Chapter 2

Clustering

Abstract

Clustering is a widely used technique to partition data in homogeneous groups. It finds applications in many fields from information retrieval to bio-informatics. The main goal of clustering algorithms is to discover the hidden structure of data and group them without any a-priori knowledge of the data domain. Clustering is often used for exploratory tasks.

The intuition behind partitioning data is that if two objects are closely related and the former is also related to a third object, then more likely also the latter has a similar relation. This idea is known as the cluster hypothesis*.*

In the first part of this chapter we survey the principal strategies for clustering, the main clustering objective functions and related algorithms, the main definitions for similarity and the clustering validation techniques. We conclude the chapter giving some results about how we improved the Furthest-point-first algorithm in terms of speed and quality.

2.1 Introduction to clustering

Clustering is a technique to split a set of objects in groups such that *similar* objects are grouped together, while objects that are not similar fall in different clusters. The choice of the notion of similarity (or distance) among objects that are needed to be clustered is of crucial importance for the final result.

Clustering algorithms have no a-priori knowledge about the data domain, its hidden structure and also the number of hidden classes in which data are divided is unknown. For this characteristic, clustering is often referred as un-supervised learning in contrast to classification (or supervised learning) in which the number of classes is known and for each class a certain number of examples are given.

The independence of clustering algorithms from the data domain is at the same time the secret of its success and its main drawback. In fact since clustering does not need any a-priori knowledge of the data domain, it can be applied to a wide range of problems in different application areas. In contrast, general purpose procedures do not allow to apply (even trivial) problem dependent optimizations and consequently they typically perform worse then ad-hoc solutions.

2.1.1 Metric space for clustering

The choice of how to represent the data objects one wants to cluster, together with the choice of the clustering strategy, is critical for the clustering result. The representation schema depends from the type of data we are working on. In some fields de-facto standards are widely used.

In *text retrieval* the vector space model is the most commonly used. Documents in this model are represented as vectors of weighted terms called *bag of words*. For weighting, many approaches are used: binary schema (in which the weight of a term is 0 if the term does not appear in the document, 1 otherwise), the tf-idf scoring and so on. In *video retrieval* frames are represented as vectors in the HSV color space. In *bio-informatics*, DNA microarrays are matrices in which each gene is stored in a row and each column corresponds to a probe.

In all the above cited cases, a set of objects $O = \{o_1, \ldots, o_n\}$ are represented with m-dimensional vectors which are stored in a matrix M of n rows and m columns, where n is the number of objects in the corpus while m is the number of features of the objects. These vector spaces endowed with a distance function define a metric space. The most widely used distance functions are:

- **Cosine similarity:** it is defined as the cosine of the angle between o_a and o_b. More formally
$$s(o_a, o_b) = \frac{o_a \cdot o_b}{\|o_a\| \cdot \|o_b\|}$$
A distance can be easily derived from cosine similarity by:
$$d(o_a, o_b) = \sqrt{1 - s^2(o_a, o_b)}$$

The most important property of cosine similarity is that it does not depend on the length of the vectors: $s(o_a, o_b) = s(\alpha o_a, o_b)$ for $\alpha > 0$. This property makes the cosine similarity widely used in text information retrieval.

- **Jaccard coefficient:** in its basic form it is defined as:

$$J(o_a, o_b) = \frac{\#(o_a \cap o_b)}{\#(o_a \cup o_b)}$$

where $o_a \cap o_b$ is the set of features in common between o_a and o_b and $o_a \cup o_b$ is the total set of features (this approach assumes binary features). Many variants of the Jaccard coefficient were proposed in the literature. The most interesting is the *Generalized Jaccard Coefficient* (GJC) that takes into account also the weight of each term. It is defined as

$$GJC(o_a, o_b) = \frac{\min_{i=1}^{m}(o_{a,i}, o_{b,i})}{\max_{i=1}^{m}(o_{a,i}, o_{b,i})}$$

GJC is proven to be a metric [Charikar, 2002]. The Generalized Jaccard Coefficient defines a very flexible distance that works well with both text and video data.

- **Minkowski distance:** it is defined as:

$$L_p(o_a, o_b) = (\sum_{i=1}^{m} |o_{a,i} - o_{b,i}|^p)^{1/p}$$

It is the standard family of distances for geometrical problems. Varying the value of the parameter p, we obtain different well known distance functions. When $p = 1$ the Minkowski distance reduces to the Manhattan distance. For $p = 2$ we have the well known Euclidean distance

$$L_2(o_a, o_b) = \sqrt{\sum_{i=1}^{m}(o_{a,i} - o_{b,i})^2}$$

When $p = \infty$ this distance becomes the infinity norm defined as:

$$L_\infty(o_a, o_b) = \max_{i=1}^{m}(o_{a,i}, o_{b,i})$$

- **Pearson correlation:** it is defined as follows:

$$P(o_a, o_b) = \frac{\sum_{k=1}^{m}(o_{a,k} - \mu_a)(o_{b,k} - \mu_b)}{\sqrt{\sum_{k=1}^{m}(o_{a,k} - \mu_a)^2} \cdot \sqrt{\sum_{k=1}^{m}(o_{b,k} - \mu_b)^2}},$$

where μ_a and μ_b are the means of o_a and o_b, respectively. Pearson coefficient is a measure of similarity. In particular it computes the similarity of the shapes between the two profiles of the vectors (it is not robust against

outliers - potentially leading to false positive, assigning high similarity to a pair of dissimilar patterns -, it is sensible to the shape but not to the magnitude). To compute a distance, we define $d_{o_a,o_b} = 1 - P(o_a, o_b)$. Since $-1 \leq P(o_a, b_b) \leq 1$, for all o_a, o_b, we have $0 \leq d_{o_a,o_b} \leq 2$. This distance is not a metric since both the triangular inequality and small self-distance ($d_{o_a,o_b} = 0$) do not hold. However, the square root of $1 - P(o_a, o_b)$ is proportional to the Euclidean distance between o_a and o_b [Clarkson, 2006], hence only the small self-distance condition fails for this variant, and metric space methods can be used.

2.2 Clustering strategy

Clustering algorithms can be classified according with many different characteristics. One of the most important is the strategy used by the algorithm to partition the space:

- **Partitional clustering**: given a set $O = \{o_1, \ldots, o_n\}$ of n data objects, the goal is to create a partition $C = \{c_1, \ldots, c_k\}$ such that:
  - $\forall i \in [1, k] \qquad c_i \neq \emptyset$
  - $\bigcup_{i=1}^{k} c_i = O$
  - $\forall i, j \in [1, k] \wedge i \neq j \qquad c_i \cap c_j = \emptyset$
- **Hierarchical clustering**: given a set $O = \{o_1, \ldots, o_n\}$ of n data objects, the goal is to build a tree-like structure (called *dendrogram*) $H = \{h_1, \ldots, h_q\}$ with $q \leq n$, such that: given two clusters $c_i \in h_m$ and $c_j \in h_l$ with h_l ancestor of h_m, one of the following conditions hold: $c_i \subset c_j$ or $c_i \cap c_j = \emptyset$ for all $i, j \neq i, m, l \in [1, q]$.

