2003

TR-2003009: A Hierarchical Projection Pursuit Clustering Algorithm

Jayson E. Rome

Alexei D. Miasnikov

Robert M. Haralick

A Hierarchical Projection Pursuit Clustering Algorithm

Jayson E. Rome, Alexei D. Miasnikov, Robert M. Haralick
Pattern Recognition Lab
Department of Computer Science
The Graduate Center of The City University of New York
365 Fifth Avenue
New York, New York 10016

8/18/2003
version 3.8

Abstract

We define a cluster to be characterized by regions of high density separated by regions that are sparse. Given a collection of observations $X = \{x_i\}$, $x_i \in \mathbb{R}^d$, $|X| = N$, we would like to find clusters in data sets in which d and possibly N are large, in which there is no known parametric distribution and in which clusters may take on arbitrary shapes. By observing the downward closure property of density, the search for interesting structure in a high dimensional space can be reduced to a search for structure in lower dimensional subspaces. We present a parameter free Hierarchical Projection Pursuit Clustering (HPPC) algorithm that repeatedly bi-partitions interesting lower dimensional projections of a high dimensional dataset. We describe a projection search procedure for use with relatively high dimensional data and a projection pursuit index function based on the Kittler and Illingworth optimal threshold technique. The output of the algorithm is a decision tree whose nodes store a projection and threshold and whose leaves represent the clusters (classes). We present several methods for cluster validation that are used to evaluate the algorithm. Experiments with various real and synthetic datasets show the effectiveness of the approach.

Portions of this work were funded by a New York State Office of Science, Technology and Academic Research grant and by Syllogy Software in cooperation with the Institute for Software Design and Development of the City University of New York

Contents

1 Introduction

A cluster is a set of data points partitioned in such a way that individual data elements within the same cluster are in some sense more similar to elements in the cluster than to elements outside of the cluster. Clustering is an old and well studied problem [25], though most standard techniques assume that the data is drawn from a known parametric distribution or that the data exists in a low dimensional space.

Given a collection of observations $X = \{x_i\}$, $x_i \in \mathbb{R}^d$, $|X| = N$, we would like to find clusters in data sets in which d and possibly N are large, in which there is no known parametric distribution and in which clusters may take on arbitrary shapes. For these kinds of data sets traditional techniques fail and new methods must be developed.

We assume that clusters are characterized by regions of high density separated by regions that are sparse and observe that any low density region in a subspace of the data corresponds to a low density region in the full space. Thus the search for interesting structure in the high dimensional space can be reduced to a search for structure in lower dimensional subspaces. Based on this downward closure property of density, we propose a Hierarchical Projection Pursuit Clustering (HPPC) algorithm. The algorithm has no required input parameters, is able to handle relatively large dimensional data, is scalable to large datasets and produces a compact description of the clusters found in the form of a binary tree. This tree, once clusters have been found, can be used to classify new observations as belonging to particular clusters.

1.1 Previous Work

There has been a great deal of recent work towards developing efficient and effective algorithms for clustering in high dimensions, including CURE [23], BIRCH [47], DBSCAN [19], DENCLUE [27], [28], STING [45], CLARANS [36], CLIQUE [2], ENCLUS [10], ORCLUS [8], WaveCluster [41], ROCK [24], [4], [46], [32]. We refer the interested reader to [32] and [5] for extensive surveys of recent clustering research.

1.2 Tour of the paper

In section 2 we look at the Projection Pursuit paradigm and discuss several clustering algorithms that utilize some form of projection and an associated notion of closure. One of the algorithms, the Apriori Algorithm [3], was designed for variable clustering. While the other algorithms, CLIQUE, ENCLUS and ORCLUS, are designed for data clustering, they are all descendants of Apriori. Finally, we describe a recent method of projection pursuit clustering, that is illustrative of a class of approaches, that utilizes an index function based on orthogonal canonical variates. In section 3 we present the indexing function, search procedure and stopping criteria that comprise the HPPC algorithm. In section 4 we address the issue of cluster validation and describe a method for computing the accuracy of a cluster algorithm. In section 5 we describe various experiments that where performed on a variety of real and synthetic data, while section 6 discusses conclusions and possible future work.

2 Projection Pursuit

We are given a data set in a large dimensional space and would like to find a subspace of lesser dimension that preserves the structure of the original data. Consider a data set X, which is a collection of $1 \times d$ observation vectors. Let α be a $d \times d_0$, $d_0 << d$ matrix such that $\alpha'\alpha = I_{d_0}$

. Then α is an orthonormal projection matrix form d to d_0 dimensions. Let $z_i = \alpha' x_i$ be the projection of the i^{th} observation. *Projection Pursuit* [22], [20], [21], [31] or PP, is defined to be the search for interesting (structured) projections. We are interested in non-linear structures like clusters and separations that cannot be captured by a mean vector μ and a covariance matrix Σ. Mathematically, interestingness is a sample estimate of a distance between the distribution of the projected data and a distribution that is known to be uninteresting. Uninteresting distributions are normally taken to be uniform or normal, though some other parametric form may be used, or they may be determined empirically for a specific case. A PP algorithm consists of two components:

- An index $I(\alpha)$ that measures the "usefulness" or "interestingness" of projection α,
- An algorithm that varies the projection direction so as to find the optimal projections, given the index function $I(\alpha)$ and the data set X.

As the dimensionality of the data increases, the search for projections becomes exponentially more difficult, and projection selection becomes crucial. A technique for projection selection which has recently been investigated both theoretically [1] and experimentally [6], [13] is random projection. We can choose a projection operator α at random according to almost any zero mean unit variance distribution on the elements α_{ij} of α [1]. Some common choices for the distribution of the α_{ij} are the standard unit normal distribution $\mathcal{N}(0,1)$[6], [13], uniform distribution over $[-1,1]$ [13], binomial distribution with parameters

$$\begin{array}{c} p(\alpha_{ij} = +1) = 1/2 \\ p(\alpha_{ij} = -1) = 1/2 \end{array},$$

[1] or multinomial distribution with parameters

$$\begin{array}{c} p(\alpha_{ij} = +1) = 1/6 \\ p(\alpha_{ij} = 0) = 2/3 \\ p(\alpha_{ij} = -1) = 1/6 \end{array}$$

[6], [1]. Random projection has been used successfully for dimensionality reduction, [6] reported results similar to those obtained when using Principle Components for text and image data.

2.1 Apriori Algorithm

2.1.1 Variable Clustering

Though many techniques are available for variable clustering, for example Gaussian and Discrete Graphical Models [18], [34], [37], we will concentrate on a projection based technique known as the Apriori Algorithm [3] that arose from work on Association Rule Mining [29]. An association rule is an expression of the probability of occurrence of one set of random items, given that another set of random items has occurred. The Apriori algorithm utilizes the downward closure property of measures of interestingness to prune the search space of rules that are not interesting.

It is important to point out that the Apriori Algorithm [3] is in fact a projection based technique. All projections are taken perpendicular to the original variable axis in the form of counts of co-occurrence of subsets of variables in the database of transactions. The support and confidence thresholds are used to filter rules (variable subsets) from consideration. In addition, Apriori is worthy of study as it forms the foundation for many algorithms that utilize projections for data clustering.

2.1.2 Downward Closure

Given a set X of observations on a set I of binary random variables $i_1, ..., i_d$ we would like to find relationships of the form

$$A \Rightarrow B$$

between subsets $A, B \subseteq I$. This relationship reflects a dependance on the occurrence of subset A and the likelihood of the co-occurrence of subset B. We can define measures of confidence and support for a given association rule and a given database of observations. We will use the following notation to describe the Apriori Algorithm: $I = \{i_1, ..., .i_d\}$ is the set of all *Items*. $x \subseteq I$ is a set of items, constituting a single *Transaction*. $A \Rightarrow B$, $A \subset I, B \subset I, A \bigcap B = \emptyset$ is an *Association Rule*. The rule states that if item A is included in a transaction x, B will also be included with some estimated confidence (or probability). *Confidence*, the percent of transactions in X that contain A, that also contain B, is denoted c.

$$conf(A \Rightarrow B) = \frac{N_{A \cup B}}{N_A} = \frac{\frac{N_{A \cup B}}{N}}{\frac{N_A}{N}}$$

$$= \frac{p(A, B)}{p(A)} = p(B|A).$$

$minconf$ is a user defined threshold on confidence. *Support*, the percentage of transactions in X that contain $A \bigcup B$, is denoted s.

$$supp(A \Rightarrow B) = \frac{N_{A \cup B}}{N}$$

$minsup$ is a user defined threshold on support.

Using these measures we would like to find the set of rules that has the highest confidence and support. A $d-1$ item set with small support cannot be a part of d item set with large support, and a d item set with large support will have $d-1$ item set subsets with large support. An iterative algorithm is used to generate candidate item sets from subsets with large support. The output is a set of association rules and a measure of support for each rule.

2.1.3 Finding Association Rules

The general problem can be stated:

- Given a collection of observations X,
- Find all significant association rules.

subject to large d, and large N. For an ideal solution variables assigned to the same rule belong to the same generative process, and the number of rules found is minimal while still reflecting the true number of generative processes. The basic variable clustering algorithm can be expressed as:

1. Given : data set X, user specified parameters θ
2. Find : All large item sets L that satisfy user specified criterion $g(\theta)$,
3. Given : Large item sets $\bigcup_d L_d$,
4. Find : The desired rules $A_i \Rightarrow B_i$.
5. Generate : Minimal description of the rule set.

2.1.4 The Algorithm

The algorithm proceeds as follows:

1. Given data set X, $minconf$, $minsup$,
2. Find all item sets with $s \geq minsup$,
   (a) Given : $L_1 =$ the set of all large $1-$item sets.
   (b) for$(d = 2;\ L_{d-1} \neq 0;\ d++)$
   (c) $\quad C_d =$ Apriori-Gen(L_{d-1})
   (d) $\quad$ for all transactions $t \in X$
   (e) $\quad\quad C_t = subset(C_d, t)$
   (f) $\quad\quad$ for all candidates $c \in C_t$
   (g) $\quad\quad\quad c.count++$
   (h) $\quad\quad L_d = \{c \in C_d | c.count \geq minsup\}$
   (i) return $\cup_d L_d$

The procedure Apriori-Gen proceeds as follows: Given the set of all large $d-1$ item sets, Find the superset of all large d item sets.

1. insert into C_d
2. join(L_{d-1}, L_{d-1})
3. prune $c \in C_d$ if some $(d-1)$ subsect of c is $\notin L_{d-1}$

Having found large item sets, we must now determine which rules are "good" using the Gen-Rules procedure:

1. For every large item set l that is found, all nonempty subsets a of l are found.
2. A rule of the form $a \Rightarrow (l-a)$ is considered "good' if

$$\frac{supp(l)}{supp(a)} \geq minconf.$$

2.1.5 Example

Consider the following simple example of the Apriori Algorithm from Agrawal and Srikant [3], given the transaction database

$X =$

TID	$itemset$
100	{A,C,D }
200	{B,C,E }
300	{A,B,C,E }
400	{B,E }

over the set of items $I = \{A, B, C, D, E\}$. The support for the 1-item sets is

$$s(I) = \{2, 3, 3, 1, 3\}.$$

With a threshold minsup of 1 we have

$$L_1 = \{A, B, C, E\}$$

The Apriori-Gen routine then performs the self-join of L_1 to obtain

$$C_2 = \{AB, AC, AE, BC, BE, CE\}.$$

The pruning step results in C_2 being unchanged. We obtain the following support

$$s(C_2) = \{1, 2, 1, 2, 3, 2\}$$

Thresholding gives

$$L_2 = \{AC, BC, BE, CE\}$$

The second call to apriori-gen results in

$$C_3 = \{BCE\}$$

The pruning step again leaves C_3 unchanged, and we obtain the following support

$$s(C_3) = \{2\}$$

A call to apriori-gen will not yield any new item sets and so the algorithm terminates. The algorithm outputs the following

$$L = \{A, B, C, E, AC, BC, BE, CE, BCE\}$$

with support

$$\{2, 3, 3, 3, 2, 2, 3, 2, 2\}$$

Gen-Rules generates the following rules

$R =$

$rule$	$confidence$
$B \Rightarrow CE$	2/3
$C \Rightarrow BE$	2/3
$E \Rightarrow BC$	2/3
$BC \Rightarrow E$	1
$CE \Rightarrow B$	1
$BE \Rightarrow C$	2/3
$A \Rightarrow C$	1
$C \Rightarrow A$	2/3
$B \Rightarrow C$	2/3
$C \Rightarrow B$	2/3
$B \Rightarrow E$	1
$E \Rightarrow B$	1
$C \Rightarrow E$	2/3
$E \Rightarrow C$	2/3

2.2 CLIQUE

CLIQUE [2] is a data clustering algorithm that was designed to be effective for large dimensions, create results that are easily interpretable, and to be scalable. The algorithm is based on the notion of *monotonicity* or downward closure, which states that if a set of points S is a cluster in a d dimensional space, then S is also part of a cluster in any $(d-1)$ dimensional subspace.

