1 Basic Concepts about Clustering

Let d be a positive integer and $\mathbb{R}$ the field of real numbers. For a set S of n points $\vec{p_i} \in \mathbb{R}^d$, we denote by $|S|$ the number of points of S. We consider the problem that we will call "k-means globally optimum clustering".

Definition 1. *The "k-means globally optimum clustering" is to split $S \subset \mathbb{R}^d$ of n points $\vec{p_i}$, $i = 1, \ldots, n$ into k disjoint nonempty subsets $S_1, \ldots, S_k$ called clusters in such a way that the following expression is minimized:*

$$f_{S_1,\ldots,S_k}(S) = \sum_{j=1}^{k} \sum_{\vec{p} \in S_j} \|\vec{p} - \vec{q_j}\|^2, \quad \textit{where } \vec{q_j} = \frac{\sum_{\vec{p} \in S_j} \vec{p}}{|S_j|}.$$

$S_1, \ldots, S_k$ is called an optimal partition of S.

It is well known that, given S, there always exists $\vec{q_1}, \ldots, \vec{q_k}$ such that the partition defined as,

$$S_j = \{\vec{p} \in S \ : \ \|\vec{p} - \vec{q_j}\|^2 < \|\vec{p} - \vec{q_l}\|^2, \ l = 1, \ldots, j-1 \\ \text{and } \|\vec{p} - \vec{q_j}\|^2 \leq \|\vec{p} - \vec{q_i}\|^2, \ i = j, \ldots, k\},$$

is an optimal partition. Indeed, the common approach to attack this problem is to use *Lloyd's heuristic* [6], which basically tries to find $\vec{q_1}, \ldots, \vec{q_k}$ through an iterative process. Although the standard algorithm was presented in [6], the term $k-$means was first used in [7] and, under minor modifications, performs quite well in practice, see [1, 10].

We will need the following concepts from topology:

- A set contained in $\mathbb{R}^d$ is *convex* if for any pair of points within the set, every point in the straight line segment that joins them is also within the object.
- Given a set of points $S \subset \mathbb{R}^d$, the convex hull of S is the smallest convex set of $\mathbb{R}^d$ which contains S.
- Given $\vec{a} \in \mathbb{R}^d - \{\vec{0}\}$ and $b \in \mathbb{R}$, the set $\mathcal{H} = \{\vec{x} \in \mathbb{R}^d : \vec{a} \cdot \vec{x} = b\}$ is called a hyperplane.
- A point $\vec{p} \in \mathbb{R}^d$ lays in the *left side* of hyperplane $\mathcal{H}$ if $\vec{a} \cdot \vec{p} > b$. If $\vec{a} \cdot \vec{p} < b$, the point $\vec{p}$ lays in the *right side* of hyperplane $\mathcal{H}$.
- An hyperplane $\mathcal{H}$ *separates* two sets S, $S' \subset \mathbb{R}^d$ if the points in S lays in the same side of $\mathcal{H}$ and none the points in S' lays in this side of $\mathcal{H}$.
- An hyperplane $\mathcal{H}$ *splits* a set S if any point $\vec{p} \in S$ lays in the right or the left side of $\mathcal{H}$.

We cite here the separating hyperplane theorem, see [3, page 46] for a proof.

Lemma 1. *For any two convex sets $S,\ S' \subset \mathbb{R}^d$ such that $S \cap S' = \emptyset$, there exists an hyperplane $\mathcal{H}$ that separates S and S'.*

As it was stated before, it is known that one optimal partition can be defined using k points of $\mathbb{R}^d$, called *centroids*. Partitions using centroids have a very interesting property.