Partitional clustering is said **hard** if a data object is assigned uniquely to one cluster, **soft** or **fuzzy** when a data object belongs to each cluster with a degree of membership.

2.2.1 Partitional clustering

When the data representation and the distance function d have been chosen, partitional clustering reduces to a problem of minimization of a given target function. The most widely used are:

- K**-center** minimizes the maximum cluster radius

$$\min \max_{j} \max_{x \in c_j} d(x, C_j)$$

- K-**medians** minimizes the sum of all the point-center distances

$$\min \sum_j \sum_{x \in c_j} d(x, \mu_j)$$

- K-**means** minimizes the sum of squares of inter-cluster point-center distances

$$\min \sum_j \sum_{x \in c_j} (d(x, \mu_j))^2$$

where $C = \{c_1, \ldots, c_k\}$ are k clusters such that C_j is the center of the j-th cluster and μ_j is its centroid.

For all these functions it is known that finding the global minimum is NP-hard. Thus, heuristics are always employed to find a local minimum.

2.2.1.1 FPF algorithm for the k-center problem

As said in 2.2.1 one of the possible goal for partitional clustering is the minimization of the largest cluster diameter solving the k-center problem. More formally the problem is defined as:

Definition 1. ***The k-centers problem:*** Given a set O of points in a metric space endowed with a metric distance function d, and given a desired number k of resulting clusters, partition O into non-overlapping clusters $C_1, \ldots, C_k$ and determine their "centers" $c_1, \ldots, c_k \in O$ so that $\max_j \max_{x \in C_j} d(x, c_j)$ (i.e. the radius of the widest cluster) is minimized.

In [Feder and Greene, 1988] it was shown that the k-center problem is NP-hard unless $P = NP$. In [Gonzalez, 1985; Hochbaum and Shmoys, 1985] two-approximated algorithms are given.

We first describe the original *Furthest Point First* (FPF) algorithm proposed by Gonzalez [Gonzalez, 1985] that represents the basic algorithm we adopted in this thesis. Then, in section 2.4 we will give details about the improvements we made to reduce the running time and obtain a better clustering quality.

Basic algorithm Given a set O of n points, FPF increasingly computes the set of centers $c_1 \subset \ldots \subset c_k \subseteq O$, where C_k is the solution to the k-center problem and $C_1 = \{c_1\}$ is the starting set, built by randomly choosing c_1 in O. At a generic iteration $1 < i \leq k$, the algorithm knows the set of centers C_{i-1} (computed at the previous iteration) and a mapping μ that associates, to each point $p \in O$, its closest center $\mu(p) \in C_{i-1}$. Iteration i consists of the following two steps:

1. Find the point $p \in O$ for which the distance to its closest center, $d(p, \mu(p))$, is maximum; make p a new center c_i and let $C_i = C_{i-1} \cup \{c_i\}$.

2. Compute the distance of c_i to all points in $O \setminus C_{i-1}$ to update the mapping μ of points to their closest center.

After k iterations, the set of centers $C_k = \{c_1, \ldots, c_k\}$ and the mapping μ define the clustering. Cluster $\mathcal{C}_i$ is the set of points $\{p \in O \setminus C_k$ such that $\mu(p) = c_i\}$, for $i \in [1, k]$. Each iteration can be done in time $O(n)$, hence the overall cost of the algorithm is $O(kn)$. Experiments have shown that the random choice of c_1 to initialize C_1 does not affect neither the effectiveness nor the efficiency of the algorithm.

FPF:
Data: Let O be the input set, k the number of clusters
Result: $\mathcal{C}$, k-partition of O
$C = x$ such that x is an arbitrary element of O;
for $i = 0$; $i < k$; $i++$ **do**
  Pick the element x of $O \setminus C$ furthest from the closest element in C;
  $C_i = \mathcal{C}_i = x$;
end
forall $x \in O \setminus C$ **do**
  Let i such that $d(c_i, x) < d(c_j, x), \forall j \neq i$ $\mathcal{C}_i$.append (x);
end
Algorithm 1: The furthest point first algorithm for the k-center problem.

2.2.1.2 K-means

The k-means algorithm [Lloyd, 1957] is probably the most widely used in the literature. Its success comes from the fact it is simple to implement, enough fast for relatively small datasets and it achieves a good quality. The k-means algorithm can be seen as an iterative cluster quality booster.

It takes as input a rough k-clustering (or, more precisely, k candidate centroids) and produces as output another k-clustering, hopefully of better quality.

K-means, as objective function, attempts to minimize the sum of the squares of the inter-cluster point-to-center distances. More precisely, this corresponds to partition, at every iteration, the input points into non-overlapping clusters $C_1, \ldots, C_k$ and determining their centroids $\mu_1, \ldots, \mu_k$ so that

$$\sum_{j=1}^{k} \sum_{x \in C_j} (d(x, \mu_j))^2$$

is minimized.

It has been shown [Selim and Ismail, 1984] that by using the sum of squared Euclidean distances as objective function, the procedure converges to a local minimum for the objective function within a finite number of iterations.

The main building blocks of k-means are:

- **the generation of the initial k candidate centroids**: In this phase an initial choice of candidate centroids must be done. This choice in critic because both the final clustering quality and the number of iterations needed to converge are strongly related to this choice. In the next section we will survey the most important initialization strategies. A more complete survey and comparison can be found in [Bradley and Fayyad, 1998; Peña *et al.*, 1999].
- **the main iteration loop**: In the main iteration loop, given a set of k centroids, each input point is associated to its closest centroid, and the collection of points associated to a centroid is considered as a cluster. For each cluster, a new centroid that is a (weighted) linear combination of the points belonging to the cluster is recomputed, and a new iteration starts[1].
- **the termination condition**: Several termination conditions are possible; e.g. the loop can be terminated after a predetermined number of iterations, or when the variation that the centroids have undergone in the last iteration is below a predetermined threshold.

The use of k-means has the advantage that the clustering quality is steadily enough good in different settings and with different data. This makes k-means the most used clustering algorithm. Due its importance, there is a vast literature that discusses its shortcomings and possible improvements to the basic framework.

A lot of efforts was spent to reduce the k-means computational time that depends on the size of the dataset, the number of desired clusters and the number of iterations to reach convergence. Some methods attempt to use clever data structures to cache distances [Elkan, 2003; Smellie, 2004], others exploit the triangular inequality for avoiding distance computations [Phillips, 2002]. For small datasets or when only few iterations are enough to achieve the desired output quality, the performance of k-means is acceptable, but for nowadays needs clustering time has become a shortcoming (i.e. in chapter 6 we will see that for 100 thousand of documents and 1000 clusters, k-means running time is of the order of a week).

Another well-known shortcoming is that some clusters may become empty during the computation. To overcome this problem, the "ISODATA" [Tou and Gonzalez, 1977] technique was proposed. Essentially when a cluster becomes empty, ISODATA splits one of the "largest" clusters so as to keep the number of clusters unchanged.