A *unit* is formed from the intersection of d axis parallel intervals, $u_i = [l_i, h_i)$, and is a single cell of the partitioning, all units having equal volume. A *cluster* is defined as a region that has higher density of points than its surrounding region, equivalent to the union of connected high density units of a subspace. A *minimal description* is defined as a non redundant covering of the cluster C with a set of maximal regions $\mathcal{R} = \{R_i\}$ such that $\bigcup_{\forall i} R_i = C$ but the union of any proper subset of $\mathcal{R}$ does not give C. (We should also add i is minimal and that accuracy is maximal).

2.2.1 Algorithm

The CLIQUE algorithm proceeds as follows

1. *Identification of subspaces that contain clusters.* The following algorithm follows from monotonicity:
   (a) Start with dimension i and find the clusters C_k.
   (b) Backproject over the next dimension $i+1$.
   (c) Remove those cells in C_k that are not in C_{k-1}
   (d) MDL - Based pruning : given the subspaces $S_1, ..., S_n$ the coverage of subspace S_j is given by: $x_{S_j} = \sum_{u_i \in S_i} count(u_i)$, remove subspaces with smallest coverage.
2. *Identification of clusters.* The dense units correspond to nodes in graph that are connected by an edge if the corresponding units share a common face. The connected components of this graph correspond to the clusters.
3. *Generation of minimal description for the clusters.* Try to cover all of the units comprising a cluster with the minimum number of regions such that all regions contain only connected units.

2.2.2 Apriori Perspective

In the case of association rule mining we where interested in an expression of the probability of co-occurrence of various sets of events, given the occurrence of other sets of events. For record clustering we are interested in dense cells that are part of the same cluster, or that equivalently belong to the same connected component. An algorithm similar to the one used to prune the search space for association rules in the Apriori-gen procedure is used to prune from consideration those cells that cannot be dense. Monotonicity states that all projections to subspaces of dimension $d_0 < d$ of a dense region in d dimensions are also dense. We need to find all of the units (items) that are dense in one dimension L_1, (these are the 1-item sets). These units are used to generate candidates in the 2 dimensional subspaces (the 2-item sets) C_2, by performing a self join of L_1. Those candidate units in C_2 that do not have a corresponding dense projection in L_1 are removed (pruned) from C_2. This iteration is repeated until no new dense units are found. Connected components of the resulting set L_d are the clusters.

In terms of Apriori algorithm we have: Extend the binary item to a discrete collection of item values, so that each feature (axis) represents an item and each item may take one of a finite set of values. Observations are points in a quantized high dimensional space. A unit (item) is a single cell in the quantized space. L_d are the dense units, candidate units are found by self join of L_d, dense units that have a projection in $(d-1)$ dimensions that is not in C_{d-1} are pruned from C_d. Candidates are pruned based on coverage, the fraction of the data set that is covered by the dense units. The output is a set of connected dense components and a minimal description in the form of a Disjunctive Normal Form (DNF) expression.

2.3 ENCLUS

ENCLUS [10] is a clustering algorithm that was designed to have the ability to discover clusters in subspaces, to work well in high dimension, to make no assumptions about the distribution of the data, to require no parameters, and to be robust to noise and outliers.

Downward closure states that if a d-dimensional subspace has good clustering, then all $(d-1)$-dimensional projections of this subspace will also have good clustering. Define an *Interest measure* (equivalent to mutual information) so that the higher the interest the stronger the correlation.

$$interest(\{X_1, ..., X_n\}) = \sum_{i=1}^{n} H(X_i) - H(X_1, ..., X_n)$$

Define *Interest gain* to be the increase in interest from adding an extra dimension.

$$interestgain(\{X_1, ..., X_n\}) =$$

$$interest(\{X_1, ..., X_n\}) - \max_i \{interest(\{X_1, ..., X_n\})\} - \{X_i\}$$

2.3.1 Algorithm

The ENCLUS algorithm proceeds as follows

1. Find subspaces whose entropy is below some threshold. These subspaces have good clustering.
2. Find 1D subspaces with good clustering,
3. Generate candidate $d-1$ subspaces,
4. Check candidate subspaces against raw data to find only those candidates with good clustering,
5. repeat until no new subspaces are found.

2.3.2 Apriori Perspective

ENtropy based subspace CLUStering (ENCLUS) is a refinement of CLIQUE that uses a modified pruning step. The pruning step in CLIQUE can result in a large portion of candidate cells that are not good. ENCLUS adds the criterion of high density and correlation, using an entropy based measure to identify subspaces with good clustering. Downward closure property of Shannon's entropy is the basis for the monotonicity property required by the apriori-gen algorithm. Candidates are pruned based on entropy and coverage. The output of the algorithm is a set of connected dense components and a minimal description in the form of a disjunctive normal form (DNF) expression.

2.4 ORCLUS

Define a *Generalized Projected Cluster* [8] to be a subset of vectors $\mathcal{E}$ together with a subset of data points $\mathcal{C}$ such that points in $\mathcal{C}$ are closely clustered in the subspace defined by vectors $\mathcal{E}$. subsubsectionAlgorithm The ORCLUS algorithm proceeds as follows:

1. *Assign* : Partition the data into k_c current clusters by assigning each point to the nearest seed, using the projected distance in the current subspace $\mathcal{E}_i$.
   (a) Replace each seed by the centroid of the cluster
2. *Find Vectors* : Find the subspace $\mathcal{E}_i$ of dimensionality l_c for each current cluster $\mathcal{C}_i$.
   (a) Compute the covariance matrix for $\mathcal{C}_i$,
   (b) Choose the l_c orthonormal vectors with the smallest eigenvalues.
   (c) Reduce l_c by fraction $\beta < 1$.
3. *Merge* : Reduce the number of clusters from k_c to $k_{new} = (1 - \alpha)k_c$ for some $\alpha < 1$.
   (a) Perform SVD on $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_j$,
   (b) Find the l_{new} smallest eigenvalues to define space $\mathcal{E}'_{ij}$
   (c) Find the centroid s'_{ij} of $\mathcal{C}_i \cup \mathcal{C}_j$
   (d) Use energy $r_{ij} = R(\mathcal{C}, \mathcal{E}'_{ij}$
   (e) Terminate when $k_c = k$.

2.4.1 Apriori Perspective

ORCLUS is a method of finding clusters in arbitrarily oriented subspaces, unique to each cluster. Differs from Apriori in the use of arbitrarily oriented subspaces, optimally chosen for each cluster individually. ECF (Extended Cluster Feature Vector based on BIRCH's Cluster Feature Vector) used to ease computational burden. The algorithm uses a k means type iteration to update the location and subspace orientation of randomly chosen seed points. Distances are computed in the projected subspace, specified for each cluster. At each iteration, the following steps are performed

1. Assign : each observation to its closest seed in the current projected subspace.
2. Find Vectors : that define the subspace the defines the current cluster.
3. Merge : those clusters whose combined projected energy is small.

Allowing arbitrary projections complicates the subset search and we lack the convenient join and prune algorithm. Seed points are chosen randomly and are used to define an optimal subspace and a new seed point. If two clusters have a small combined projected energy, then they are merged and a new subspace and seed are computed. Database points are reassigned to the closest seed and the process repeats until the desired number of clusters is found. Output is a set of cluster descriptors consisting of a centroid and a subspace (ECF descriptor) for each cluster.

2.5 Bolton and Krzanowski

Bolton and Krzanowski [7] describe a projection pursuit clustering index, based on orthogonal canonical variates, that takes into account the scale in the data. They show that, in the case that the data is sphered, their index is identical to that of Bock.

2.5.1 Bock's Algorithm

Bock's method seeks a partition of the data into k subgroups, or clusters, for a user specified k, such that the within group variance is minimized. The data are partitioned into k clusters at random, and elements are moved among groups so as to minimize the within group sum of squares in the lower dimensional partition.

$I(\alpha, C)$ is a function of a d dimensional projection α and the clustering C. Projection matrix α is orthonormal, so that $\alpha\alpha' = I_d$. The clustering of objects into k classes at step s is denoted $C_{k,s}$.

1. INIT: $s = 0$, choose $C_{k,0}$, an initial clustering of k groups
2. ITERATE until $I(\alpha, C)$ does not improve significantly:
   (a) α_s = Optimize $I(\alpha, C = C_{k,s})$ over all projections α.
   (b) $C_{k,s+1}$ = Optimize $I(\alpha = \alpha_s, C)$ over all clusterings C with k groups.
   (c) $s = s + 1$.

Calculations in the second step of the optimization are done on the projected data, resulting in a computational speed up. Bock's method involves finding the projection that maximizes the between group sum of squares, and using k-means to find a partition that minimizes the within-groups sum of squares. Given that the total sum of squares of products is given by the sum of the between and within sum of squares of product matrices:

$$T = B_C + W_C,$$

Bock's procedure reduces to maximizing the index

$$I_{Bock}(\alpha, C) = trace(\alpha B_C \alpha'),$$

with α orthonormal.

2.5.2 The dDOCV Index

Bolton and Krzanowski observe that Bock's index tends to find groups in the direction of the data that has largest variance, even when the structure is better defined in other directions. They propose a new index function based on orthogonal canonical variates:

$$I_{dDOCV}(\alpha, C) = \frac{\sum_{i=1}^{d} \alpha_i T \alpha_i'}{\sum_{j=1}^{d} \alpha_j W_c \alpha_j'},$$

such that $\alpha_i\alpha_i' = 1$, and $\alpha_i\alpha_j' = 0$, when $i \neq j$. The optimal projection is found by an iterative procedure that computes the d^{th} projection α_d by fixing $\alpha_1, ..., \alpha_{d-1}$ and maximizing I_{dDCOV} over all α_d subject to the orthogonality constraint. The d dimensional clustering is found using k means. The algorithm suffers from the need to specify the number of clusters k and an initial

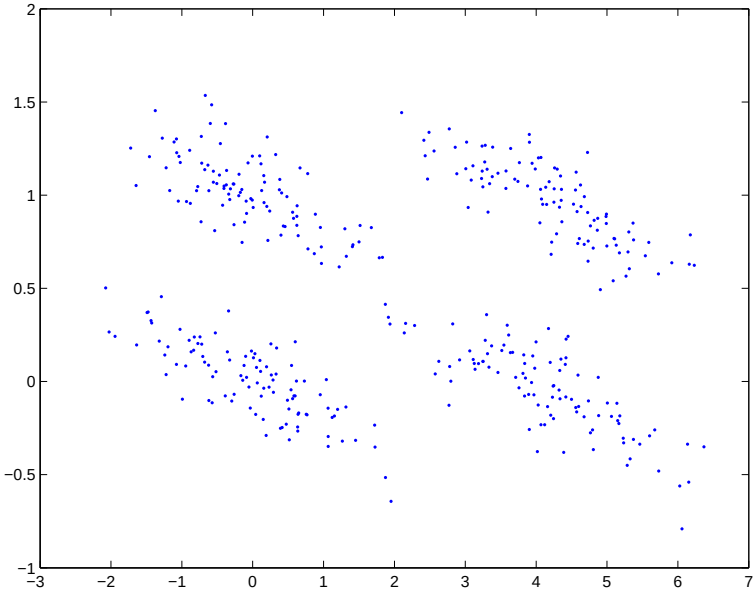

Figure 1: A simple dataset with four clusters composed of a mixture of four multivariate normals with different means and the same covariance structure.

seed. The following quote from the paper summarizes the initialization requirements. "Both data sets contain *a-priori* groups, so we set the algorithm to look for the number of groups described in the original classification and set the starting seed as the original classification. This may appear to be leading the witness somewhat, but this method is in lieu of an ideal situation in which we would employ a visual grand tour to assist us in choosing our initial clustering [7]."

3 The HPPC Algorithm

Consider the simple dataset shown in figure 1 and one of its projections shown in figure 2. Diaconis and Freedman [15] showed that, as a consequence to the central limit theorem, most projections of high dimensional data to low dimension will be approximately normally distributed. We therefore consider interesting projections to be those for which there exists a natural partition of the data into two normally distributed components. We make the observation that low density regions in the projected space correspond to open corridors in the full space. Therefore, given the projected data, we would like to find a low density region which partitions the data. This corresponds to searching for valleys in the reduced space.

We have several options available for locating the valleys in the subspace including methods based on derivatives, global thresholds or localized thresholds [35], [40]. Instead, HPPCluster casts the problem as one of a two class classification problem and seeks the 1D projection that has minimal classification error.

