated on the labelled data. MMC corresponds to a nonconvex integer optimisation problem, for which exact methods are only feasible for very small datasets. MMC algorithms that can handle reasonably sized datasets are not guaranteed to converge to the global optimum. The minimum density hyperplane (MDH), the minimum normalised cut hyperplane (NCUTH), and the spectral connectivity projection pursuit (SCPP) algorithm (when one-dimensional projections are used) converge to the maximum hard margin hyperplane as the bandwidth (scaling) parameter is reduced towards zero [18, 5, 7].

We illustrate this using the test set of the optical recognition of handwritten digits dataset, which is widely used as a benchmark in the MMC literature. One of the most difficult binary classification problems from this dataset is the separation of digits 3 and 9 [33, Table IV]. We standardise the dataset as in all previous examples. In the next example we recursively halve the bandwidth parameter employed by the `mdh` function until the vector that is orthogonal to the estimated MDH converges. At each iteration the projection vector is initialised to the vector that is orthogonal to the previously identified MDH.

```
>> load('datasets/optidigitsTest.mat');
>> % Observations corresponding to digits 3 and 9
>> index = find(labels==3 | labels==9);
>> % Standardise data matrix
>> X = normalise(X(index,:), 1);
>> labels = labels(index);
>>
>> % Estimate MDH with default bandwidth
>> [~,hp0] = mdh(X);
>>
>> hp = hp0;
>> % Value of alpha for which MDH is obtained
>> a = hp.params.alpha;
>> v0 = 0*hp.v;
>> while abs(hp.v' * v0) < 1-1.e-6,
>>      % obtain initial projection vector and bandwidth parameter
>>      v0 = hp.v;
>>      h = 0.5*hp.params.bandwidth;
>>      [idx,hp1]= mdh(X,'v0',v0,'alphamin',a,'alphamax',a,'bandwidth',h);
>>
>>      % Numerical problems can occur at very low bandwidth:
>>      % isinf(fval) signals projection pursuit failed
>>      if isinf(hp1.fval),
>>              break;
>>      else
```



(a) Initial MDH

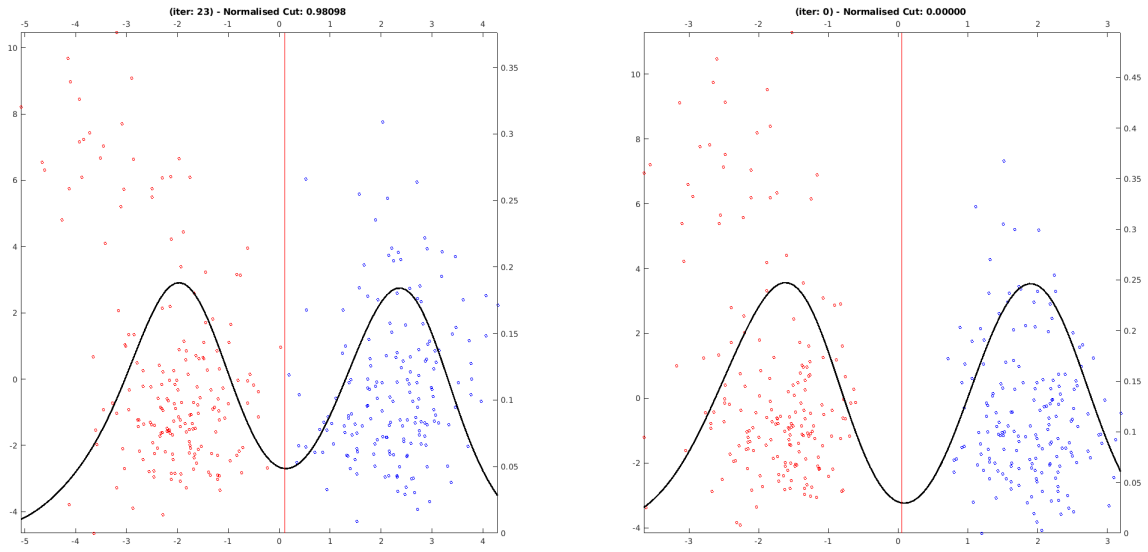
(b) Final MDH

Figure 7.1: Large hard margin separators obtained by recursively applying `mdh` for decreasing sequence of bandwidths

```
>>              hp = hp1;
>>      end
>> end
>> plot(hp0,X);
>> plot(hp,X);
>> % Misclassification error
>> er = 1 - purity(idx, labels(index))

er =

    0.0248
```