Initialize k-means Essentially k-means accepts as input an initial clustering that can be made with any clustering algorithm. It is well-known that the quality of the initialization (i.e. the choice of the initial k centroids) has a deep impact on the resulting accuracy. Several methods for initializing k-means are compared in [Bradley and Fayyad, 1998; Peña *et al.*, 1999]. The three most common initializations are:

[1] Note that k-means is defined on vector spaces but not in general on metric spaces, since in metric spaces linear combinations of points are not points themselves.

RC The simplest (and widely used) initialization for k-means is the one in which the initial centroids are Randomly Chosen among the input points and the remaining points are assigned to the closest centroid. The resulting clustering is often referred as *random clustering*.

RP In the Random Perturbation, for each dimension d_j of the space, the distribution of the projections on d_j of the data points is computed, along with its mean μ_j and its standard deviation σ_j; the k initial centroids are obtained through k perturbations, driven by the μ_j's and σ_j's, of the centroid of all data points [Peña *et al.*, 1999].

MQ MacQueen's [MacQueen, 1967] proposed a variant of k-means: the initial centroids are randomly chosen among the input points, and the remaining points are assigned one at a time to the nearest centroid, and each such assignment causes the immediate recomputation of the centroid involved. Then k-means is initialized with the resulting clustering. Since it was experimentally shown that this initialization achieves generally a good quality in considerably less time than k-means, this initialization is often used in place of the standard k-means and it is often referred as *one-pass* k-means.

2.2.1.3 PAM: partition around medoids

Partition around medoids [Kaufman and Rousseeuw, 1990] was introduced by Kaufman and Rousseeuw. PAM introduces the concept of *medoid*. A medoid is a point of the input, it means that PAM is particularly suitable in all those cases in which the concept of centroid in not well defined. Moreover, in many cases, the more the number of objects increase, the less centroids tend to be representative; instead medoids are not affected by this problem.

PAM builds a k-clustering and it can be described as follows [Ng and Han, 1994]:

1. Select a set of k random input objects $O = \{o_1, \ldots o_k\}$,
2. for each input object $x \notin O$ compute the cost function $TC(x, o_i)$,
3. select the pair of objects x and o_i that minimize TC,
4. if $TC(x, o_i) < 0$ replace o_i with x and restart from step 2.

The final clustering is obtained using the objects o_i as cluster centers and assigning the input points to the cluster with the nearest center.

PAM is computationally expensive, in fact there are $(n - k)$ different pairs of object for each of th k medoids. It means that for each iteration TC is computed $k(n - k)$ times. Due to its computational cost, many variations and performance improvements were proposed in the literature [Zhang and Couloigner, 2005].

2.2.1.4 SOM: self organizing maps

Self organizing Maps [Kohonen, 2001] were introduced by Teuvo Kohonen as subtype of artificial neural networks used to produce low dimensional representation of the training samples while preserving the topological properties of the input space.

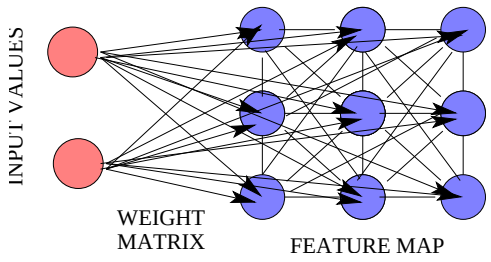

Figure 2.1. A simple 3×3 self organizing map.

A self-organizing map is a single layer feed-forward network, that is a network without direct cyclic paths. Neurons are arranged in a low dimensional grid (typically two-dimensional or tridimensional). Each neuron has associated a vector of weights $w_i = \{w_{i,1}, \ldots, w_{i,m}\}$ of the same size of input vectors. There are two main ways to initialize the weights vectors:

- using small random values,
- using a random perturbation from the subspace spanned by the two largest principal component eigenvectors. This initialization was shown to speed up the training phase of the SOM because they are already a good approximation of the SOM weights.

Self-organizing maps work in two phases:

- **training**: the training phase can be seen as the process in which the self-organizing map attempts to adapt the weight vectors of its nodes to the training data. For this purpose a large number of examples must be fed in input. If a training set is not available the input data are often used to train the network. The training algorithm is based on a *competitive learning* approach: when a new sample $x(t)$ is presented to the network it is compared with all the weights vectors and the neuron with closest weight vector (called Best Matching Unit) is selected (i. e. the neuron i such that $\min_i d(x(t), w_i)$). The weight vector of the BMU and its neighbors, are modified according with the sample. More formally let i the BMU and e a generic neuron of the SOM. Let $h(e, i)$ be a proximity function between the two neurons and $w_e(t)$ be the value of w_e at the epoch t. The weight vector of the generic neuron e is updated according with the following:

$$w_e(t+1) = w_e(t) + \alpha(t) * h(e,i) * (x(t) - w_e(t))$$

where $\alpha(t)$ is a monotonically decreasing learning coefficient.

- **mapping**: in this phase the input vectors are simply assigned to the closest neuron. Borrowing the terminology of k-means the nodes of the network in this phase play the same role of centroids. It is interesting to note that the number of clusters in output depends on the number of neurons in the network. This means that the structure of the SOM drastically influences the clustering results.

Learning:
Data: the SOM $M = \{m_j \forall j \leq TOT_{Nodes}\}$, $\alpha(t)$, $h(-,-)$,
$X = \{x(t) \forall t \leq TOT_{Sample}\}$
Result: the trained SOM M
forall $m \in M$ **do**
  initialize (m);
end
for $t = 1$; $t \leq TOT_{Sample}$; $t++$ **do**
  $i = \arg\min_j d(x(t), m_j)$;
  forall $m_e \in M$ **do**
    $m_e(t+1) = m_e(t) + \alpha(t) * h(e,i) * (x(t) - m_e(t))$
  end
end
return M;

Mapping:
Data: the SOM $M = \{m_j \forall j \leq TOT_{Nodes}\}$, $X = \{x(t) \forall t \leq TOT_{Sample}\}$
Result: The clustering C
for $i = 1$; $t \leq TOT_{Nodes}$; $t++$ **do**
  $C_i = \emptyset$;
end
for $t = 1$; $t \leq TOT_{Sample}$; $t++$ **do**
  $i = \arg\min_j d(x(t), m_j)$;
  $C_i = C_i \cup x(t)$
end
return C;

Algorithm 2: The self-organizing map algorithm.

In the case in which the size of input vectors is higher than the number of nodes in the output grid, SOM becomes a powerful tool to make dimensionality reduction [Tan *et al.*, 2005] (Feature selection).

2.2.2 Hierarchical clustering

The main difference between partitional clustering and hierarchical clustering consists in the fact the latter does not limit only in grouping the data objects in a flat partition, but it also arranges the data into a tree like structure. This structure is known as *dendrogram*. Each data object is assigned to a leaf of the tree, while internal nodes represent groups of objects such that for each pair of elements in such group, their distance is within a certain threshold. The root of the dendrogram contains all the objects. A flat clustering can be easily obtained by cutting the dendrogram at a certain level.