HPPCluster is a hierarchical projective clustering algorithm that repeatedly bi-partitions interesting one dimensional projections of a dataset. The data are first projected to a one dimensional subspace and a histogram of the projected data is formed. This histogram is

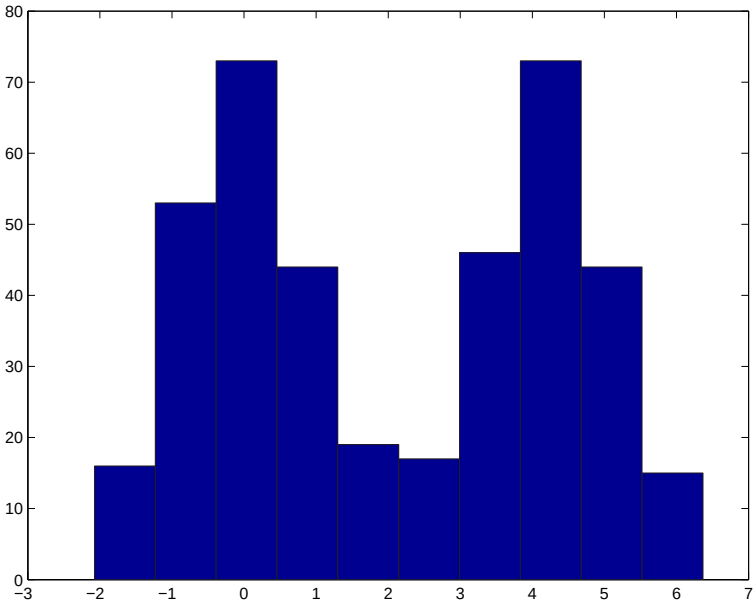

Figure 2: The histogram of the data projected to the $x2$ coordinate axis. Low density regions (valleys) in the projected space correspond to open corridors in the full space.

thresholded using the Kittler and Illingworth [33] minimum error thresholding technique. An index function based on the Kittler and Illingworth criterion function as well as the separation between means of the two components relative to their respective variances is used to measure the interestingness of the projection. At each iteration the data are split according to the best projection found. Each time a split is chosen a node is created in a decision tree, the data is partitioned and the algorithm is repeated on the data in each of the leaves of the tree, until a stopping condition is satisfied. The output of the algorithm is a decision tree whose nodes store the projection and optimal threshold and whose leaves represent the clusters. The tree that HPPCluster constructs can be used as a classifier, in which at least one terminal node is associated with each pattern class, and the interior nodes represent various collections of mixed classes. In particular, the root node represents the entire collection of classes into which a unit may be classified. When an unknown unit enters the decision tree at the root node, a decision rule associated with the root node is applied to the unit's measurement vector to determine the descendant path that the unit will follow. This process is repeated until a terminal node is reached. Every terminal node has an associated class to which the unit is finally assigned.

The pseudo-code in figures 3 and 4 describe the algorithm.

3.1 The Index Function

Kittler and Illingworth [33] describe a method originally designed for segmenting object from background in gray scale images. The procedure views the segmentation as a two class classification problem that can be distinguished based on the grey level histogram of the image. The goal is to find a threshold that minimizes the number of miss-classified pixels. Assume that we have a histogram (1D projection) $h(g)$ which gives the frequency of occurrence of the various

procedure *[τ, θ, α] = SplitData(dataset X)*
Initialize best threshold $\tau = 0$, best index $\theta = -\infty$, best projection $\alpha = \mathbf{0}$;
Construct S : the set of candidate 1D projections;
FOR $\alpha_i \in S$ **DO**

$hist_i = histogram(\alpha_i X)$;

current threshold $\tau_i = FindMinimumErrorThreshold(hist_i)$;

if(StoppingConditionSatisfied($hist_i$)) **CONTINUE**;

current evaluation $\theta_i = EvaluateThreshold(\tau_i, hist_i)$;

if($\theta_i > \theta$) $\theta = \theta_i, \tau = \tau_i, \alpha = \alpha_i$;

END FOR LOOP

Figure 3: The splitting procedure

levels of $g = \alpha x_i, i = 1, ..., N$. The histogram is viewed as an estimate of the probability density function of a mixture of two clusters. We assume that each component $p(g|i)$ of the mixture is drawn from a normal distribution with mean μ_i, standard deviation σ_i and *a priori* probability P_i, so that

$$p(g) = \sum_{i=1}^{2} P_i p(g|i),$$

where

$$p(g|i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-(g-\mu_i)^2/2\sigma_i^2}.$$

Given the (μ_i, σ_i, P_i), there exists a threshold τ such that

$$P_1 p(g|1) > P_2 p(g|2) \;\; if \;\; g \leq \tau$$

and

$$P_1 p(g|1) < P_2 p(g|2) \;\; if \;\; g > \tau.$$

Threshold τ is the Bayes minimum error threshold [16, 17]. As shown in figure 5, the minimum error threshold can be found by solving the quadratic equation obtained by:

$$P_1 \frac{1}{\sqrt{2\pi}\sigma_1} e^{-(g-\mu_1)^2/2\sigma_1^2} = P_2 \frac{1}{\sqrt{2\pi}\sigma_2} e^{-(g-\mu_2)^2/2\sigma_2^2}.$$

Since the true (μ_i, σ_i, P_i) are rarely known, they need to be estimated using computationally expensive fitting techniques. Kittler and Illingworth propose a simpler technique for obtaining these estimates. Suppose that the histogram is thresholded at an arbitrary threshold T, then we can model the two resulting populations by a normal density $h(g|i, T)$ with parameters $(\mu_i(T), \sigma_i(T), P_i(T))$ given by:

$$P_i(T) = \sum_{g=a}^{b} h(g)$$

$$\mu_i(T) = \frac{\sum_{g=a}^{b} g * h(g)}{P_i(T)}$$

procedure *HPPCluster(dataset X, tree Dtree)*

$[\tau, \theta, \alpha]$ = *SplitData*(X);

If($\tau = 0$ *or* $\theta = -\infty$ *or* $\alpha = 0$)

no such projection exists, **RETURN**;

Else split the data

$X_L = \{x_i \in X | \alpha x_i < \tau\}$;

$X_R = \{x_i \in X | \alpha x_i \geq \tau\}$;

Add new node (α, τ) *to Dtree;*

HPPCluster(X_L, *Dtree*);

HPPCluster(X_R, *Dtree*);

Figure 4: The clustering procedure

$$\sigma_i^2(T) = \frac{\sum_{g=a}^{b}(g - \mu_i(T))^2 * h(g)}{P_i(T)}$$

where $a = 0$ if $i = 1$, $a = T + 1$ if $i = 2$, $b = T$ if $i = 1$ and $b = n$ if $i = 2$. The probability of level g being correctly classified is given by

$$e(g, T) = \frac{h(g|i, T)P_i(T)}{h(g)}$$

where $i = 1$ if $g \leq T$ and $i = 2$ otherwise. We wish to find the threshold T that maximizes this function. Since $h(g)$ is independent of i and T it can be safely ignored. Furthermore, since the logarithm is a strictly increasing function, taking the logarithm of both sides will not change the maximizing value. For simplicity, we further multiply by -2 and minimize

$$\varepsilon(g, T) = \left(\frac{g - \mu_i(T)}{\sigma_i(T)}\right)^2 + 2\log\sigma_i(T) - 2\log P_i(T)$$

$i = 1$, if $g \leq T$, $i = 2$, if $g > T$. The overall performance is characterized by

$$J(T) = \sum_g h(g)\dot{\varepsilon}(g, T)$$

and the optimal threshold is given by

$$\tau = \arg\min_T J(T)$$

Writing out $J(T)$ gives

$$\sum_{g=0}^{T} h(g)\left(\left(\frac{g - \mu_1(T)}{\sigma_1(T)}\right)^2 + 2\log\sigma_1(T) - 2\log P_1(T)\right)$$

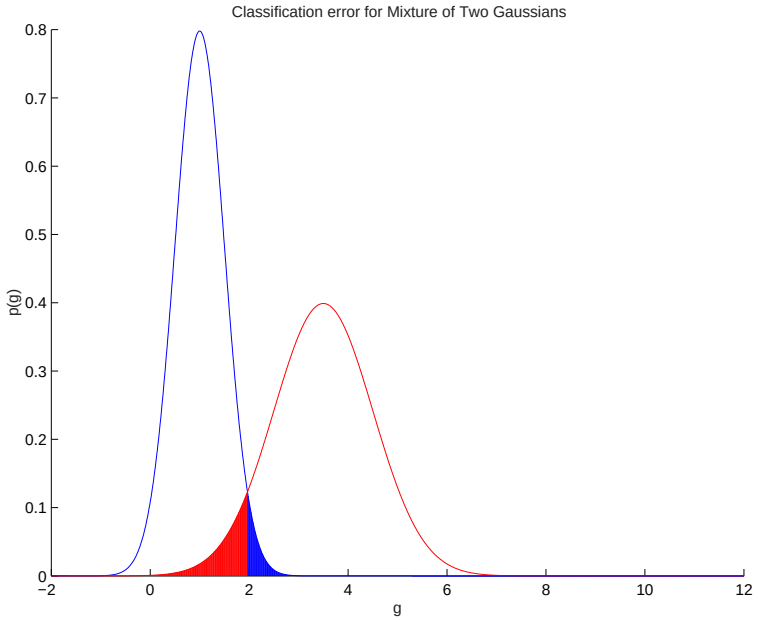

Figure 5: The classification error associated with a mixture of two Normal distributions is given by the integral of the minimum of the two distributions. The Kittler and Illingworth method casts the thresholding problem as a classification problem and seeks the threshold for which the error is minimum.

$$+ \sum_{g=T+1}^{n} h(g) \left(\left(\frac{g - \mu_2(T)}{\sigma_2(T)} \right)^2 + 2 \log \sigma_2(T) - 2 \log P_2(T) \right)$$

which reduces to

$$J(T) = 1 + 2 \left(P_1(T) \log \sigma_1(T) + P_2(T) \log \sigma_2(T) \right)$$
$$-2 \left(P_1(T) \log P_1(T) + P_2(T) \log P_2(T) \right).$$

Note that the tails of the distribution have been truncated by the thresholding operation and therefore the models $h(g|i, T), i = 1, 2$ will be biased estimates of the true mixture components. Cho, Haralick and Yi [11] proposed an improvement of Kittler and Illingworth's criterion function that corrects the biased variance estimates.

Of all possible partitions we prefer the ones with the largest separation between the peaks of the two distributions, relative to their variances. In addition, we would prefer the projections that have the most dramatic variation in the Kittler and Illingworth criterion function. We use as our index function a measure that considers both of these requirements. Once the histogram is thresholded the separation between the peaks of the thresholded distributions is computed by

$$sep = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}.$$

The depth of the criterion function is given by the difference between the criterion function evaluated at the optimal threshold and the value of the criterion function evaluated at the

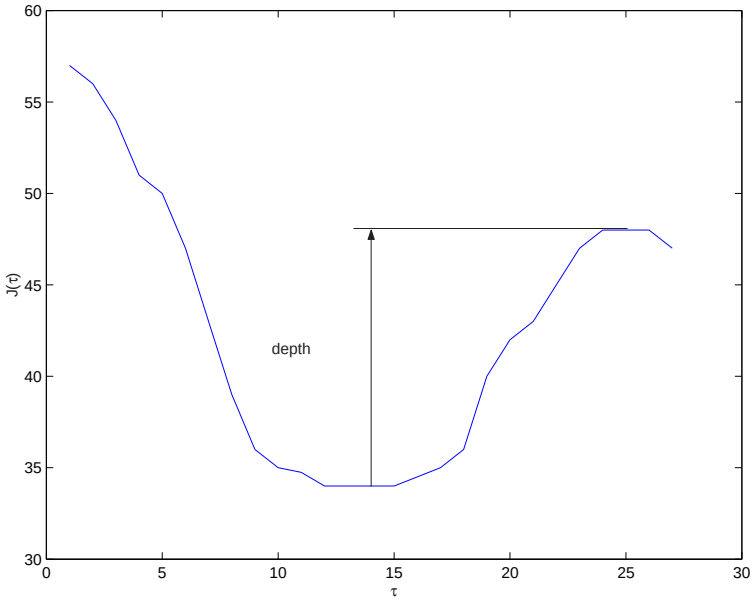

Figure 6: Computing the depth of the criterion function.

closest local maxima, as shown in figure 6. The composite measure of the projection is found by the index function:

$$I(\alpha) = sep * depth.$$

3.2 Searching for Projections

3.2.1 Brute Force

The most straight forward method of projection selection is brute force. Since all projections are unit norm, this amounts to an enumeration of points on a d dimensional unit sphere. For lower dimensional data ($d \leq 5$), we are able to achieve stable and satisfactory results. Brute force methods for iterating over projections are infeasible for large dimensional datasets ($d > 5$) and a more intelligent search procedure is required.

3.2.2 Genetic Algorithms

By viewing the projection vectors as genes we can use a genetic search. Genetic Algorithms were introduced by J. H. Holland in 1975 [30]. Since then they have been successfully applied in solving a number of numerical and combinatorial problems. In most cases genetic algorithms are used in optimization problems when searching for an optimal solution or its approximation (see, for example, survey [39]).