The minimum normalised cut hyperplane (NCUTH) also converges to the maximum hard margin hyperplane as σ, the scaling parameter of the Gaussian kernel used to compute pairwise similarities, is reduced towards zero. In the next example we illustrate this using the `ncuth` function.

```
>> % Estimate NCUTH with default bandwidth
>> [~,hp0] = ncuth(X);
>> hp = hp0;
>> v0 = 0*hp.v;
>> while abs(hp.v'*v0) < 1-1.e-10,
>>      v0 = hp.v;
>>      s = 0.5*hp.params.sigma;
>>      [id,hp1]= ncuth(X,'v0',v0,'sigma',s);
>>
>>      % Numerical problems can occur for very low scaling parameter:
>>      % isinf(fval) signals projection pursuit failed
>>      if isinf(hp1.fval),
>>              break;
>>      else
>>              hp = hp1;
>>      end
```



(a) Initial NCUTH

(b) Final NCUTH

Figure 7.2: Large hard margin separators using `ncuth` for a decreasing sequence of scaling parameters for the Gaussian kernel

```
>> end
>> plot(hp0,X);
>> plot(hp,X);
>> % Misclassification error
>> er = 1 - purity(id, labels)

er =

    0.0248
```

The large margin hyperplanes obtained by both NCUTH and MDH achieve a substantial improvement over all the methods reported in [33, Table IV]. These include the standard clustering algorithms: k-means, kernel k-means, spectral clustering, as well as the maximum margin clustering methods: generalised maximum margin clustering [26], and the iterative support vector regression [33]. NCUTH and MDH also outperform the more recent cutting plane maximum margin clustering algorithm [29]. (A MATLAB implementation of the last method is available from the lead author's personal page.)

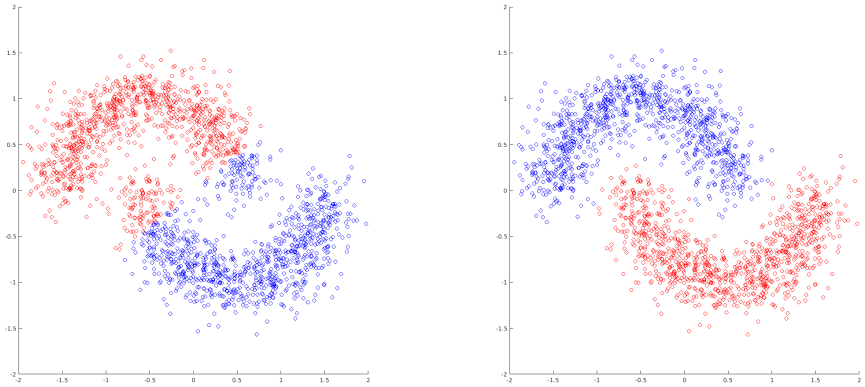
7.2 Kernel PCA: Non-linear clustering

In this section we illustrate a simple example of using Kernel Principal Component Analysis (KPCA) [20] to enable the identification of non–linearly separable, by hyperplane based methods. Through KPCA the data is 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. We illustrate this through a widely used two dimensional data set containing two clusters, each in the shape of a half moon, arranged so that they cannot be separated by a hyperplane [9].