An important characteristic of hierarchical clustering is that it requires the computation of the *proximity matrix* that is the squared matrix of the distances between all the pairs of points in the data set. This makes the time and space complexity of this family of algorithms at least quadratic in the number of data objects. In recent years, a lot of effort was done to improve the hierarchical clustering algorithms performances and make them suitable for large scale datasets. Typical example are: BIRCH [Zhang *et al.*, 1996] and CUTE [Guha *et al.*, 1998].

The two main strategies for hierarchical clustering are:

- **Divisive**: in this case the dendrogram is built from the root to the leafs. Initially all the n objects are in the same cluster. A series of split operations is made until all clusters contains just a single element. The splitting operation is made by computing all the distances between the pairs of objects in the same cluster and selecting the two diametral points as seeds, then all the points in the group are assigned to the closest seed.

- **Agglomerative**: the dendrogram is built from the leaves to the root. At the beginning each object is inserted in a cluster (that represent a leaf of the dendrogram), than a series of merge operations is made until all the points belong to the same cluster. Since the data objects are n and each merge operation reduces the number of objects of one unit, $n - 1$ merge operations are needed. It is important to note that the operations of merge are made between the two closest entities (either objects or clusters). A notion of cluster-cluster distance and cluster-object distance must to be defined.

2.2.2.1 Divisive clustering

As mentioned in section 2.2.2, hierarchical divisive clustering algorithms start with considering the whole input set as a single cluster that is the root of the dendrogram. Before to start the procedure, a threshold distance must be chosen. Once this is done, hierarchical divisive clustering proceeds as follows:

- the proximity matrix M is calculated and for each cluster and the furthest pair of objects is selected,

- if the cluster satisfies the algorithm splitting criterion, (i.e. the distance between the diametral pair is higher than a certain threshold) the cluster is divided into two clusters by using the pair selected in the previous step as seeds,
- when no more clusters must to be splitted, the algorithm stops.

One of the most important issues in divisive hierarchical clustering is the choice of the splitting criterion [Savaresi *et al.*, 2002]. The following strategies are typically used [Karypis *et al.*, 1999]:

- each cluster is recursively splitted until each subcluster contains exactly one element. In this case a complete tree is obtained. The main advantage of this method is that a complete tree is obtained. The main disadvantage is that the final clustering quality is not taken into account by this schema.
- The cluster with the largest number of elements is splitted. Using this approach a balanced tree is obtained.
- The cluster with the highest variance with respect to its "centroid" is splitted. This is a widely used method to choose the cluster to split because it is related to the distribution of the elements inside the cluster.

2.2.2.2 Agglomerative clustering

As mentioned in section 2.2.2, hierarchical agglomerative clustering attempts to cluster a set of n objects providing also a tree like structure built from the leafs to the root.

In the merging operation the two closest entities of the dendrogram (leafs or internal nodes) are joined into a single entity. Considering leafs as clusters containing only an element, the notion of inter-cluster distance must be defined. There are many different possibilities for this choice. The most common ones are based on a linkage criterion (i. e. the distance between two clusters is the distance between two points that are associated to them in such a way). Given two clusters C_i and C_j we have:

- **Single linkage**: $d(C_i, C_j) = \min_{p \in C_i q \in C_j} d(p, q)$ is the distance between the closest pair of objects from different clusters. This method has the drawback that it tends to force clusters together due to a single pair of close objects regardless of the positions of the other elements in the clusters. This is known as *chaining phenomenon*.

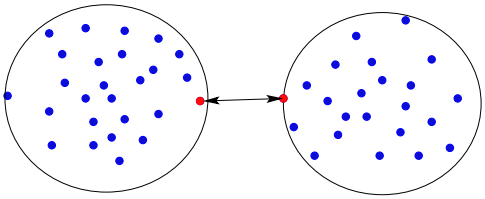

Figure 2.2. Single linkage criterion.

- **Complete linkage**: $d(C_i, C_j) = \max_{p \in C_i q \in C_j} d(p, q)$ is the distance between the farthest pair of objects from different clusters. This method tends to make more compact clusters, but it is not tolerant to noisy data.

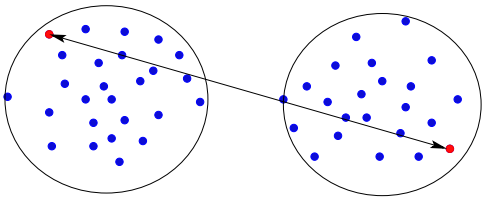

Figure 2.3. Complete linkage criterion.

- **Average linkage**: $d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q)$ is the mean of the distance among all the pairs of objects coming from different clusters. This method is more robust with respect to the previous ones, in fact the impact of outliers is minimized by the mean and the chaining phenomenon is typically not observed.

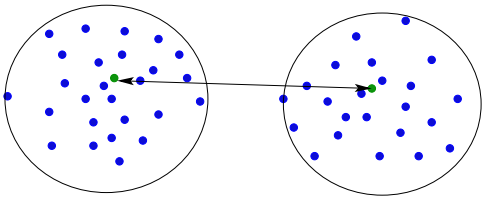

Figure 2.4. Average linkage criterion.

Single linkage and complete linkage can be generalized as suggested by Lance and Williams in [Lance and Williams, 1967] using the following formula:

$$d(C_l, (C_i, C_j)) = \alpha_i d(C_l, C_i) + \alpha_j d(C_l, C_j) + \beta d(C_i, C_j) + \\ + \gamma \mid d(C_l, C_i) - d(C_l, C_j) \mid \qquad (2.1)$$

where d is the distance between two entities, (C_i, C_j) is the cluster coming from the union of C_i and C_j and the four parameters α_i, α_j, β, γ, depend on the specific strategy used. Note that when $\alpha_i = \alpha_j = 1/2$, $\beta = 0$ and $\gamma = -1/2$, formula (2.1) becomes

$$d(C_l, (C_i, C_j)) = \min(d(C_l, C_i), d(C_l, C_j))$$

that is the single linkage formula. Instead, the choice of $\alpha_i = \alpha_j = \gamma = 1/2$ and $\beta = 0$ makes (2.1) be

$$d(C_l, (C_i, C_j)) = \max(d(C_l, C_i), d(C_l, C_j))$$

that is the formula of complete linkage.

The hierarchical agglomerative clustering algorithm can be summarized by the following procedure:

1. Initialize the proximity matrix M such that $M_{i,j}$ is the distance between the i-th and the j-th entity
2. Find i and j such that $i \neq j$ and $\forall h, k$: $h \neq k$, $M_{i,j} \leq M_{h,k}$
3. Join C_i and C_j and update M accordingly
4. Repeat from step 2 until all the clusters are merged

2.2.3 The choice of the number k of clusters

All the algorithms we considered in this chapter are not able to discover the number of groups in which the hidden structure of the input set should be divided. For all the described algorithms, the number of clusters is part of the input. In some cases, like SOMs, the choice of k is subjugated to the algorithm constraints. It is clear that the final clustering quality is strongly dependent from this choice. In fact, a too large number of clusters can have the effect to complicate the analysis of results, while too few clusters can lead to information loss or inaccurate modeling.