Genetic algorithms are stochastic search algorithms driven by a heuristic, which is represented by an evaluation function, and special random operators: crossover, mutation and selection.

procedure *Genetic Algorithm*
Initialize current population $P \in \mathcal{S}^r$*;*
Compute fitness values $Fit(m)$, $\forall m \in P$*;*
WHILE NOT *the termination condition satisfied* **DO**

If we assume that greater values of function Fit *correspond to the better solutions, then the probability* $Pr(m)$ *of the member* $m \in P$ *to be selected*

$$Pr(m) = \frac{Fit(m)}{\sum_{m_i \in P} Fit(m_i)},$$

Create new members by applying crossover and/or mutation to the selected members;

Generate a new population by replacing members of the current population by the new ones;

Recompute fitness values;

END WHILE LOOP

Figure 7: Structure of the standard Genetic Algorithm

Let $\mathcal{S}$ be a search space. We are looking for an element in $\mathcal{S}$ which is a solution to a given problem. A tuple $P \in \mathcal{S}^r$ (r is a fixed positive integer) is called a *population* and components of P are called *members* of the population. The initial population P_0 is chosen randomly. On each *iteration* $i = 1, 2, \ldots$ Genetic Algorithm produces a new population P_i by means of random operators. The goal is to produce a population which contains a solution to the problem. One iteration of Genetic Algorithm simulates natural evolution. A so-called *fitness function* $Fit : \mathcal{S} \to \mathbb{R}_+$ implicitly directs this evolution: members of the current population P_i with higher fitness value have more impact on generating the next population P_{i+1}. The function Fit measures on how close is a given member m to a solution. To halt the algorithm one has to provide in advance a *termination condition* and check whether it holds or not on each iteration. The basic structure of the standard Genetic Algorithm is given in figure 8. The choice of random operators and evaluating functions is crucial here. This requires some problem specific knowledge and a good deal of intuition.

3.2.3 Combined Methods

Since the genetic algorithm is sensitive to the quality of the initial population a combined technique can be used to improve results. A coarse sampling of the unit sphere is created by brute force. The results of this sample are evaluated and the best projections added to the initial population. Thus, the coarse optimization puts us close to a solution and the genetic algorithm refines this solution.

Another approach is to use a localized gradient based search to refine the results of the genetic algorithm. In this method a population is created at random and allowed to evolve. Once the algorithm has halted local search is applied, starting from each output gene in turn. The best projection after local search is then returned.

procedure *Genetic Algorithm*
Initialize current population of projections $\alpha = \{\alpha_1, ..., \alpha_{popSize}\}$*;*
Compute fitness values $Fit(\alpha_i)$, $\forall \alpha_i \in \alpha$*;*
WHILE NOT *the termination condition satisfied* **DO**

The probability $Pr(\alpha_i)$ *of the member* $\alpha_i \in \alpha$ *to be selected*

$$Pr(\alpha_i) = \frac{Fit(\alpha_i)}{\sum_{\alpha_j \in \alpha} Fit(\alpha_j)},$$

Create new members by applying crossover and/or mutation to the selected members;

Generate a new population by replacing members of the current population by the new ones always keeping the best K *members;*

Recompute fitness values;

Perform Local Optimization for all $m \in$ *current population;*
WHILE *current_best improving* **DO**

Δ *given;*
$current\_best =< c_1, ..., c_d >= m$*;*
for $i = 1, ..., d$
$S^{\pm} =< c_1, ..., c_i \pm \Delta, ..., c_d >$
if $Fit(S^{\pm}) > Fit(current\_best)$
$current\_best = S^{\pm}$

END WHILE LOOP

END WHILE LOOP

Figure 8: Structure of the genetic search algorithm with local search.

3.3 Stopping Conditions

HPPCluster is a divisive algorithm, splitting the dataset into smaller and smaller pieces with each iteration. The algorithm continues to try and split the data until a stopping condition is met. We investigated several stopping conditions based on various measures.

3.3.1 Known Number of Clusters

The simplest stopping condition is to specify the number of clusters in the data and to stop when the specified number of clusters have been found. Since HPPCluster is a hierarchical bi-partitioning algorithm, another approach is to place an upper bound on the number of clusters by specifying a maximum tree depth. For a binary three of depth k, the maximum number of clusters (leaves) is given by 2^k. This approach has the advantage of being less sensitive to the effects of noise, but still allowing one to incorporate prior knowledge of the structure of a given dataset. In many cases the number of clusters is not known in advance and an automated condition is needed. For these cases the algorithm will continue splitting until no interesting projection can be found.

3.3.2 Analyzing the Projected Density

One way to evaluate the interestingness of the projected data is to test the projection for unimodality. If all inspected projections of the data are deemed unimodal, the data is not split any further.

We begin by smoothing the projected density estimate $h(g)$ with a smoothing kernel and perform gradient guided walks along the smoothed histogram, away from the optimal threshold in either direction. At each step of the walks we compute the gradient. If the gradient does not change sign in both of the walks, then we consider the distribution to be unimodal.

This method suffers from sensitivity to noise. In addition the method requires choice of a setting for the width of the smoothing kernel, which is difficult. If the smoothing kernel is narrow it will result in excessive splitting and if the kernel is wide, then structural detail will be ignored.

3.3.3 Analyzing the Criterion Function

The problem with testing the projected data is that there is no smoothness constraint on the data, making the choice of smoothing kernel width vital and resulting in excessive splitting. The criterion function $J(T)$, on the other hand, contains information about the modality of the projection and, in addition, has an explicit smoothness constraint. Thus we can analyze $J(T)$ to assess the modality of the projection by checking for the presence of a global minima. If the only such minima exist at the boundaries, then the projection is considered unimodal. This method takes advantage of the smoothness of the criterion function, but it tends to over split the data due to the randomness of the data.

3.3.4 Analyzing the Relative Heights of Valley and Basin

In many instances stopping conditions based solely on a test for unimodality result in excessive splitting of the data. We would prefer not to split bimodal distributions with significant overlap. Viewing the projected density as a topological surface, the search for a threshold is essentially a search for the basins of the function. The problem with the above mentioned techniques is that they only consider the basin, which gives no indication of the density in the corridor. If the height of the valley is significantly smaller than the height of the basin, then the corridor density is not significant. If the valley density is sufficient, however, then the valley may in fact represent significant structure in the full space and a split should not be made. A coarse estimate of the valley and basin densities can be made from the valley and basin heights, as illustrated in figure 9. If the basin height is small compared to the valley height then no split is made. This method gives a sensitivity control parameter that is intuitively satisfying. Because the measure is computed from a point sample, it suffers from the effects of noise. One way to reduce the effects of noise are to take the measurements over an interval enclosing the optimal threshold. Another disadvantage of the method is the introduction of the ratio parameter, which may be difficult to select.

3.3.5 Analyzing the Relative Integrals of Valley and Basin

The valley basin height ratio is a course estimate of the relative densities of the valley and the basin. A better estimate can be obtained from the integrals of the valley and basin. Having defined a threshold τ the boundaries g_l and g_h of the basin can be determined and the integral of the valley and basin computed. The ratio of the two integrals is formed and used as a criterion. If the integral of the basin is small compared to the integral of the valley (figure 10), then no

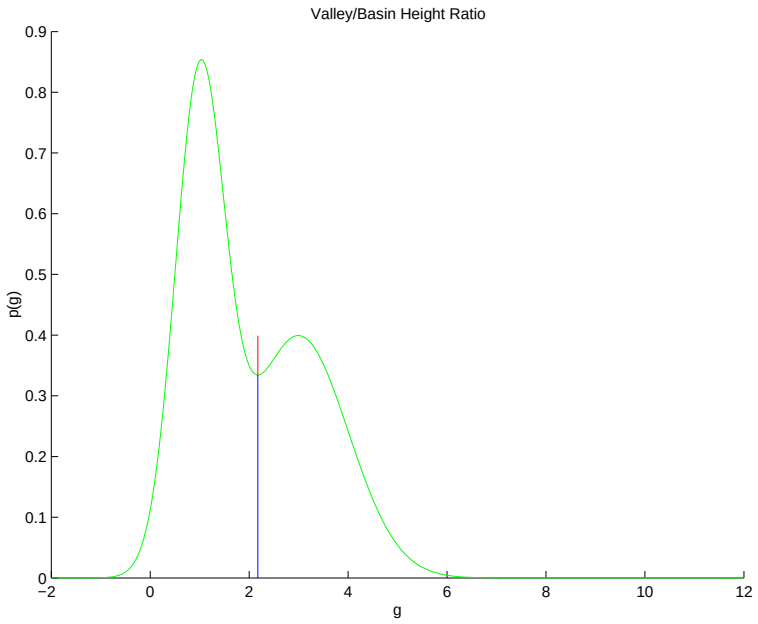

Figure 9: The valley/basin height ratio. If the depth of the basin (the distance from the function evaluated at the optimal threshold to the value of the function evaluated at the minimal peak) is much smaller than the depth of the valley (the height of the function evaluated at the optimal threshold), then the split is rejected.

split is made. While this method is more robust that the method based on the valley/basin height ratio, the problem of determining the value of the ratio parameter remains.

3.3.6 Null Distribution of Uninteresting Distribution

Knowing that uninteresting clusters a drawn from a particular population, we can train the algorithm to recognize an uninteresting cluster based on the distribution of the projected data. If we know something about the structure of an uninteresting cluster, we can construct a distribution of the values of the criterion function to be used in a test of the Null hypothesis that an unknown projection is in fact drawn from the Null distribution. We can empirically determine the null distribution of the value of the criterion function for a training set of clusters that are known to be uninteresting. These clusters may be determined empirically or may be chosen from some parametric form. To construct the null distribution we project a collection of data drawn from a known uninteresting population to various subspaces. For the i^{th} sample from the population we compute the j^{th} projection, evaluate $I(\alpha_j)$ and form a distribution f_I of the occurrence of various values of I. We can form the empirical cumulative distribution F_I for the null distribution and use it in a test of hypothesis that the distribution we are examining in fact comes from a Null distribution. We can set a level of significance ϕ, which is the probability that a distribution that is in fact from the Null distribution will not be accepted as being drawn

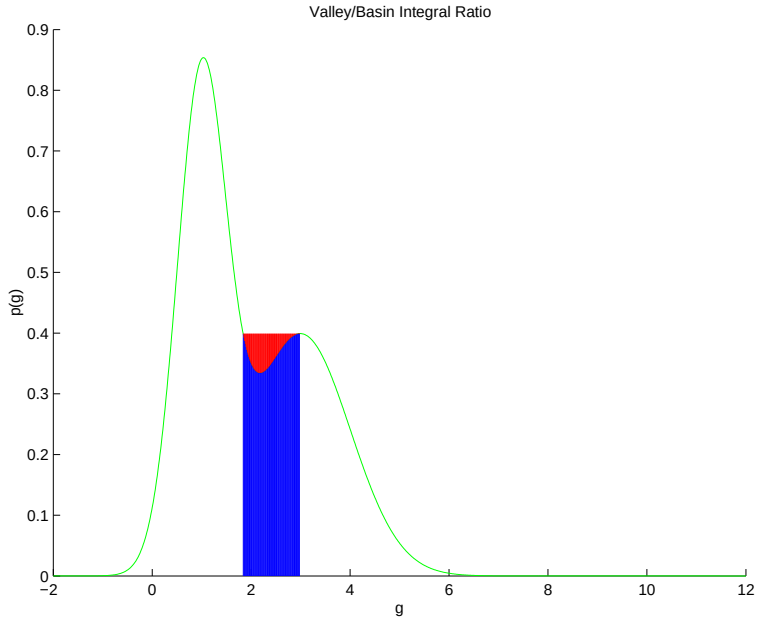

Figure 10: The valley/basin integral ratio. If the volume contained in the basin (the red area above the curve) is much smaller than the volume of the valley (the blue area below the curve), then the split is rejected.

from the Null distribution. Let I_ϕ be the value of the index function such that

$$\int_{-\infty}^{I_\phi} f_I dI = 1 - \phi.$$

We test the following null hypothesis

$$H_0 : unimodal$$

against the alternative hypothesis

$$H_1 : not\ unimodal.$$

Thus for significance level ϕ

$$reject\ H_0\ if\ I > I_\phi$$

and

$$accept\ H_0\ if I \leq I_\phi.$$

Having trained the algorithm we can project a dataset and form its histogram, compute the index $I(\alpha)$ of the projection α and verify with probability $1 - \phi$ that the projection is not-interesting. The value of ϕ, which is the probability that a truly unimodal density will be split by the algorithm, can be viewed as a sensitivity parameter and can be tuned for a particular application. For all of our experiments $\phi = 0.005$. This method was utilized for subsequent experiments due to its empirically determined accuracy and stability, as well its intuitive explanation in terms of the algorithms sensitivity.

4 Cluster Validation and Evaluation

An ideal clustering algorithm classifies points such that points assigned the same class label belong to the same cluster and the number of cluster labels and class labels are identical to the ground-truth classification. To evaluate a clustering algorithm, we face the following problem: How can we map the labels assigned by a clustering algorithm to the ground-truth cluster labels in order to make valid comparisons? In addition, once we have established this mapping, is there a measure that will allow us to compare algorithms? Having such a measure, which algorithms make for valid comparison? In the following section we describe a method for mapping the assigned labels to the ground-truth labels such that a measure of accuracy (the percentage of points that are correctly classified) is maximized. We discuss unsupervised clustering methods that serve as benchmarks and also discuss supervised methods that are used to establish an upper bound on the expected accuracy.