Lemma 2. *Given a set of point $S \subset \mathbb{R}^d$ and $\vec{q_1}, \ldots, \vec{q_k} \in \mathbb{R}^d$, the partition $S_1, \ldots, S_k$ defined as*

$$S_j = \{\vec{p} \in S \ : \ \|\vec{p} - \vec{q_j}\|^2 < \|\vec{p} - \vec{q_l}\|^2,\ l = 1, \ldots, j-1 \\ \text{and } \|\vec{p} - \vec{q_j}\|^2 \leq \|\vec{p} - \vec{q_i}\|^2,\ i = j, \ldots, k\},$$

for $j = 1, \ldots, k$ satisfies:

- *the intersection of the convex hull of any two different clusters S_i, S_j is empty,*
- *for each pair S_i, S_j exists an hyperplane $\mathcal{H}$ that separates S_i and S_j.*

Proof. The first assertion of the lemma is proved by induction on k. For $k = 2$, consider the following sets,

$$\mathcal{S}_1 = \{\vec{p} \in \mathbb{R}^d \ : \ \|\vec{p} - \vec{q_1}\|^2 \leq \|\vec{p} - \vec{q_2}\|^2\}, \quad \mathcal{S}_2 = \{\vec{p} \in \mathbb{R}^d \ : \ \|\vec{p} - \vec{q_1}\|^2 > \|\vec{p} - \vec{q_2}\|^2\}.$$

It is trivial that both sets are convex, the intersection is empty and they contain S_1, S_2, respectively. By the definition of convex hull, $\mathcal{S}_1, \mathcal{S}_2$ must contain the convex hull of S_1, S_2 and this finishes the proof for $k = 2$.

The general case is proved in a similar way noting that the intersection of two convex sets is a convex set. So, the convex hull of

$$S_j = \{\vec{p} \in S \ : \ \|\vec{p} - \vec{q_j}\|^2 < \|\vec{p} - \vec{q_l}\|^2,\ l = 1, \ldots, j-1 \\ \text{and } \|\vec{p} - \vec{q_j}\|^2 \leq \|\vec{p} - \vec{q_i}\|^2,\ i = j, \ldots, k\},$$

is contained in the following set

$$\mathcal{S}_j = \{\vec{p} \in \mathbb{R}^d \ : \ \|\vec{p} - \vec{q_j}\|^2 < \|\vec{p} - \vec{q_l}\|^2,\ l = 1, \ldots, j-1 \\ \text{and } \|\vec{p} - \vec{q_j}\|^2 \leq \|\vec{p} - \vec{q_i}\|^2,\ i = j, \ldots, k\}.$$

Again, it is easy to check that $\mathcal{S}_j \cap \mathcal{S}_l = \emptyset$ for $l \neq j$ and each of these sets can be put as an intersection of convex sets. The rest of the proof of this item is straightforward.

The second assertion is a direct application of Lemma 1 and that $\mathcal{S}_j$ is a family of convex sets with empty intersection. □

2 Reverse Enumeration

Reverse Enumeration is a method for listing objects that satisfy a specific property. It was introduced in [2] to solve the following problem,

Problem 1. *Suppose that $G = (V, E)$ be an undirected graph, where V is a set of vertex and E is the edge set. Enumerate all the elements in V.*

The difficulty of this particular problem lays in the fact that V is not given explicitly, however given a node $v \in V$ it is possible to calculate its neighbors.

The problem of graph traversal is well-known, and there are several algorithms like breadth-first search and depth-first search, see [4, Page 597].

Unfortunately, these algorithms needs to mantain a data structure with all the nodes that have been visited in order to avoid looping endlessly in a cycle of the graph. This implies a big drawback in terms of memory usage.

For introducing a more efficient way of solving Problem 1 we need to introduce the definition of *local search*.

Definition 2. *A local search (G, R, f) is a triple satisfying $R \subset V$, f is a mapping $f : V \mapsto V$ with the following properties:*

- *$(v, f(v))$ is an edge of the graph for $v \in V - R$,*
- *for all $v \in V - R$, there exists a positive integer t depending on v such that $f^t(v) \in R$.*

The function f is said to be the local search function, R the set of solutions and G the underlaying graph structure.

Informally a local search algorithm is a way of explore a graph in a non systematic way, starting in any candidate and heading for the set of solutions R, see [8, Page 110] for a more detailed exposition.