Many different techniques were proposed in the literature to find the "right" value for k; the most common approaches are based on: the construction of indices that take into account properties like homogeneity, separation and silhouette (a survey of some of them and an evaluation of their performances can be found in [Milligan and Cooper, 1985]); the optimization of some probabilistic functions and heuristics.

It is also important to note that all those methods, based on the computation of indices or on the optimization of probabilistic functions, must be applied to many choices of k. This makes desirable to have clustering algorithms able to make clusters incrementally without the need to know k in advance and to backtrack if needed. To this aim divisive hierarchical clustering and FPF are more flexible with respect to k-means and SOMs.

2.2.3.1 Stability based techniques

We describe here in more details the stability based technique based on the prediction strength method (developed by Tibshirani et al [Tibshirani *et al.*, 2005]) to estimate the number k of clusters. Then we describe an efficient variant of this schema applied to the FPF algorithm as we adopted in [Geraci *et al.*, 2007]. This approach can be used efficiently for all the incremental cluster algorithms such as the divisive hierarchical clustering.

To obtain the estimate of a good value of k, the method proceeds as follows. Given the set O of n objects, randomly choose a sample O_r of cardinality μ. Then, for increasing values of t ($t = 1, 2, \dots$) repeat the following steps:

1. using the clustering algorithm, cluster both $O_{ds} = O \setminus O_r$ and O_r into t clusters, obtaining the partitions $C_t(ds)$ and $C_t(r)$, respectively;

2. measure how well the t-clustering of O_r predicts co-memberships of mates in O_{ds} (i.e. count how many pairs of elements that are mates in $C_t(ds)$ are also mates according to the centers of $C_t(r)$).

Formally, the measure computed in step 2 is obtained as follows. Given t, clusterings $C_t(ds)$ and $C_t(r)$, and objects o_i and o_j belonging to O_{ds}, let $D[i, j] = 1$ if o_i and o_j are mates according to both $C_t(ds)$ and $C_t(r)$, otherwise $D[i, j] = 0$. Let $C_t(ds) = \{C_{t,1}(ds), \dots, C_{t,t}(ds)\}$, then the prediction strength $PS(t)$ of $C_t(ds)$ is defined as:

$$PS(t) = \min_{1 \le l \le t} \frac{1}{\#pairs \in C_{t,l}(ds)} \sum_{i,j \in C_{t,l}(ds), i<j} D[i, j] \qquad (2.2)$$

where the number of pairs in $C_{t,l}(ds)$ is given by its binomial coefficient over 2. In other words, $PS(t)$ is the minimum fraction of pairs, among all clusters in $C_t(ds)$, that are mates according to both clusterings, hence $PS(t)$ is a worst case measure. The above outlined procedure terminates at the largest value of t such that $PS(t)$ is above a given threshold, setting k equal to such t.

We now describe the modified version of the stability based technique we applied to FPF in [Geraci *et al.*, 2007]. Note that this modified procedure depends only on the ability of the clustering algorithm to create clusters one by one. We first run the clustering algorithm on O_r up to $t = \mu$, storing all the computed centers $c_1, \dots, c_\mu$. In a certain sense, the order in which centers are selected by FPF, is used as a sort of ranking of the points of O_r. In the case of using FPF this step costs $O(\mu|O_r|) = O(\mu^2)$.

We then cluster the input set O_{ds}. Suppose at step t we have computed the clusters $C_{t,1}(ds), \dots, C_{t,t}(ds)$ and suppose, for each $o \in O_{ds}$, we keep the index $i(o, t)$ of its closest center among $c_1, \dots, c_t$. Such index can be updated in constant time by comparing $d(o, c_{i(o,t-1)})$ with $d(o, c_t)$, i.e., the distance of o from the "current" center and that to the new center c_t. Now, for each $C_{t,l}(ds)$, $l \in [1, \dots, t]$ we

can easily count in time $O(|C_{t,l}(ds)|)$ the number of elements that are closest to the same center c_j, $j \in [1, \ldots, t]$, and finally compute the summations in formula 2.2 in time $O(|O_{ds}|)$.

After the last iteration, we obtain the clustering of O by simply associating the points $c_1, \ldots, c_\mu$ to their closest centers in $C_k(ds)$. The overall cost of the modified procedure using FPF as clustering algorithm is $O(\mu^2 + k(n-\mu) + k\mu) = O(kn)$ for $\mu = O(n^{1/2})$. Note that, differently from the original technique, we stop this procedure at the first value of t such that $PS(t) < PS(t-1)$ and set $k = t - 1$. In [Geraci *et al.*, 2007] we have empirically demonstrated that this choice of the termination condition gives good results.

2.3 Clustering validation

Since the clustering task has an ambiguous definition, the assessment of the quality of results is also not well defined. There are two main philosophies for evaluating the clustering quality:

- **internal criterion**: is based on the evaluation of how the output clustering approximates a certain objective function,
- **external criterion**: is based on the comparison between the output clustering and a predefined handmade classification of the data called *ground truth*.

When a ground truth is available, it is usually preferable to use an external criterion to assess the clustering effectiveness, because it deals with real data while an internal criterion measures how well founded the clustering is according with such mathematical definition.

2.3.1 Internal measures

There is a wide number of indexes used to measure the overall quality of a clustering. Some of them (i.e. the mean squared error) are also used as goal functions for the clustering algorithms.

2.3.1.1 Homogeneity and separation

According with the intuition, the more a cluster contains homogeneous objects the more it is a good cluster. Nevertheless the more two clusters are well separated the more they are considered good clusters. Following the intuition, homogeneity and separation [Shamir and Sharan, 2002] attempt to measure how compact and well distanciated clusters are among them.

More formally given a set of objects $O = \{o_1, \ldots, o_n\}$, we denote with $S(o_i, o_j)$ the similarity of the objects o_i and o_j according to a given similarity function. We say that o_i and o_j are mates if they belong to the same cluster. We define:

Homogeneity of a clustering: the average similarity between mates. Let $\mathcal{M}$ be the number of mate pairs:

$$H_{ave} = \frac{1}{\mathcal{M}} \sum_{o_i,o_j \ mates, i<j} S(o_i, o_j)$$

Separation of a clustering: the average similarity between non-mates. As $\mathcal{M}$ is the number of mate pairs, the number of non-mates pairs is given by $n(n-1)/2 - \mathcal{M}$.

$$S_{ave} = \frac{2}{n(n-1) - 2\mathcal{M}} \sum_{o_i,o_j \ non-mates, i<j} S(o_i, o_j)$$

Observe that the higher homogeneity is, the better the clustering is. Analogously, the lower separation is, the better the clustering is.

Alternative definition can be given using distances instead of similarities. In this case a better solution is given with a higher separation and a lower homogeneity.