4.1 Computing the Accuracy of a Clustering Algorithm

Given the ground-truth labels and the labels determined by some clustering algorithm we form a contingency table by counting the number of times the algorithm assigns a label j to a cluster point when the actual label is i. We form all possible mappings from the smaller of the actual and assigned label sets onto the larger of these two sets and build a confusion matrix for each mapping. Since the diagonal elements of the confusion matrix represent the number of times that an instance of class i was correctly identified as belonging to class i, we can therefore derive a criterion function that returns a value for each mapping based on the number of correct classifications. The optimal mapping is the one associated with the confusion matrix whose sum along the diagonal (the trace) is maximal.

More formally, given a ground truth data-set consisting of a collection of N feature vectors and corresponding actual cluster label for each vector, we wish to evaluate the performance of a given clustering algorithm.

Let $Y = \{y_1, y_2, ..., y_N\}$ be the set of N d-dimensional feature vectors generated by the data generation methodology, and let $C = \{c_1, c_2, ..., c_{|C|}\}$ be the set of cluster labels. Let $T = \{t_1, t_2, ..., t_N\}$ such that $t_i \in C$ for $i = 1, 2, ..., N$ be the set of cluster labels for each feature vector in the generated data set Y.

Let $L = \{l_1, l_2, ..., l_{|L|}\}$ be the possible labels assigned by the clustering algorithm. Note that $|L|$ is not necessarily equal to the $|C|$ and the labels need not have the same representation, for example actual labels may be alphanumeric characters or strings, while assigned labels may be some subset of the integers. Let $A = \{a_1, a_2, ..., a_N\}$, such that $a_i \in L$ for $i = 1, 2, ..., N$ be the assigned labels that are the output of a clustering algorithm. We form a $|C| \times |L|$ matrix $\mathbf{Z}$, whose (i, j) entry is equal to the number of times that a vector with true label i was assigned label j by the algorithm.

Given $\mathbf{Z}$, we form all possible permutations of the rows or columns of $\mathbf{Z}$, depending on which is smaller. For the j^{th} permutation we compute

$$v_j = \sigma_{i \in \min(|C|,|L|)} \mathbf{Z}[i, i],$$

and choose the permutation with $\arg\max_j v_j$

Having found the optimal permutation, we can form the $\min(|C|, |L|) \times \min(|C|, |L|)$ confusion matrix $\mathbf{M}$. The diagonal elements of $\mathbf{M}$ indicate the number of correct classifications and

$$\sum_{i \in \min(|C|,|L|)} \frac{\mathbf{M}[i, i]}{N},$$

gives the percentage of points correctly classified, providing a measure of the accuracy of the algorithm.

4.2 Cluster Validation

The issue of cluster validity is a complex one [42] further complicated when dealing with real datasets with known class labels, but unknown structure. Knowing the accuracy is not enough to determine the efficacy of a given algorithm without knowing the upper bound on expected accuracy. To assess the quality of the clustering we constructed decision tree classifiers, using various decision rules, for each of the real datasets. The accuracies obtained by these optimal classifiers are used as upper bounds on the expected performance for our algorithm. The notion of comparing the performance of HPPCluster with that of a decision tree classifier is intuitively satisfying due to the similarity between the approaches, the major difference being that HPPCluster is unsupervised, while the classifiers are supervised. A decision tree is a binary tree that makes a decision at each node to discriminate between various input classes. At each node the tree chooses the optimal partitioning of k classes into two classes such that class purity in the child nodes is optimized. For each dataset several trees were constructed, using various decision rules.

4.3 Comparison to Unsupervised Methods

4.3.1 k-means

The k-means algorithm is an iterative unsupervised non-hierarchical method [17] for partitioning a dataset with N elements into k disjoint subsets. The algorithm is shown in figure 11. For a given k, the centers $\mu_1, ..., \mu_k$ are chosen by some procedure (usually random) and each point $x \in X$ is assigned to the center μ_i to which it is closest, based on some distance measure $d(x, \mu_i)$. The procedure iterates until the centers converge or some maximum number of iterations is exceeded. Issues when using the algorithm include the choice of k, the initialization of $\mu_1, ..., \mu_k$ and the choice of distance funtion $d()$. For our experiments the initial centers are chosen at random and a Euclidean distance measure is used.

procedure *KMeans(dataset X, int k)*

Initialize $\mu_1, ..., \mu_k$*;*

while $\mu_1, ..., \mu_k$ *have not converged:*

classify each $x \in X$ *to* $\arg\min d(x, \mu_i)$, $i = 1, ..., k$*;*

recompute $\mu_1, ..., \mu_k$*;*

return $\mu_1, ..., \mu_k$*;*

Figure 11: The k-means clustering procedure

4.3.2 EM algorithm

The Expectation Maximization (EM) algorithm [14], [38], [17] can be utilized to estimate the parameters of a mixture model. Given a model for the data X with parameters θ, the EM

algorithm tries to find θ such that $P(X|\theta)$ is maximized. Let θ_i be the current best estimate and $Q(\theta;\theta_i)$ be the likelihood of the data for a given θ. At each step of the algorithm we seek the θ that maximizes Q, given our last best estimate θ_i. For

procedure *EM(dataset X, int k, threshold T)*

Initialize θ_0*;*

while : $Q(\theta_{i+1};\theta_i) - Q(\theta_i;\theta_{i-1}) \leq T$

E step: compute $Q(\theta;\theta_i)$*;*

M step: $\theta_{i+1} = \arg\max_\theta Q(\theta,\theta_i)$

return θ_{i+1}*;*

Figure 12: The EM algorithm

4.4 Comparison to Supervised Methods

4.4.1 Binary Decision Tree

A decision tree classifier makes a class assignment through a hierarchical design procedure. The classification process can be described by means of a tree, in which at least one terminal node is associated with each pattern class, and the interior nodes represent various collections of mixed classes. In particular, the root node represents the entire collection of classes into which a unit may be classified. Figure 13 shows a typical binary decision tree classifier. When an unknown unit enters the decision tree at the root node, a decision rule associated with the root node is applied to the unit's measurement vector to determine the descendant path that the unit will follow. This process is repeated until a terminal node is reached. Every terminal node has an associated class to which the unit is finally assigned.

For the construction of a decision tree, we need a training set of feature vectors with true class labels. The maximum limit on the depth of the tree is $log_2 N - 1$ where N is the number of feature vectors used for training. Let $U = \{u_k : k = 1...N\}$ be the training set to be used to design the binary tree classifier. Each unit U_k has an associated measurement X_k associated with a known true class. At any non-terminal node, let Ω^n be the set of M^n classes still possible for a unit at node n. Let $U^n = \{u_k^n : k = 1...N^n\}$ be a subset of N^n training units associated with a node n. Therefore,

$$N^n = \sum_{c=1}^{M^n} N_c^n.$$

Now we can define how the decision rule works at node n. Consider a unit u_k^n which has a measurement vector x_k^n. Let f be the decision rule that is used at every non-terminal node. The decision rule used in the classifier is described in section 4.5.

If $f(x_k^n)$ is less than or equal to a threshold, then u_k^n is assigned to class Ω_{left}^n, otherwise it is assigned to class Ω_{right}^n. An assignment to Ω_{left}^n means that the unit descends to the left child node and Ω_{right}^n means that the unit descends to the right child node.

Given a discriminant function f, we sort the units in the set U^n in an ascending order according to their discriminant function value. Let x_k^n, for $k = 1, \cdots, N^n$, be the measurement

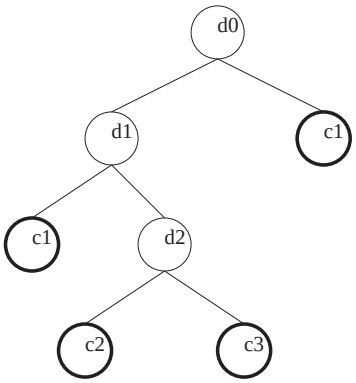

Figure 13: A binary decision tree. Bold circles represent terminal nodes, and plain circles represent nonterminal decision nodes. Note that class $c1$ can be found in more than one leaf node.

vectors sorted in such a way that $f(x_k^n) \leq f(x_{k+1}^n)$ for $k = 1, \cdots, N^n - 1$. Let w_k^n be the true classes associated with measurement vectors x_k^n. Then a set of candidate thresholds T^n for the decision rules are defined by

$$T^n = \left\{ \frac{f(x_{k+1}^n) + f(x_k^n)}{2} \mid w_{k+1}^n \neq w_k^n \right\}$$

For each threshold value, unit u_k^n is classified by using the decision rule specified above. We count the number of samples n_{Lc}^t assigned to Ω_{left}^n whose true class is c and we count the number of samples n_{Rc}^t assigned to Ω_{right}^n whose true class is c.

$$n_{Lc}^t = \#\left\{u_k^n | f(x_k^n) \leq t \text{ and } w_k^n = c\right\}$$

$$n_{Rc}^t = \#\left\{u_k^n | f(x_k^n) > t \text{ and } w_k^n = c\right\}$$

Let n_L^t be the total number of samples assigned to Ω_{left}^n and n_R^t be the total number of samples assigned to Ω_{right}^n, that is

$$n_L^t = \sum_{c=1}^{M^n} n_{Lc}^t$$

$$n_R^t = \sum_{c=1}^{M^n} n_{Rc}^t$$

The purity of the assignment made by node n is defined as

$$PR_n^t = \sum_{c=1}^{M^n} \left(n_{Lc}^t \ln p_{Lc}^t + n_{Rc}^t \ln p_{Rc}^t \right)$$

where

$$p_{Lc}^t = \frac{n_{Lc}^t}{n_L}$$

and

$$p_{Rc}^{t} = \frac{n_{Rc}^{t}}{n_R}.$$

The discriminant threshold t is chosen such that it maximizes purity value PR_n^t. Purity is such that it gives a maximum value when the samples are completely separable. The expansion of the tree is stopped if the decision rules used at the nodes cannot be applied to smaller training sets or if the number of feature vectors goes smaller than a predetermined value.

4.5 Decision Rules

4.5.1 Thresholding Decision Rule

The simplest form for a linear decision rule is a comparison of one measurement component to a threshold. This is called a thresholding decision rule. One measurement component is selected and a set of candidate thresholds, T, is computed for that component. For each threshold in the set T, all units in the training set U_n are classified into either class Ω_{LEFT} or class Ω_{RIGHT} according to their value of the selected measurement component. The number of units for each assigned to class Ω_{LEFT} and to class Ω_{RIGHT} is counted, and the entropy purity is computed from the resulting classification. A threshold is selected from the set of threshold candidates T such that, when the set U^n is classified with that threshold, a maximum purity in the assignment results. This process is repeated for all possible measurement components, and the component and the threshold that yield an assignment with the maximum purity are selected.

When a feature vector is input to a decision tree, a decision is made at every non-terminal node as to what path the feature vector will take. This process is continued until the feature vector reaches a terminal node of the tree, where a class is assigned to it.

4.5.2 Fisher's Linear Decision Rule

Let x_k be the $d-$dimensional feature vector associated with a unit u_k to be classified. Then the linear discriminant function is a linear function of the form

$$f(x_k) = V^t x_k + v_{d+1}$$

Where V is a weighting vector and v_{d+1} is called the threshold. This form of linear discriminant function uses a hyperplane to separate the feature space into two regions. The decision rule assigns u_k to one region if $f(x_k) > 0$ and to the other region if $f(x_k) < 0$. If $f(x_k) = 0$, the indeterminant case may be resolved arbitrarily. It is easy to see that using a linear discriminant function is equivalent to projecting a pattern vector x_k onto a line in the direction of vector V and comparing the position of the projection with that of the threshold v_{d+1}. Fisher's linear discriminant function is obtained by maximizing the Fisher's discriminant ratio, which, as described below, is the ratio of the projected between-class scatter to the projected within-class scatter.