A local search define a subgraph of G called the *trace* $T = (V, E(f))$ where,

$$E(f) = \{(v, f(v) \ : \ v \in V - R\}.$$

An important fact that appear in [2, Property 2.1] is that the trace of any local search contains all the nodes of G and each component contains only one element of the set R and no cycles.

So, most efficient way to output all the component is to traverse all the components of T, starting at every element of R. However, to do so, it is necessary to invert f, i. e. given v, find all the elements v', such that $f(v') = v$. For a general local search function f is normally difficult to find the preimages of a node v. In most of the cases where reverse enumeration is useful, there is an *adjacency oracle* which gives the neighbours of a node. The original definition can be found in [2], but here we simplify it for our purposes.

Definition 3. *A graph G is given by an adjacency oracle if there exists a function Adj that takes a node v an return an ordered list of neighbors of v.*

Procedure 1 General Reverse Search

Input: An adjacency oracle Adj, a local search (G, R, f)
Output: all nodes in V without any repetition

```
for all r ∈ R do
    yield r
    i, do, v = 1, True, r
    while do do
        while there are at least i elements in Adj(v) do
            let Next be the ith element of Adj(v)
            if f(Next) == v then
                v, i = Next, 0
            else
                i = i + 1
            end if
        end while
        if v == r then
            do = False
        else
            u, v, i = v, f(v), 1
            while ith element of Adj(v) is not u do
                i = i + 1
            end while
        end if
    end while
end for
```

So, given a graph G given by an adjacency oracle Adj and a local search (G, R, f), Algorithm 1 will output all the nodes. The complexity of the algorithm is given in [2, Theorem 2.2], which we enunciate here.

Theorem 1. *Let (G, R, f) be a local search where G is given by a adjacency oracle Adj. Suppose that $Adj(v)$ does not contain more than δ nodes for any $v \in V$. Then, if $t(f)$ and $t(Adj)$ are the times complexity for f and Adj, respectively, the time complexity of reverse search is of order of magnitude,*

$$\delta t(Adj)|V| + t(f)|E|.$$

In the next subsections, we show how to apply reverse search for the problem of enumerating all possible partitions defined by centroids.

2.1 Reverse search for two clustering problem

In order to adapt reverse search for the problem of enumerating all possible partitions defined by two clusters, we need to define the following objects:

- the graph $G = (V, E)$,
- a local search (G, R, f),
- an adjacency oracle Adj.

V is a complete description of all possible ways to split S by hyperplanes. It is represented by a set of hyperplanes satisfying two properties:

- each of the hyperplanes splits the set S in a different way.
- If S_1 and S_2 is a partition of S and the intersection of the convex hulls of S_1 and S_2 is empty, then there exists an hyperplane $\mathcal{H} \in V$ which separates S_1 and S_2.

In other words, V is a set of representatives of all possible partitions into two clusters given by hyperplanes.

Now, we define the set E implicitly by defining when two nodes are neighbors. Two hyperplanes $\mathcal{H}, \mathcal{H}' \in V$ are neighbours in graph G if and only if the points of S that lay in the left side of $\mathcal{H}$ are exactly the points that lay in the left side of $\mathcal{H}$, except for one point in S. Informally, two nodes are neighbors if the partitions they defined are the same except from one point.

2.1.1 Local search

The local search is an important part in the efficiency of the algorithm. For the rest of the paper, we suppose that points in S are ordered, so $S = \{\vec{p}_1, \ldots, \vec{p}_n\}$. One possible local search function is given in Algorithm 2. The set $R = \{\mathcal{R}\}$ contains only an hyperplane $\mathcal{R}$ with the property that all the points lay in the left side of this hyperplane.