The next example illustrates the use of the functions, `kpca` and `kpca_predict` included in OPC. These functions are based on the KPCA description in [20] and have the same syntax and default arguments as

the implementation of KPCA in the `R` package `kernelab` [11]. The only important difference between `kpca` and `kpca_predict` in OPC and `kernlab` is that in OPC the user needs to specify the kernel matrix as an argument.

```
>> load('datasets/halfmoons.mat');
>> % Hyperplane separator in original space
>> idx1 = ncuth(X);
>>
>> % Select randomly 200 observations to estimate KPCA
>> s = randperm(size(X,1),200);
>> % Compute Kernel matrix for data sample
>> sigma = 6;
>> K = exp(-sigma*squareform(pdist(X(s,:)).^2));
>> % Compute Kernel Principal Components
>> pcv = kpca(K);
>> % Project all feature vectors onto pcv
>> Kp = exp(-sigma*mypdist2(X, X(s,:)).^2);
>> X2 = kpca_predict(K,Kp,pcv);
>>
>> % Hyperplane separator in feature space
>> idx2 = ncuth(X2);
```



(a) Original space

(b) Feature space

Figure 7.3: Clustering obtained by hyperplane separators in the original, Euclidean space and in the kernel defined feature space.

8

Extending OPC

The OPC library provides a simple interface for users to create divisive hierarchical algorithms based on projection pursuit methods for clustering. The function `gppdc` implements a framework for *generic projection pursuit divisive clustering* algorithms. Like all divisive clustering algorithms in OPC, this function returns the cluster assignment `idx`, and a `ctree` object that represents the cluster hierarchy. The user can interact with this object in the same way as in all the previous examples. The function requires three mandatory inputs: the data matrix, `X`, the number of clusters, `k`, and a handle to a function that bi-partitions the data through projection pursuit clustering, `pphandle`.

The handle `pphandle` must be associated to a function that performs binary partitioning through projection pursuit. This is the only function that the user needs to specify. The function must have the following syntax:

```
[v,fval,idx] = ppfunc(X,param)
```

where `X` is the data matrix, and `param` is a structured array, which contains any additional arguments which are required to perform projection pursuit. For example the `param` structure can contain a handle to a function that determines the initial projection matrix. The output arguments of `ppfunc` are: the projection matrix, `v`, the value of the projection index, `fval`, and the binary cluster assignment, `idx` $\in \{1, 2\}^n$.

By default at each step of the divisive algorithm `gppdc` splits the cluster with the largest value of the projection index, `fval`. This behaviour can be modified by setting the `split_index` argument discussed below. Note that if the projection pursuit algorithm aims to minimise the projection index and the user wants at each step to split the cluster with the lowest projection index, an immediate way to achieve this without the need to define a `split_index` function is for `pphandle` to return minus one times the actual projection index.

The generic projection pursuit divisive clustering function, `gppdc`, accepts three optional input arguments: `param`, `split_index`, and `labels`.

1. `param` is used to specify the structured array that contains all the necessary parameters for the projection pursuit algorithm, as well as for the computation of the split index. This structured array is used as the second input argument to the `pphandle` function handle, and the third argument of the `split_index` function (described next).

2. `split_index` is a handle to a function that takes as input the projection matrix, `v`, the data matrix, `X`, and a structured array of parameters (`param`) and returns a scalar. The output of this function is the value of the splitting index for the cluster containing the observations in `X`. At each step of `gppdc` the cluster with the highest splitting index is bi-partitioned.

3. `labels` is used to specify the true cluster assignment. This information is only used after the clustering model is estimated to compute the success ratio and the purity at internal and terminal nodes of the tree, respectively.

The next code snippet specifies a function which can be used as a projection pursuit function to implement the bisecting k-means algorithm [23]. The function does not require the specification of any parameters. The data matrix is the only input argument. It applies 2-means to identify the two cluster centroids (`C`), and the assignment of observations to clusters (`idx`). The projection matrix in this case is the vector connecting the two centroids (normalised to have unit-length). In [23] it is recommended to split the cluster with the largest size. The projection index, `fval`, is therefore equal to the number of observations.

```
function [v,fval,idx] = bisKmeansPP(X)
        [idx,C,sumd] = kmeans(X,2,'EmptyAction','singleton','Replicates',1);
        v = (C(1,:) - C(2,:))';
        v = v./norm(v,2);;
        fval = size(X,1);
end
```