First, this decision rule needs a prior partition of the set of categories. This partition induces a partition on the unit training set U^n. Let v be the unknown weighting vector and $z_k = v^t x_k$ be the projected point associated with unit u_k. For $i = 1$ or 2 (designating class Ω_{LEFT} and class Ω_{RIGHT}) let the estimated class conditional mean vector μ_i and the mean vector of the mixture distribution μ be defined as

$$\mu_i = \frac{1}{N_i} \sum_{u_k \in i} x_k$$

$$\mu = \sum_{i=1}^{2} P_i \mu_i$$

where N_i and P_i represent the number of samples and the probability of the class i in the training sample associated with the node, respectively. Then the between-class scatter matrix S_b for a two-class case is given by

$$\begin{aligned} S_b &= \sum_{i=1}^{2} P_i(\mu_i - \mu)(\mu_i - \mu)^t \\ &= P_1 P_2(\mu_1 - \mu_2)(\mu_1 - \mu_2)^t \end{aligned}$$

If we let S_i be the class conditional scatter matrix of class i, then

$$S_i = \frac{1}{N_i} \sum_{u_k \in i} [(x_k - \mu_i)(x_k - \mu_i)^t], \ i = 1, 2$$

and if we let S_w be the average class conditional scatter matrix , then

$$S_w = \sum_{i=1}^{2} P_i S_i$$

Finally, if we let S designate the scatter matrix of the mixture distribution,

$$S = \frac{1}{N_1 + N_2} \sum_{k=1}^{N^n} [(x_k - \mu)(x_k - \mu)^t]$$

then

$$S = S_w + S_b$$

In the one-dimensional projected space of $z_k = v^t x_k$, one can easily show that the projected between-class scatter s_b and the projected within-class scatter s_w are expressed as

$$s_b = v^t S_b v$$

$$s_w = v^t S_w v$$

Then the Fisher discriminant ratio is defined as

$$F(v) = \frac{s_b}{s_w} = \frac{v^t S_b v}{v^t S_w v}$$

When using the Fisher discriminant ratio, we seek to compute an optimum direction v, such that orthogonally projected samples are maximally discriminated. The optimum direction v can be found by taking the derivative of $F(v)$ and setting it to zero, as

$$\bigtriangledown F(v) = (v^t S_w v)^{-2}(2 S_b v v^t S_w v - 2 v^t S_b v S_w v) = 0$$

From this equation it follows that

$$v^t S_b v S_w v = S_b v v^t S_w v$$

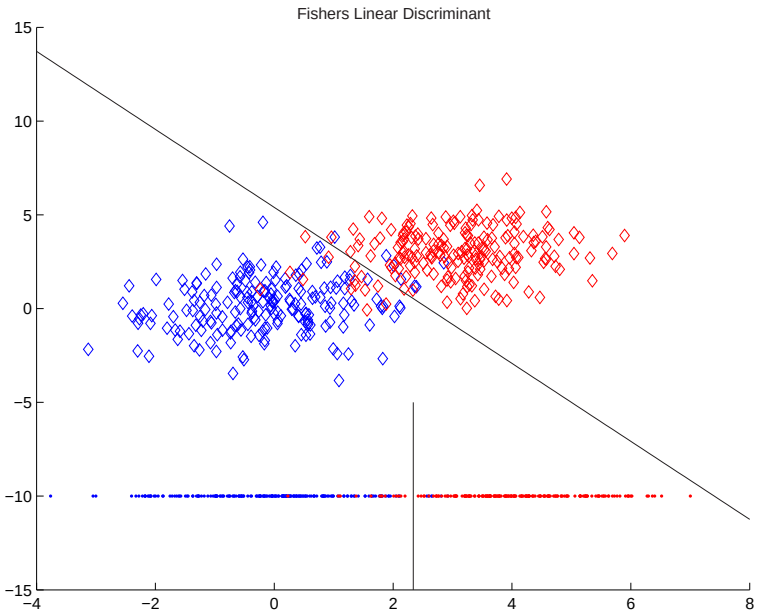

Figure 14: Fisher's linear discriminant tries to find the hyperplane to maximize the projected distances between classes.

If we divide both side by the quadratic term $v^t S_b v$, then

$$\begin{aligned} S_w v &= (\frac{v^t S_w v}{v^t S_b v}) S_b v \\ &= \lambda S_b v \\ &= \lambda P_1 P_2 (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t v \\ &= \lambda \kappa (\mu_1 - \mu_2) \end{aligned}$$

Where λ and *kappa* are some scalar values defined as

$$\lambda = \frac{v^t S_w v}{v^t S_b v}$$

$$\kappa = P_1 P_2 (\mu_1 - \mu_2)^t v$$

Thus we have the weighting vector v as

$$v = K S_w^{-1} (\mu_1 - \mu_2)$$

where $K = \lambda\kappa$ is a multiplicative constant. The threshold v_{d+1} is the value that maximally discriminates between two classes.

4.5.3 Support Vector Machine Decision Rule

A Support Vector Machine (SVM) is a statistical classifier that attempts to construct a hyperplane as the decision surface in such a way that the margin of separation between positive and

negative examples is maximized. Classification involves representing data by a feature vector

$$x = (x_1, \ldots, x_n)^T ,$$

consisting of n separate features. Given a training set $\{x_i, y_i\}$, $1 \leq i \leq N$,

$$w^T x + \omega_0 = 0$$

is the equation of the decision surface (hyper-plane) that achieves the separation. Define the margin of separation ρ to be the distance between the hyper-plane and the closest data point. The SVM tries to find a hyper-plane so that ρ is maximized [43, 44]. Consider a two class classification problem, with classes ω_+ and ω_- . Given a training set which consists of N observations and corresponding "true" labels y_i , we wish to find a hyper-plane which will separate the two classes such that all points on one side of the hyper-plane will be labelled +1, all points on the other side will be labelled -1:

$$x_i \cdot w + \omega_0 \geq +1, \quad for \quad y_i = +1$$

$$x_i \cdot w + \omega_0 \leq -1, \quad for \quad y_i = -1$$

or equivalently

$$y_i \left(x_i \cdot w + \omega_0 \right) - 1 \geq 0, \quad \forall i.$$

Where $\{x_i, y_i\}$, is the training data, $i = 1, \ldots, l$, $y_i \in \{-1, 1\}$ is the label of the i^{th} sample, $x_i \in \Re^d$ is an d dimensional input vector for the i^{th} sample, w is an adjustable weight vector, and ω_0 is a bias term. Let $r_+(r_-)$ be the shortest distance from the hyper-plane to the closest positive(negative) point and define the margin of separation $\rho = r_+ + r_-$. The optimal hyper-plane is the one that maximizes ρ . Define a discriminant function $g(x) = w_0^T x + \omega_0$, where w_0, ω_0 are the parameters of the optimal hyper-plane, which gives the distance from x to the optimal hyper-plane. We can maximize the margin by minimizing the cost function

$$\Phi(w) = \frac{1}{2} w^T w = \frac{1}{2} \left\| w \right\|^2 ,$$

subject to the constraint that

$$y_i \left(x_i \cdot w + \omega_0 \right) - 1 \geq 0, \forall i.$$

We solve this optimization by means of Lagrange multipliers α_i , $i = 1, \ldots, l$ by forming the Lagrangian

$$Lp \equiv \frac{1}{2} \left\| w \right\|^2 - \sum_{i=1}^{l} \alpha_i y_i \left(x_i \cdot w + \omega_0 \right) + \sum_{i=1}^{l} \alpha_i$$

and minimize Lp with respect to w, ω_0 while requiring that derivatives of Lp with respect to all the α_i vanish, subject to the constraint that $\alpha_i \geq 0$. We can equivalently solve the dual problem of maximizing Lp , subject to the constraints that the gradient of Lp with respect to w and ω_0 vanish, subject to the constraint that $\alpha_i \geq 0$. This gives:

$$w = \sum_i \alpha_i y_i x_i,$$

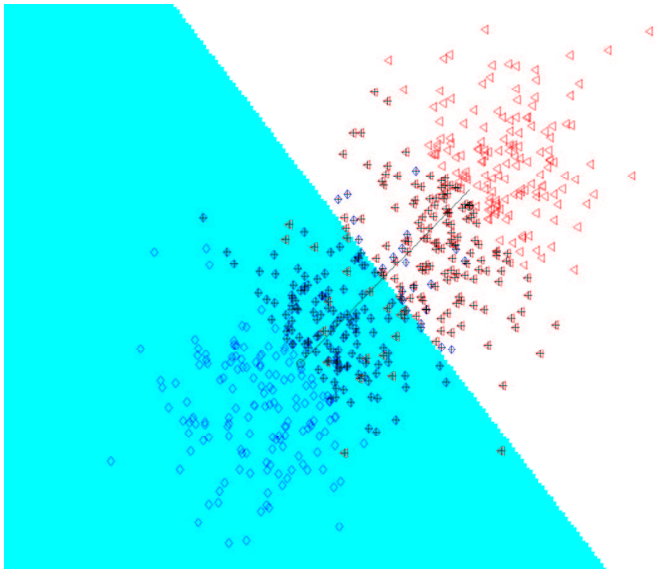

Figure 15: Support Vector Machines find the hyperplane that maximizes the margin between two classes. Example of a linearly separable data set.

$$\sum_i \alpha_i y_i = 0.$$

Substituting these constraints back into the Lagrangian gives:

$$L_D \equiv \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

The small subset of non-zero Lagrange multipliers that result are called support vectors. The support vectors lie closest to the decision boundary, making them the critical elements of the training set [26]. After the optimization

$$g(x) = \sum_{i=1}^{l} \alpha_i y_i x_i^T x + \omega_0,$$

showing that the distance can be computed as a weighted sum of the training data and the Lagrange multipliers, and that the training vectors x_i are only used in inner products.

4.6 Entropy Reduction Criterion

To construct a binary linear decision tree classifier, at each nonterminal node we need to find a linear decision function which splits the training subset arrived at that node into another two subsets in an optimal way so that each of the split sample subsets becomes as pure as possible. Entropy is a concept used to measure how impure a sample set is. A completely pure sample will attain the lowest entropy, which is zero.

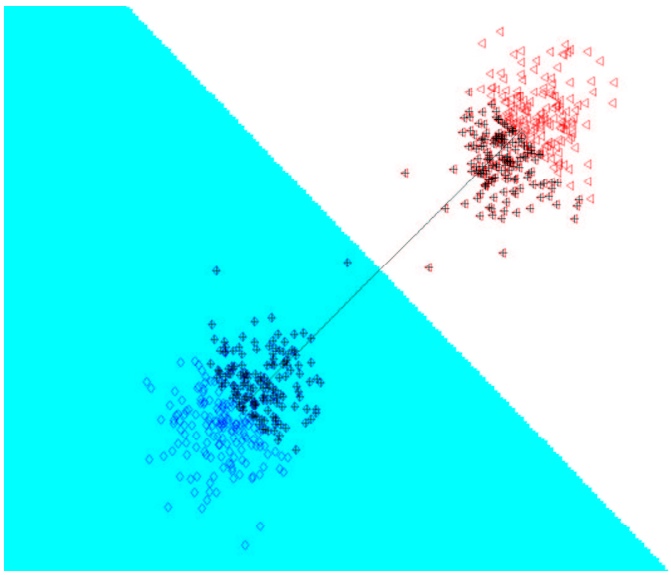

Figure 16: Support Vector Machines find the hyperplane that maximizes the margin between two classes. Example of a data set that is not linearly separable.

Let $X_t = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{N_t}\}$ be the training subset arrived at the current node t with the sample size N_t, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{id}, 1)^t$ is the augmented feature vector and d denotes its original dimension. Assume $X_t = X_t^1 \bigcup X_t^2 \bigcup \cdots \bigcup X_t^c$, where c represents the number of classes and X_t^i contains all samples in X_t which belongs to class w_i, i.e.,

$$X_t^i = \{\mathbf{x} | \mathbf{x} \in X_t \; and \; \mathbf{x} \; from \; w_i\}.$$

Then the entropy of X_t is defined as

$$E(X_t) = \sum_{i=1}^{c} -p(w_i|X_t) \ln p(w_i|X_t),$$

where

$$p(w_i|X_t) = \frac{\#X_t^i}{\#X_t}$$

and "#" denotes the size of a set.

It is easy to verify that the entropy of X_t is zero if and only if X_t becomes "pure", i.e., contains only samples from one class. Assume that a linear decision function $\mathbf{w}^T\mathbf{x}$ splits X_t into $X_{tL}(\mathbf{w})$ and $X_{tR}(\mathbf{w})$, i.e.,

$$X_t = X_{tL}(\mathbf{w}) \bigcup X_{tR}(\mathbf{w}),$$

and

$$X_{tL}(\mathbf{w}) = \{\mathbf{x} | \mathbf{x} \in X_t \; and \; \mathbf{w}^T\mathbf{x} < 0\},$$
$$X_{tR}(\mathbf{w}) = \{\mathbf{x} | \mathbf{x} \in X_t \; and \; \mathbf{w}^T\mathbf{x} < 0\},$$

where $\mathbf{w} = (w_1, w_2, \cdots, w_d, w_{d+1})^T$ is a weight vector of dimension $d+1$. The total entropy after the split is defined as the weighted average of entropies of sample subsets $X_{tL}(\mathbf{w})$ and $X_{tR}(\mathbf{w})$,

$$E'(X_t, \mathbf{w}) = p(X_{tL}(\mathbf{w})|X_t)E(X_{tL}(\mathbf{w})) + p(X_{tR}(\mathbf{w})|X_t)E(X_{tR}(\mathbf{w})),$$

where

$$p(X_{tL}|X_t) = \frac{\#X_{tL}}{\#X_t},$$

and

$$p(X_{tR}|X_t) = \frac{\#X_{tLR}}{\#X_t}.$$

The entropy reduction is thus defined as

$$\triangle E(X_t, \mathbf{w}) = E(X_t) - E'(X_t, \mathbf{w}).$$

The criterion of the maximum entropy reduction requires a linear decision vector maximizing the entropy reduction $\triangle E(X_t, \mathbf{w})$.