Finally, homogeneity and separation can be approximated so that they can be calculated in linear time with the number n of objects (instead of quadratic). Given a clustering $C = \{C_1, \ldots, C_k\}$, let $cr(t)$ be the center (or centroid) of cluster C_t:

$$H_{approx} = \frac{1}{n} \sum_{t=1}^{k} \sum_{o_i \in C_t} S(o_i, cr(t)),$$

$$S_{approx} = \frac{1}{\sum_{t<z} |C_t||C_z|} \sum_{t<z} |C_t||C_z| S(cr(t), cr(z)).$$

Again, these measures can be expressed in terms of distances instead of similarities.

These two measures are inherently conflicting, because typically an improvement on one will correspond to a worsening of the other.

2.3.1.2 Average silhouette

Another measure that is worth calculate for a given clustering is the *average silhouette* [Rousseeuw, 1987]: for each element we compute a quantity, called silhouette, that gives an indication of how well the element fits into the cluster it is assigned to. The silhouette is based on homogeneity and separation; in particular we compute the homogeneity of the element with the elements in its cluster and the separation of the element with the closest cluster (among the others). In this way we can see if the element is well placed or if it is better placed in another cluster. The silhouette of object o_i that belongs to cluster $c \in \mathcal{C}$ is given by:

$$sil(o_i) = \frac{b_i - a_i}{\max\{a_i, b_i\}},$$

where a_i is the average distance of o_i to the elements in its cluster, while b_i is the average distance of o_i to the elements of the closest cluster. In formulas:

$$\begin{aligned} a_i &= \frac{1}{|c|} \sum_{o_j \in c} d(o_i, o_j) \\ b_i &= \min_{c' \in \mathcal{C}, c' \neq c} \left\{ \frac{1}{|c'|} \sum_{o_j \in c'} d(o_i, o_j) \right\}. \end{aligned}$$

(The values a_i and b_i can be approximated using the centers (or centroid) of clusters, in the same way as for homogeneity and separation).

Observe that for each element o_i we have $-1 < sil(o_i) < 1$ and that whenever o_i fits in its cluster, then $b_i > a_i$ and $sil(o_i) > 0$, while if o_i fits better in another cluster, then we have $b_i < a_i$ and $sil(o_i) < 0$.

To measure the quality of the whole clustering we use the *average silhouette*:

$$sil(\mathcal{C}) = \frac{1}{n} \sum_{i \in n} sil(o_i).$$

The higher this value is, the better the clustering is.

1. A singleton $\{o_i\}$ has silhouette equal to one because $a_i = 0$ and $b_i > 0$ (each element fits well in a cluster by its own).

2. If there is only one big cluster then for each $o_i \in n$ we have $sil(o_i) = -1$, because $b_i = 0$ and $a_i > 0$ (no element fits well in a cluster with all other elements).

The silhouette is not only used for assessing the clustering quality but can be helpful to guide the clustering task in many ways:

1. Given a cluster, the elements with lower silhouette might be excluded from the cluster to have more homogeneous clusters.

2. Given two clusterings of the same set of objects, done with the same clustering algorithm, but with different number of clusters, the one with higher average silhouette is preferable to the one with lower average silhouette. Thus, it can be used to decide k, the number of clusters in the clustering [Lamrous and Tailerb, 2006]. Experiments show that silhouette index is not very useful for this purpose.

2.3.2 External measures

In the following, we denote with $GT(S) = \{GT_1, \ldots, GT_k\}$ the *ground truth* partition formed by a collection of *classes*; and with $C = \{c_1, \ldots, c_k\}$ the outcome of the clustering algorithm that is a collection of *clusters*.

2.3.2.1 F-measure

The F-measure was introduced in [Larsen and Aone, 1999] and is based on the *precision* and *recall* that are concepts well known in the information retrieval literature [Kowalski, 1997], [Van Rijsbergen, 1979]. Given a cluster c_j and a class GT_i we have:

$$precision(GT_i, c_j) = \frac{|GT_i \cap c_j|}{|c_j|} \quad recall(GT_i, c_j) = \frac{|GT_i \cap c_j|}{|GT_i|},$$

Note that precision and recall are real numbers in the range $[0, 1]$. Intuitively precision measures the probability that an element of the class GT_i falls in the cluster c_j while recall is the probability that an element of the cluster c_j is also an element of the class GT_i. The F-measure $F(GT_i, c_j)$ of a cluster c_j and a class GT_i is the harmonic mean of precision and recall:

$$F(GT_i, c_j) = 2\frac{precision(iGT_, c_j) recall(GT_i, c_j)}{precision(GT_i, c_j) + recall(GT_i, c_j)}$$

The F-measure of an entire clustering is computed by the following formula:

$$F = \sum_i \frac{|GT_i|}{n} \max_j (F(GT_i, c_j)),$$

where n is the sum of the cardinality of all the classes. The value of F is in the range $[0, 1]$ and a higher value indicates better quality.

2.3.2.2 Entropy

Entropy is a widely used measure in information theory. In a nutshell we can use the relative entropy to measure the amount of uncertainty that we have about the ground truth provided the available information is the computed clustering. Given a cluster c_j and a class GT_i, we can define

$$p_{i,j} = \frac{|GT_i \cap c_j|}{|GT_i|},$$

$$E_j = \sum_i p_{i,j} \log p_{i,j},$$

$$E = \sum_j \frac{|c_j|}{n} E_j,$$

where n is the number of elements of the whole clustering. The value of E is in the range $[0, \log n]$ and a lower value indicates better quality.

2.3.2.3 Accuracy

While the entropy of a clustering is an average of the entropy of single clusters, a notion of accuracy is obtained using simply the maximum operator:

$$A_j = \max_i p_{i,j}$$

$$A = \sum_j \frac{|c_j|}{n} A_j.$$

The accuracy A is in the range $[0, 1]$ and a higher value indicates better quality.

2.3.2.4 Normalized mutual information

The *normalized mutual information* (see e.g. [Strehl, 2002, page 110]), comes from information theory and is defined as follows:

$$NMI(C, GT) = \frac{2}{\log |C||GT|} \sum_{c \in C} \sum_{c' \in GT} P(c, c') \cdot \log \frac{P(c, c')}{P(c) \cdot P(c')}$$

where $P(c)$ represents the probability that a randomly selected object o_j belongs to c, and $P(c, c')$ represents the probability that a randomly selected object o_j belongs to both c and c'. The normalization, achieved by the $\frac{2}{\log |C||GT|}$ factor, is necessary in order to account for the fact that the cardinalities of C and GT are in general different [Cover and Thomas, 1991].

Higher values of NMI mean better clustering quality. NMI is designed for hard clustering.