Procedure 2 naive local search function f

Input: an hyperplane $\mathcal{H}$

Let S_1 and S_2 the partition defined by $\mathcal{H}$
for all $\vec{p}$ in S_2 **do**
if there exists an hyperplane $\mathcal{H}'$ that separate $S_1 \cup \{\vec{p}\}$ and $S_2 - \{\vec{p}\}$ **then**
return $\mathcal{H}'$
end if
end for
return $\mathcal{H}$

The problem with this proposal is that it is necessary to find an hyperplanes that separates two sets of points and this can be quite expensive in term of time, see [5]. We adapt the proposal for the local search (G, R, f) in [9] because it is more efficient and only requires to solve n different linear equations in one variable. Unlike Algorithm 2, this local search function depends on $\mathcal{R}$. Although

Procedure 3 local search function f

Input: An hyperplane $\mathcal{H}$ which separates S and the hyperplane $\mathcal{R}$
Output: An hyperplane $\mathcal{H}^{(t)}$ which splits S

$first\_minimum = 1$
$second\_minimum = 1$
for all $\vec{p}$ in S **do**
if $0 < Distance(\vec{p}, \mathcal{H}, \mathcal{R}) < 1$ and $first\_minimum > Distance(\vec{p}, \mathcal{H}, \mathcal{R})$ **then**
$second\_minimum = first\_minimum$
$first\_minimum = Distance(\vec{p}, \mathcal{H}, \mathcal{R})$
end if
end for
$\epsilon = second\_minimum - first\_minimum$
$t = first\_minimum + \epsilon/2$
return $\mathcal{H}^{(t)}$

in a cryptic way, the geometric intuition of what this function does is simple. First, consider the two hyperplanes,

$$\mathcal{H} = \{\vec{x} \in \mathbb{R}^d : \vec{a} \cdot \vec{x} = b\}, \quad \mathcal{R} = \{\vec{x} \in \mathbb{R}^d : \vec{r} \cdot \vec{x} = s\}. \tag{1}$$

We remark that $\mathcal{R}$ defines the trivial partition, because all points are in the left side of $\mathcal{R}$.

Now, define the following set of hyperplanes, depending on a parameter t,

$$\mathcal{H}^{(t)} = \{\vec{x} \in \mathbb{R}^d : (t\vec{a} + (1-t)\vec{r}) \cdot \vec{x} = tb + (1-t)s\}, \quad 0 \leq t \leq 1.$$

It is trivial to see that $\mathcal{H}^{(0)} = \mathcal{H}$, $\mathcal{H}^{(1)} = \mathcal{R}$, for some value $0 \leq t \leq 1$. Perturbating $\mathcal{H}$, in a way that it splits S in the same way, we can suppose that $\mathcal{H}^{(t)}$ contains at most one point in S for $0 < t < 1$.

In these conditions, there exists a value $0 < t < 1$, such that the partition defined by $\mathcal{H}$ and $\mathcal{H}^{(t)}$ differ in only one point $\vec{p}$. This point $\vec{p}$ lays in the right side of $\mathcal{H}$ and in the left side of $\mathcal{H}^{(t)}$.

Algorithm 3 returns $\mathcal{H}^{(t)}$ and it is easy to check that satifies all the conditions of a local search.

Procedure 4 Algorithm Distance

Input: Two hyperplanes $\mathcal{H}, \mathcal{R}$ and one point $\vec{p}$
Output: the value t such that $\mathcal{H}^{(t)}$ contains $\vec{p}$
let $\vec{a}, \vec{r}, b, s$ as in Equation (1)
return

$$\frac{\vec{r} \cdot \vec{p} + s}{(\vec{a} - \vec{r}) \cdot \vec{p} - b + s}$$

2.1.2 The adjacency oracle

The adjacency oracle is just an algorithm which that returns which partitions are adjacents. Again, we remark that two partitions are adjacent if they differ in only one point. This is equivalent to give which points are vertices of the convex hull of each of the clusters. To know if a point $\vec{p_i}$ is a vertix of the convex hull can be done solving the following linear program,

$$\begin{aligned} &\text{maximize} && \vec{p_i} \cdot \vec{x} - y, \\ &\text{subject to} && \begin{cases} \vec{p_j} \cdot \vec{x} - y \leq 0, & \vec{p_j} \text{ lies in the right side of the hyperplane} \\ \vec{p_j} \cdot \vec{x} - y \geq 0, & \vec{p_j} \text{ lies in the left side of the hyperplane} \end{cases} \\ & && j = 1, \ldots, n, \quad j \neq i. \end{aligned}$$

If the value is strictly greater than 0, then the point $\vec{p_i}$ is a vertex of the convex hull.