4.6.1 Thresholding Decision Rule

For the case of thresholding decision rule, the tree size can become very large due to the limitation that only one feature can be used at each nonterminal node. The entropy reduction criterion is used to decrease the tree size.

At each nonterminal node, the entropy before distinction E_B and the entropy after distinction E_A are computed. Let θ be a predetermined value. If $E_B - E_A > \theta$, we keep the distinction. Otherwise, the node will be deleted.

4.6.2 Fisher's Linear Decision Rule and SVM Decision Rule

At each nonterminal node $\frac{k!(k-1)!}{2}$ combinations of 2 classes are inspected, instead of $2^k/2 - 1$ combinations to get a distinction. The distinction which provides the biggest entropy reduction $E_B - E_A$ is used.

5 Experiments

To verify the effectiveness of our method the algorithm was implemented in the C++ programming langauge. We ran several experiments on various real and synthetic datasets with known groundtruth. We performed a series of experiments designed to assess the sensitivity of the algorithm to various parameters. For all subsequent experiments crossover rate was set to 0.8, mutation rate to 0.4, population size was set at 20, 100 histogram bins where used and α was set to 0.01. Since the genetic optimization is a probabilistic algorithm, all experiments where performed multiple times and summary results presented. In particular we present the resulting mean, standard deviation and median for each set of 50 runs.

5.1 Synthetic Data Generation

We used a synthetic data generation procedure proposed by Bolton and Krznowski [7], to construct mixtures of multivariate normal distributions for use in our experiments. The means

where chosen from:

$$\mu_1 = (0, 0, 0, 0, 0, 0, 0, 0)$$
$$\mu_2 = (4, 0, 0, 0, 0, 0, 0, 0)$$
$$\mu_3 = (4, 1, 0, 0, 0, 0, 0, 0)$$
$$\mu_4 = (0, 1, 0, 0, 0, 0, 0, 0)$$
$$\mu_5 = (4, 1, 1, 0, 0, 0, 0, 0)$$
$$\mu_6 = (4, 0, 1, 0, 0, 0, 0, 0)$$
$$\mu_7 = (0, 1, 1, 0, 0, 0, 0, 0)$$
$$\mu_8 = (0, 0, 1, 0, 0, 0, 0, 0)$$

and all classes had the following covariance structure:

$$\Sigma = \begin{bmatrix} 0.8 & -0.15 & 0.15 & 0 & 0 & 0 & 0 & 0 \\ -0.15 & 0.05 & -0.025 & 0 & 0 & 0 & 0 & 0 \\ 0.15 & -0.025 & 0.05 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \end{bmatrix}$$

Dataset `bolton_2` was generated using mean vectors μ_1 and μ_2 and was intended to be easily separable. Datasets `bolton_4` and `bolton_8` where generated using mean vectors $\mu_1, ..., \mu_4$ and $\mu_1, ..., \mu_8$, respectively.

5.2 Real Data

We ran HPPCluster on the following real datasets chosen from the public domain:

- crab: Australian crab data used in [9]. There are 200 records involving observations on five variables for crabs divided in groups of 50 according to gender (male and female) and species (Blue or Orange).
- glass: Crime Scene Glass Identification Database. Collected by B. German of the Central Research Establishment, Home Office Forensic Science Service, Aldermaston, Reading, Berkshire.
- diabetes: Pima Indians Diabetes data. From the National Institute of Diabetes and Digestive and Kidney Diseases. The binary-valued class variable investigated whether a population that lived near Phoenix, Arizona, shows signs of diabetes according to World Health Organization criteria. Used in [12].
- shuttle: Data from the position of radiators within the Space Shuttle. Provided by Jason Catlett Basser of the Department of Computer Science, University of Sydney, N.S.W., Australia and NASA. Used in [12].

5.3 Limiting Tree Depth

We performed several experiments to determine the effectiveness of the stopping criterion. For synthetic datasets of $c = 2^k$ classes, the tree depth was limited to depth $D = k$. The results where compared with those obtained when the tree was limited to a depth of 10, giving

Name	Dimension	Classes	Number Records
bolton_2	8	2	600
bolton_4	8	4	1200
bolton_8	8	8	2400
crab	5	4	200
glass	9	6	214
diabetes	8	2	768
shuttle	9	7	14500

Figure 17: Characteristics of the datasets used for experiments. The number of classes is specified by the ground truth labelling of the data.

Name	Depth	Mean	Max	Min	Median	std dev
bolton_2	1	0.9980	1.0000	0.9851	1.0000	0.0043
bolton_2	10	0.9812	1.0000	0.9500	0.9825	0.0107
bolton_4	2	0.8153	0.9992	0.5017	0.7500	0.1220
bolton_4	10	0.9689	0.9883	0.9475	0.9683	0.0086
bolton_8	3	0.8513	0.9879	0.4888	0.8608	0.1198
bolton_8	10	0.9608	0.9813	0.8429	0.9646	0.0246

Figure 18: Effect of limiting tree depth for synthetic data.

$2^{10} = 1024$ possible classes. This limit was chosen to represent a reasonable maximum in order to accommodate computation, though a higher limit could have been chosen. Experiments showed that limiting the depth of the tree can result in degraded performance and for remaining experiments tree depth was set at 10.

5.4 Brute Force

For datasets with less than 5 dimensions, we ran the algorithm in brute force mode in order to verify that the optimization procedure was in fact reaching the global optima. For larger datasets, brute force is infeasible.

5.5 Combined Optimization

The initial population for the genetic algorithm was shown to affect the quality of the results obtained. We experimented with several methods of generating an initial population:

- Random: All members are chosen at random, as this method produced the poorest results it was not investigated further.
- Random Plus Coordinated Axes: The coordinate axes are added to the population in addition to randomly chosen members.
- Random plus Brute Force: A coarse sampling of 1000 projections from the space of possible projections is taken, together with the coordinate axes. From this set of results the top $k = 5$ elements are chosen and the remaining members chosen at random.

Name	Mean	Max	Min	Median	std dev
bolton_2	0.9812	1.0000	0.9500	0.9825	0.0107
bolton_4	0.9689	0.9883	0.9475	0.9683	0.0086
bolton_8	0.9608	0.9813	0.8429	0.9646	0.0246
crab	0.4585	0.7500	0.2500	0.3850	0.2191
glass	0.4700	0.5243	0.3738	0.4709	0.0300
diabetes	0.6248	0.6406	0.5482	0.6322	0.0213
shuttle	0.8423	0.9060	0.7841	0.8513	0.0469

Figure 19: Comparison of average results obtained by HPPCluster using Random plus Axes initialization.

Name	Mean	Max	Min	std dev	Median
bolton_2	0.9915	1.0000	0.9750	0.0076	0.9933
bolton_4	0.9723	0.9925	0.7358	0.0348	0.9779
bolton_8	0.9568	0.9792	0.8417	0.0403	0.9710
crab	0.6565	0.7500	0.5600	0.0589	0.6625
glass	0.4282	0.5000	0.3544	0.0254	0.4199
diabetes	0.6276	0.6471	0.6094	0.0107	0.6263
shuttle	0.8644	0.9232	0.7841	0.0580	0.9018

Figure 20: Comparison of average results obtained by HPPCluster using Random plus Brute initialization.

The use of coarse sampling improved the average results obtained for most datasets. The most dramatic increase is seen in the case of the crab data, where the mean increased and variance decreased significantly. Overall the addition of a coarse brute force sample in the initial population increased the performance and made the procedure more stable.

5.6 Comparison to Unsupervised Classifiers

For this set of experiments the HPPCluster algorithm was applied to a variety of datasets. The average and median accuracy values where compared to results obtained by several unsupervised classifiers. Figure 5.6 shows a graph of the results. HPPC significantly outperforms EM for all datasets except glass, and outperforms k means for all datasets glass and diabetes. Note, however, that when HPPC does better, it usually does much better and when it does worse it does not do much worse. In other words the performance that we possibly gain far outweighs the possible performance degradation. Also note that HPPC is relatively stable, while both k means and EM tend to be very sensitive to the initial estimate.

5.7 Comparison to Supervised Classifiers

For this set of experiments the HPPCluster algorithm was applied to a variety of datasets. The average and median accuracy values where compared to results obtained by several classifiers. These comparisons are intended to show that the algorithm will in fact achieve a significant level of the maximum obtainable accuracy. The clustering algorithm can never match the results of a supervised algorithm for a variety of reasons. The actual structure of the data may have no

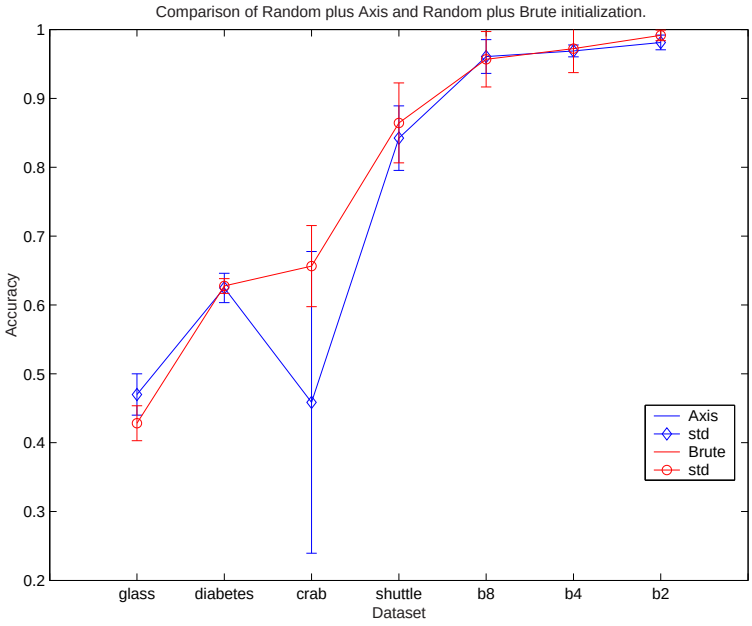

Figure 21: Comparison of initialization with and without a coarse brute force sampling.

Name	Mean	Max	Min	std dev	Median
bolton_2	1.000000	1.000000	1.000000	0.000000	1.000000
bolton_4	0.813333	1.000000	0.809167	0.026941	0.809167
bolton_8	0.636442	0.998750	0.494167	0.116215	0.621042
crab	0.526300	0.645000	0.335000	0.142313	0.645000
glass	0.465631	0.519417	0.368932	0.033915	0.475728
diabetes	0.523438	0.523438	0.523438	0.000000	0.523438
shuttle	0.607425	0.729103	0.414483	0.079901	0.619586

Figure 22: Results obtained using EM algorithm to fit a mixture of Gaussians to the data (1000 iterations maximum). Each experiment was repeated 50 times and summary statistics reported.

Name	Mean	Max	Min	std dev	Median
bolton_2	0.9862	0.9867	0.9850	0.0008	0.9867
bolton_4	0.5128	0.5200	0.4500	0.0099	0.5171
bolton_8	0.3484	0.4688	0.2808	0.0438	0.3356
crab	0.3321	0.3400	0.3200	0.0083	0.3350
glass	0.5120	0.5291	0.4223	0.0212	0.5194
diabetes	0.6602	0.6602	0.6602	0.0000	0.6602
shuttle	0.6888	0.7142	0.4821	0.0344	0.6909

Figure 23: Results obtained using k-means algorithm to fit a mixture of Gaussians to the data. Each experiment was repeated 50 times and summary statistics reported.

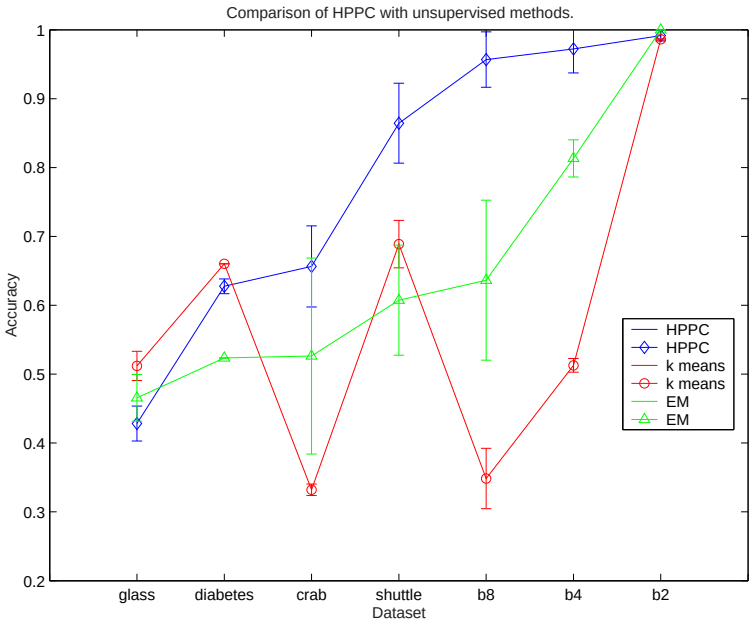

Figure 24: Comparison of HPPC with other unsupervised methods.