2.3.2.5 Normalized complementary entropy

In order to evaluate soft clustering, the *normalized complementary entropy* [Strehl, 2002, page 108] is often used. Here we describe a version of normalized complementary entropy in which we have changed the normalization factor so as to take overlapping clusters into account. The entropy of a cluster $c_j \in C$ is

$$E_j = \sum_{k=1}^{|GT|} -\frac{|GT_k \cap c_j|}{|GT_k|} \log \frac{|GT_k \cap c_j|}{|GT_k|}$$

The normalized complementary entropy of c_j is

$$NCE(c_j, GT) = 1 - \frac{E_j}{\log |GT|}$$

NCE ranges in the interval $[0, 1]$, and a greater value implies better quality of c_j. The complementary normalized entropy of C is the weighted average of the contributions of the single clusters in C. Let $n' = \sum_{j\in 1}^{|C|} |c_j|$ be the sum of the cardinalities of the clusters of C. Note that when clusters may overlap it holds that $n' \geq n$. Thus

$$NCE(C, GT) = \sum_{j\in 1}^{|C|} \frac{|c_j|}{n'} NCE(c_j, GT)$$

2.4 Improving the FPF algorithm for the k-center problem

One of the mayor effort we did in this thesis was devoted to improve the Furthest Point First algorithm from both the computational cost point of view and the output clustering quality. Since theoretically the FPF algorithm as proposed by Gonzalez [Gonzalez, 1985] is optimal (unless $P = NP$), only heuristics can be used to obtain better results and, in the worst case, it is not possible to go behind the theoretical bounds. We profiled FPF and analyzed the most computational expensive parts of the algorithm. We found that most of the distance computation are devoted to find the next furthest point. We observed that there are cases such that some distance computations can be avoided without changing the final clustering algorithm. In section 2.4.1 we describe our results in this sense. FPF clustering quality can be improved modifying part of the clustering schema. In section 2.4.2 we describe an approach that use the random sampling technique to improve clustering output quality, we call this algorithm M-FPF. Another crucial shortcomings of FPF is that it selects a set of centers not representative of the clusters. This phenomenon must be imputed to the fact that, when FPF creates a new center, it selects the furthest point from the previous selected centers and thus the new center can likely be close to a boundary of the subspace containing the data set. To overcame this problem in section 2.4.3 we modify M-FPF to use *medoids* instead of centers. Other domain specific modifications to FPF will be presented in chapters 5 and6.

2.4.1 Exploiting the triangular inequality to improve the FPF speed

We observed that most of the running time of the FPF algorithm is devoted to compute distances for finding the closest center to each point. More precisely at a generic iteration $1 < i \leq k$, after finding the center μ_k, $n - k$ distances must be computed to decide whether or not to assign a point to the new center. If this is done in a straightforward manner it takes $O(n)$ time per iteration, thus the total computational cost of the algorithm is $O(nk)$.

Exploiting the triangular inequality, in certain conditions we can avoid to compute the distances among all the points in a cluster and the new center being sure that they are closer to their center. Unfortunately the worst case time complexity still remain $O(nk)$ because the number of saved distance computations depends on data distribution and thus, it can not be predicted in advance. The modifications discussed here do not change the FPF output, they only speed up the algorithm. In chapter 5 we will discuss some approximations to speed up the algorithm which are data driven and thus not widely applicable. We modified the algorithm as follows: consider, in the FPF algorithm, any center c_i and its associated set of closest points $\mathcal{C}_i$. Store $\mathcal{C}_i$ as a ranked list, in order of decreasing distance to c_i. When a new center c_j is selected, scan $\mathcal{C}_i$ in decreasing order of distance, and stop scanning when, for a point $p \in \mathcal{C}_i$, it is the case that $d(p, c_i) \leq \frac{1}{2} d(c_j, c_i)$. By the triangular inequality, any point p that satisfies this condition cannot be closer to c_j than to c_i. This rule filters out from the scan points whose neighbor cannot possibly be c_j, thus significantly speeding up the identification of neighbors. Note that all distances between pairs of centers must be available; this implies an added $O(k^2)$ cost for computing and maintaining these distances. Note that this modified algorithm works in any metric space, hence in any vector space[2].

In the remainder of this thesis, when we will refer to FPF, we mean this version of the algorithm since the final output is identical to that of the original one.

2.4.2 Using a random sample

The efficiency of the algorithm is further improved by applying FPF algorithm not to the whole data set but only to a random sample of size $n' = \sqrt{nk}$ of the input points (sample size suggested in [Indyk, 1999]). Note that given that $k \leq n$, it is always true that $n' \leq n$. Then we add the remaining $(n - n')$ points to the cluster of their closest center, one by one.

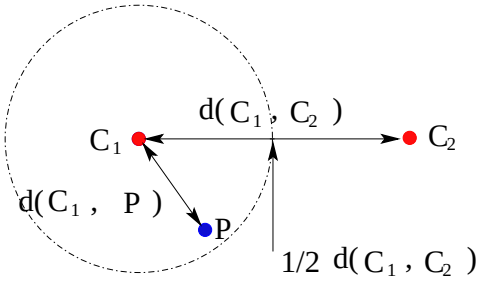

Figure 2.5. Exploiting the triangular inequality.

Also in the operation of insertion of the $(n - n')$ remaining points, the bottleneck is the time spent computing distances to the point to the closest center.

[2]We recall that any vector space is also a metric space, but not vice-versa.

According with [Phillips, 2002] this operation can be made more efficiently exploiting the triangular inequality (see figure 2.5), even if the worst case running time does not change.

Consider to have available the distances between all the pairs of centers of the clustering. Let p be the new point to be inserted in the clustering, by the triangular inequality if $\frac{1}{2}d(c_i, c_j) > d(c_i, p)$ then $d(c_i, p) < d(c_j, p)$. It means that the computation of the distance $d(c_j, p)$ can be safely avoided. Note that the distances between each pair of centers is available in this phase because they were already computed for the optimization described in section 2.4.1. We will refer to this algorithm as M-FPF.

M-FPF:
Data: Let O be the input set, k the number of desired clusters
Result: $\mathcal{C}$: a k-partition of O
Initialize R with a random sample of size $\sqrt{|O|k}$ elements of O;
$\mathcal{C}$ = **FPF**(R, k);
forall $C_i \in \mathcal{C}$ **do**
  μ_i = getCenter (C_i);
end
forall p *in* $O \setminus R$ **do**
  assign p to cluster C_i such that $d(p, \mu_i) < d(p, \mu_j), \forall j \neq i$;
end

Algorithm 3: M-FPF.

2.4.3 Using medoids as centers

The concept of medoid was introduced by Kaufman and Rousseeuw in [Kaufman and Rousseeuw, 1990]. Medoids have two main advantages with respect to centroids: first of all, they are elements of the input and not "artificial" objects. This make medoids available also in those environments in which the concept of centroid is not well defined or results artificious. Nevertheless, in many environments (i.e texts) centroids tends to become dense objects with a high number of features more of which of poor meaning. This makes centroids to lose representativeness and compute distances with them becomes more expensive with respect to distances between "normal" objects.

The main drawback of the original definition is that the clustering algorithm (Partition Around Medoids) and the computation of medoids is expensive. As illustrated in section 2.2.1.3 to overcome this disadvantage many different re-definitions of medoids were introduced in the literature.

In the context of the Furthest Point First heuristic where some input points are elected as cluster centers and are used to determinate which input points belong to the cluster, the restrictions of the use of centroids are not present. However, we observed that, although the objects selected from FPF as centers determine the points belonging to the cluster, they are not "centers" in the sense suggested by the

human intuition.