The above function is in the file `bisKmeansPP.m` in the directory `src/generic`. To perform bisecting k-means through OPC using as cluster splitting criterion the total scatter we can use the `gppdc` function as:

```
>> load('datasets/optidigits.mat');
>> [idx,t] = gppdc(X,10, @(x,p)(bisKmeansPP(x)), 'split_index', @(v,x,p)(total_scatter(x)));
```

The user can visualise and modify the cluster hierarchy obtained through `gppdc` through the functions, `plot`, `nplot`, `prune`, and `split` discussed extensively in Section 6. No additional code needs to be written for this functionality to be available.

A *projection pursuit* version of bisecting k-means can be obtained by using LDA-k-means instead of standard k-means to recursively bi-partition the data. To achieve this the user need only define a function that reorganises the order of the output arguments of `ldakmeans`, to match the `pphandle` syntax requirements:

```
function [v,fval,idx] = lda2m(X)
        [idx,v,fval,C,iter] = ldakmeans(X,2);
end
```

The above function is in the file `lda2m.m` in the directory `src/generic` script file. A "bisecting" version of LDA-k-means is readily obtained as:

```
[idx, t] = gppdc(X,10, @(x,p)(lda2m(x)));
```

By not specifying a `split_index` function in the above algorithm the cluster that has the maximum projection index (which in the LDA-k-means case is the ratio of between–cluster to within–cluster scatter) is split at each step.

Bibliography

[1] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035. SIAM, 2007.

[2] D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[3] 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, 2007.

[4] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[5] D. P. Hofmeyr. Clustering by minimum cut hyperplanes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1547–1560, 2017.

[6] D. P. Hofmeyr and N. G. Pavlidis. Maximum clusterability divisive clustering. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 780–786, 2015.

[7] D. P. Hofmeyr, N. G. Pavlidis, and I. A. Eckley. Minimum spectral connectivity projection pursuit. *Statistics and Computing*, 2018.

[8] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec 1985.

[9] Anil K. Jain and Martin H. C. Law. Data clustering: A user's dilemma. In Sankar K. Pal, Sanghamitra Bandyopadhyay, and Sambhunath 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 Berlin Heidelberg.

[10] I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.

[11] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab - an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[12] 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.

[13] M. Lichman. UCI machine learning repository, 2013.

[14] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis. Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems*, pages 1113–1120, 2008.

[15] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. 2001.

[16] D. Niu, J. G. Dy, and M. I. Jordan. Dimensionality reduction for spectral clustering. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *JMLR W&CP*, pages 552–560, 2011.

[17] 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.

[18] N. G. Pavlidis, D. P. Hofmeyr, and S. K. Tasoulis. Minimum density hyperplanes. *Journal of Machine Learning Research*, 17(156):1–33, 2016.

[19] 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.

[20] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[21] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, Aug 2000.

[22] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[23] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Workshop on text mining, ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD)*, pages 525–526, 2000.

[24] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, Dec 2002.

[25] S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos. Enhancing principal direction divisive clustering. *Pattern Recognition*, 43(10):3391–3411, 2010.

[26] Hamed Valizadegan and Rong 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.

[27] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review. Technical Report TiCC-TR 2009-005, Tilburg University, 2009.

[28] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[29] F. Wang, B. Zhao, and C. Zhang. Linear time maximum margin clustering. *IEEE Transactions on Neural Networks*, 21(2):319–332, 2010.

[30] Linli Xu, James Neufeld, Bryce Larson, and Dale 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.

[31] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, pages 1601–1608, 2004.

[32] B. Zhang. Dependence of clustering algorithm performance on clustered-ness of data. *HP Labs Technical Report HPL-2001-91*, 2001.

[33] K. Zhang, I. W. Tsang, and J. T. Kwok. Maximum margin clustering made practical. *IEEE Transactions on Neural Networks*, 20(4):583–596, 2009.

[34] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM, 1996.

[35] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.