Name	I_{PPC}
bolton_2	NA
bolton_4	NA
bolton_8	0.8775
crab	0.9150
glass	0.4533
diabetes	NA
shuttle	NA

Figure 25: Results obtained using Bolton and Krzanowski's I_{PPC} clustering algorithm. Results originally reported in [7]. The starting seed was the initial classification.

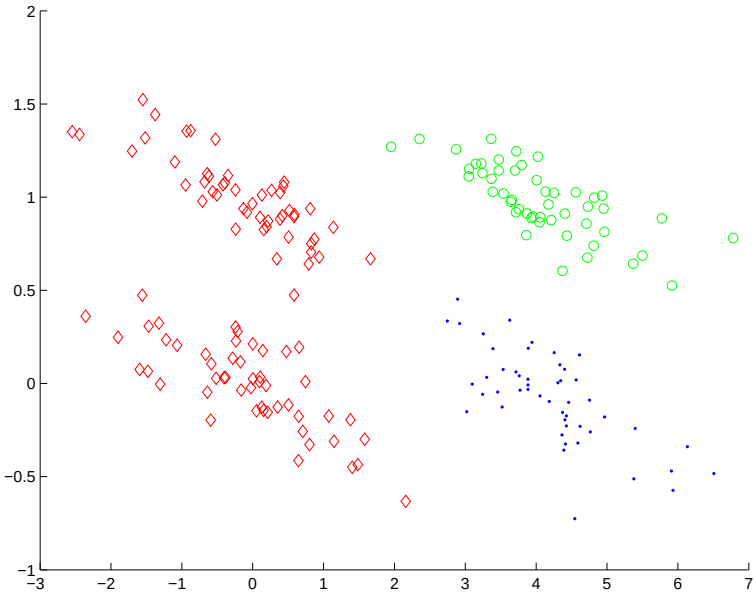

Figure 26: An example of a simple dataset in which the structure of the data does not correspond to the class labels. For such a dataset we would expect an accuracy of 100% for a supervised algorithm. A clustering algorithm would result in an accuracy of approximately 75%.

relationship to the human assigned class, or that relationship may be obscured by the structure. Consider the simple example shown in figure 26. For this dataset HPPCluster will find four distinct clusters, though there exist only three unique classes, and would therefore have an accuracy measure of around 75%, rather than the 100% that is achievable using supervised methods. Empirical results indicate that HPPCluster is able to achieve good results in the case of linearly separable data. Figure 5.7 shows that HPPC follows the overall trend of the supervised methods.

6 Conclusions

We have shown that our algorithm can perform very well with a variety of real and synthetic data. The algorithm requires no external input parameters, is able to handle large dimensional data, is scalable to large datasets and produces a compact description of the clusters found in the form of a binary decision tree which can be used for classification purposes. Each node in the tree stores an optimal projection with its optimal threshold and the leaves of the tree store the actual clusters. Experimental results show the effectiveness of the technique, particularly in the case of linearly separable data.

6.1 Extensions to Non-Linearly Separable Data

It is important to note that while the HPPCluster algorithm is only able to locate separations in linearly separable data, the methodology can be extended to the case of non-linearly separable

Name	Mean	Median	std dev	svm	Fisher	Threshold
bolton_2	0.9812	0.9825	0.0107	1.0000	1.0000	1.0000
bolton_4	0.9689	0.9683	0.0086	0.9992	1.0000	0.9858
bolton_8	0.9608	0.9646	0.0246	NA	0.9983	0.9850
crab	0.4585	0.3850	0.2191	0.9850	0.9800	0.6900
glass	0.4700	0.4709	0.0300	0.7850	0.6822	0.6589
diabetes	0.6248	0.6322	0.0213	NA	0.8125	0.8346
shuttle	0.8423	0.8513	0.0469	NA	0.9959	0.9997

Figure 27: Comparison of average results obtained by HPPCluster with several supervised learning methods.

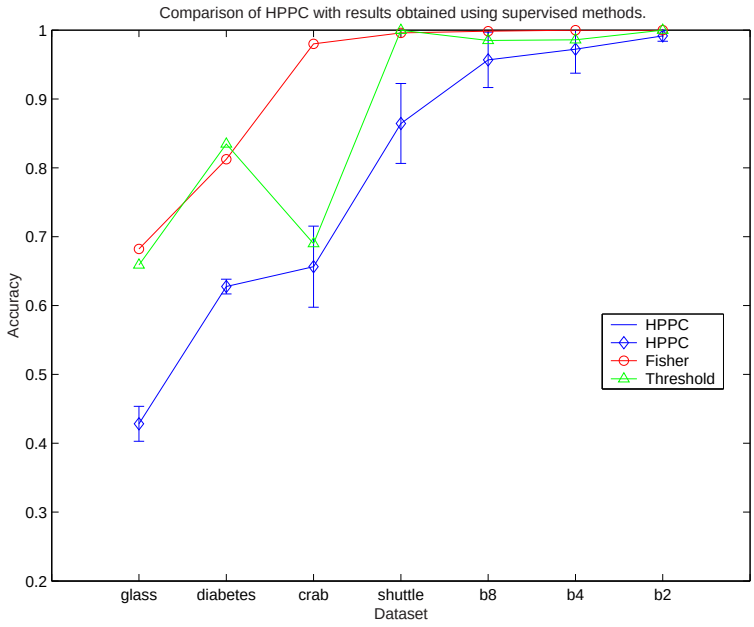

Figure 28: Comparison of HPPC with supervised methods.

data by appropriate pre-processing of the data. Various non-linear mappings can be applied to the data to create a new data set in a new feature space in which the data are linearly separable.

If the standard clustering algorithm fails to find significant structure in the data one can change the representation of the data by replacing the feature vector x by its image via a non-linear mapping $\varphi = (\varphi_1, \ldots, \varphi_k)$:

$$x \to \varphi(x) =< \varphi_1(x), \ldots, \varphi_k(x) > .$$

One of the most common approaches is to use a general quadratic mapping (where the quadratic terms are written in some order)

$$\varphi(x_1, \ldots, x_d) = (x_1, \ldots, x_d, x_1x_1, x_1x_2, \ldots, x_dx_{d-1}, x_dx_d)$$

so

$$x_\varphi = \varphi(x)$$

is a vector consisting of components of x and all their pair-wise products written in some order. Now one can apply the standard algorithm to the vectors x_φ and search for structure in the new feature space.

6.2 Extensions to Large Datasets

If we have a large number of observations to consider the computation can made to run faster by working on a sub-sample. Given a large dataset we can choose some random sample of size N/k, for some k and cluster on the sample. The remaining data can be used for evaluation of the obtained clustering scheme. The test data is passed through the resulting tree and the leaf nodes are inspected. Should a leaf not receive a significant number observations, or meets a stopping condition, then it can be pruned.

References

[1] Dimitris Achlioptas, *Database-friendly random projections*, Symposium on Principles of Database Systems, 2001.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*, In Proceedings of 1998 ACM-SIGMOD International Conference on Management of Data (Seattle, Washington), June 1998, pp. 94–105.

[3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*, Proc. 20th Int. Conf. Very Large Data Bases, VLDB (Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, eds.), Morgan Kaufmann, 12–15 1994, pp. 487–499.

[4] M. Ankerst, M. M. Breunig, H. P. Kriegel, and J. Sander, *OPTICS: ordering points to identify the clustering structure*, Proceedings of the ACM SIGMOD International Conference on Management of Data (Philadelphia, PA), 1999, pp. 49–60.

[5] Pavel Berkhin, *Survey of clustering data mining techniques*, Tech. report, Accrue Software, San Jose, CA, 2002.

[6] Ella Bingham and Heikki Mannila, *Random projection in dimensionality reduction: applications to image and text data*, Knowledge Discovery and Data Mining, 2001, pp. 245–250.

[7] R. J. Bolton and W. J. Krzanowski, *Projection pursuit clustering for exploratory data analysis*, Journal of Computational and Graphical Statistics **12** (2003), no. 1.

[8] P.S. Yu C. C. Aggarwal, *Redefining Clustering for High-Dimensional Applications*, IEEE Transactions on Knowledge and Data Engineering **14** (2002), no. 2.

[9] N. A. Campbell and R. J. Mahon, *A multivariate study of variation in two species of rock crab of the genus Leptograpsus*, Australian Journal of Zoology **22** (1974).

[10] C. H. Cheng, A. W. Fu, and Y. Zhang, *Entropy-based Subspace Clustering for Mining Numerical Data*, Knowledge Discovery and Data Mining, 1999, pp. 84–93.

[11] S. Cho, R. M. Haralick, and S. Yi, *Improvement of Kittler and Illingworth's Minimum Error Thresholding*, Pattern Recognition **22** (1989), no. 5.

[12] D. Michie and D. J. Spiegelhalter and C. C. Taylor (ed.), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, London, U.K., 1994.

[13] S. Dasgupta, *Experiments with random projection*, Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (Stanford, CA), July 2000.

[14] A. P Dempster, N. M. Laird, and D. B. Rubin, *Maximum Likelihood from Incomplete Data Via the EM algorithm*, Journal of the Royal Statistical Society Series B (1977), no. 39.

[15] P. Diaconis and D. Freedman, *Asymptotics of graphical projection pursuit*, Annals of Statistics **12** (1984).

[16] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[17] R. Duda, P. Hart, and D. Stork, *Pattern Classication, 2nd edition*, Wiley, New York, 2000.

[18] D. Edwards, *Introduction to Graphical Modelling*, Second ed., Springer, 2000.

[19] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[20] J. Friedman, *Exploratory Projection Pursuit*, Journal of the American Statistical Association **82** (1987), no. 397.

[21] J. Friedman and W. Stuetzle, *Projection pursuit methods for data analysis*, Tech. Report SLAC PUB-2768, Stanford Linear Accelerator Center, June 1981.

[22] J. H. Friedman and J. W. Tukey, *A projection pursuit algorithm for exploratory data analysis*, IEEE Transactions on Computers **C-23** (1974), no. 9.

[23] S. Guha, R. Rastogi, and K. Shim, *CURE: an efficient clustering algorithm for large databases*, 1998, pp. 73–84.

[24] _____, *ROCK: A Robust Clustering Algorithm for Categorical Attributes*, Information Systems **25** (2000), no. 5, 345–366.

[25] J. A. Hartigan, *Clustering Algorithms*, Wiley, New York, 1975.

[26] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, New Jersey, 1999.

[27] A. Hinneburg and D. A. Keim, *An Efficient Approach to Clustering in Large Multimedia Databases with Noise*, Knowledge Discovery and Data Mining, 1998, pp. 58–65.

[28] A. Hinneburg and Daniel A. Keim, *Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering*, The VLDB Journal, 1999, pp. 506–517.

[29] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, *Algorithms for association rule mining – a general survey and comparison*, SIGKDD Explorations **2** (2000), no. 1, 58–64.

[30] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[31] P. J. Huber, *Projection pursuit*, Annals of Statistics **13** (1985), no. 2.

[32] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, ACM Computing Surveys **31** (1999), no. 3, 264–323.

[33] J. Kittler and J. Illingworth, *Minimum Error Thresholding*, Pattern Recognition **19** (1986), no. 1.

[34] S. Lauritzen, *Graphical Models*, Oxford University Press, 1996.

[35] F. Meyer, *Topographic distance and watershed lines*, Signal Processing **38** (1994).

[36] R. Ng and J. Han, *Efficient and effective clustering methods for spatial data mining*, Proceedings of the 20th International Conference on Very Large Databases (VLDB), September 1994.

[37] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, 1988.

[38] R. A. Redner and H.F. Walker, *Mixture Densities, Maximum Likelihood and the EM Algorithm*, SIAM Review **26** (1984), no. 2.

[39] C. R. Reeves, *Genetic algorithms for operations research*, INFORMS Journal on Computing **9** (1997), no. 3.

[40] S. L. Stoev, *RafSi - a Fast Watershed Algorithm Based on Rainfalling Simulation*, Proceedings of 8-th International Conference on Computer Graphics, Visualization, and Interactive Digital Media (WSCG'2000), 2000.

[41] G. Sheikholeslami, S. Chatterjee, and A. Zhang, *WaveCluster: A multi-resolution clustering approach for very large spatial databases*, Proc. 24th Int. Conf. Very Large Data Bases, VLDB, 24–27 1998, pp. 428–439.

[42] S. Theodoridis and K. Koutroumbas, *Pattern recogniton*, Academic Press, San Diego, 1999.

[43] V. N. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, New York, 1998.

[44] ———, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 2000.

[45] W. Wang, J. Yang, and R. R. Muntz, *STING: A statistical information grid approach to spatial data mining*, Twenty-Third International Conference on Very Large Data Bases (Athens, Greece) (Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, eds.), Morgan Kaufmann, 1997, pp. 186–195.

[46] X. Xu, M. Ester, H. P. Kriegel, and J. Sander, *A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases*, ICDE, 1998, pp. 324–331.

[47] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: an efficient data clustering method for very large databases*, ACM SIGMOD International Conference on Management of Data (Montreal, Canada), June 1996, pp. 103–114.