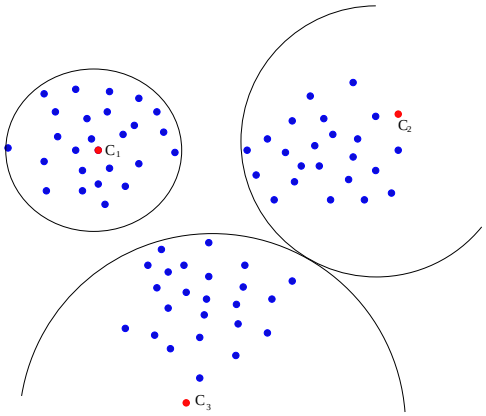

Figure 2.6. An example of clustering made using FPF.

Figure 2.6 shows a clustering with three clusters. The first center c_1 is central also in the human sense. c_2 is the furthest point from c_1. It can be easily observed that according to the human feeling it is not in the center of the cluster it defines. The same holds for c_3.

This fact can impact negatively on the final clustering quality. Moreover we will see in chapter 5 that there are applications in which we want to use the center as a representative point of the cluster. In that case c_2 and c_3 are not a good choice.

To understand how centers as defined in the original FPF algorithm can not be representative, consider the example in figure 2.7:

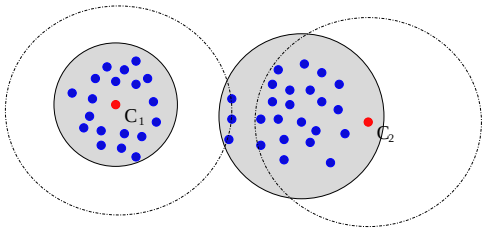

Figure 2.7. An example with two clusters made by FPF, in gray the ground truth.

In the figure there are two clusters. The two gray filled circles represent the expected correct clustering. Due to the choice of c_2 as the furthest point from c_1, the obtained clustering is the one formed from the two balls with centers c_1 and c_2 respectively. This has as side effect that some points (three in this example) are assigned to the wrong cluster. The error is due to the choice of c_2 that is not a good candidate to be a center. Starting from this observation, we used medoids instead of centers in our evolution of the FPF heuristic.

Our definition of medoid is quite different from those present in the literature. In fact we want to make the computation of medoids the more efficient as possible, and in certain cases quickly approximable.

Given a set of n points $O = \{o_1, \ldots, o_n\}$ endowed with a distance function $d()$, let $(a, b) = \arg\max_{x,y \in O^2} d(x, y)$ two diametral points for O. We say that the point $m \in O$ such that

$$m = \arg\min_{o_i \in O} |d(o_i, a) - d(o_i, b)| + |d(o_i, a) + d(o_i, b) - d(a, b)|$$

is the medoid of o.

This formula is composed by two main components: $|d(o_i, a) - d(o_i, b)|$ constraints the medoid to be as equidistant as possible from the diametral points, while $|d(o_i, a) + d(o_i, b) - d(a, b)|$ attempts to select the closest possible point to the diametral pair.

The medoid formulae can be generalized via weighting the two components

$$m = \arg\min_{o_i \in O} \alpha|d(o_i, a) - d(o_i, b)| + \beta|d(o_i, a) + d(o_i, b) - d(a, b)| \qquad (2.3)$$

where α and β are real numbers and $\alpha + \beta = 1$.

According with this definition, the computation of the medoid is quadratic in the number of points of O. In fact, one should compute the distance between all the possible pairs of objects of the input in order to find the diametral points. Following [Ömer Egeciolu and Kalantari, 1989] it is possible to find a good approximation a and b in linear time using the following search schema:

1. select a random point $p \in O$
2. in $O(n)$ find the furthest point from p and call it a
3. in $O(n)$ find the furthest point from a and call it b

Note that the n distances computed in the step 3 can be stored and used for the computation of formula (2.3).

According with the clustering strategy described in section 2.4.2, every time a new point p is inserted in a cluster, the medoid should be updated. This can be unacceptable for its computational cost. If the new point is not diametral, update can be done just computing $d(p, a)$ and $d(p, b)$. Otherwise all the distances must be recomputed. This effort can be reduced using another approximation: if for example $d(p, a) > d(a, b)$ and $d(p, a) > d(p, b)$, one can consider as new diametral pair the couple (a, p). This allow us to avoid the re-computation of the diametral points and, by keeping updated a cache of all the distances between each diametral point and all the other points, also the distances computation between a and the other points of the input set can be saved. Using this approximation it is possible to update a medoid at the cost of n distance function invocations instead of $3n$. We will refer to this algorithm as M-FPF-MD.

A further approximation of the medoid computation is still possible. Although it reduces drastically the cost during the update procedure, it is quite rough and should be used only in those online contexts where computational time makes the difference, or in those environments where there is a huge amount of redundant data. After the first time in which we find a, b and the medoid m, when a new point p is inserted in the cluster, the update of the medoid can be done using the following procedure:

- if $d(p,a) > d(a,b) \wedge d(p,a) > d(p,b)$ discard b and replace it with p
- if $d(p,b) > d(a,b) \wedge d(p,b) > d(p,a)$ discard a and replace it with p
- if $d(a,b) > d(p,a) \wedge d(a,b) > d(p,b)$:
  - if $|d(p,a) - d(p,b)| + |d(p,a) + d(p,b) - d(a,b)| < |d(m,a) - d(m,b)| + |d(m,a) + d(m,b) - d(a,b)|$ discard m and p become the new medoid
  - otherwise discard p

After the first initialization, this procedure requires only the computation of two distances. In chapter 5 we will use successfully this approximation for the generation of static storyboards from HSV vectors.

M-FPF-MD:
Data: Let O be the input set, k the number of desired clusters
Result: $\mathcal{C}$: a k-partition of O
Initialize R with a random sample of size $\sqrt{|O|k}$ elements of O;
$\mathcal{C}$ = **FPF**(R, k);
forall $C_i \in \mathcal{C}$ **do**
  t_i = getRandomPoint (C_i);
  $a_i = c_i$ such that $\max$ d(c_i, t_i) for each $c_i \in C_i$;
  $b_i = c_i$ such that $\max$ d(c_i, a_i) for each $c_i \in C_i$;
  $m_i = c_i$ such that
  $\min |d(c_i,a_i) - d(c_i,b_i)| + |d(c_i,a_i) + d(c_i,b_i) - d(a_i,b_i)|$;
end
forall p *in* $O \setminus R$ **do**
  assign p to cluster C_i such that $d(p,m_i) < d(p,m_j), \forall j \neq i$;
  if $d(p,b_i) > d(a_i,b_i)$ **then** $a_i = p$;
  if $d(p,a_i) > d(a_i,b_i)$ **then** $b_i = p$;
  if $d(p,b_i) > d(a_i,b_i)$ *or* $d(p,a_i) > d(a_i,b_i)$ **then**
    $m_i = c_i$ such that
    $\min |d(c_i,a_i) - d(c_i,b_i)| + |d(c_i,a_i) + d(c_i,b_i) - d(a_i,b_i)|$;
  end
end

Algorithm 4: M-FPF-MD.