2.1.3 Time complexity of Reverse search

The time complexity of the algorithm is a direct application of Theorem 1. Also, notice that the algorithms here only required,

- the points in set S,
- the first hyperplane $\mathcal{R}$,
- the current hyperplane $\mathcal{H}$.

This means that it is only needed to keep track of $O(nd)$. Another important thing is that for any partition, there are at most n partitions that are adjacents. This is implied by the definition of adjacent partitions. Using this together and applying Theorem 1, we get the following result.

Theorem 2. *Let $l(n,d)$ be the number of operations necessary to solve a linear programming problem with n equations and d variables and P be the number of partitions of S defined by two centroids. Then, the implementation of Reverse search algorithm has time complexity $O(l(n,d)ndP)$ operations and space complexity $O(nd)$.*

Notice that this algorithm is polynomial in the size of the number of different partitions.

3 Reverse search for arbitrary k

The general clustering problem is done enumerating all possible partitions with less than k clusters. This correspond to allowing a partition to have empty clusters and it is not difficult to check that the number of partitions with no empty clusters and with empty clusters are of the same order of magnitude. So, we study the general case of enumerate all possible partitions with empty clusters.

The general case to enumerate all possible partitions of at most k clusters uses the hyperplanes found in the case $k = 2$. The hyperplanes found for $k = 2$ can be used to find all possible clusters that appear in a partition of S given by k centroids. The idea is giving in the following lemma.

Lemma 3. *Suppose that $\{\mathcal{H}_1, \ldots, \mathcal{H}_s\}$ are a set of hyperplanes satisfying that for any partitions into two clusters S_1, S_2 of S, there is an hyperplane $\mathcal{H}_i$ that separates S_1, S_2. Then, for any cluster S_j of a partition $S_1, \ldots, S_k$ of S defined by k centroids, there exists $k-1$ hyperplanes $\mathcal{H}_{i_\ell}$, $\ell = 1, \ldots, k, \ell \neq j$ such that $\mathcal{H}_{i_\ell}$ separates S_j and S_l.*

Proof. The proof is inmediate noticing that, by Lemma 2, there is an hyperplane separating two different clusters. This hyperplanes also induces two clusters in S which implies that there is an hyperplane in $\{\mathcal{H}_1, \ldots, \mathcal{H}_s\}$ which separates this two clusters. Repeating this process $k-1$ times, we get the result. □

Thanks to this results, it would be possible to make an algorithm to enumerate all possible partitions of S into k clusters. The idea would be the following,

- generate a set of all possible clusters that can appear in a partition of k clusters,
- for k elements of this set of clusters, check if they form a partition, i. e. their intersection is empty and their union is S. If that is the case, yield the clusters.

It is not difficult to show that the time complexity of this algorithm is polynomial on the number of different clusters that can appear in a partition of S into k clusters[1] and this polynomial is of degree at least k.

[1] not on k

We present a version of reverse search in order to obtain better complexity. Among other properties, the algorithm enumerates all possible partitions without repeating partitions with reordered clusters. We are presenting this as in the previous section, introducing all the elements necessary to define the reverse search procedure for this case.

The graph $G = (V, E)$ is defined in the following way, any vertex is a partition which is represented as a set of at most k binary list. Each list v has length n and represents a cluster in the following way, a point $\vec{p}_i$ belongs to the cluster if and only if $v[i] = 1$. Two vertices v, v' are adjacent if and only if one can be transform in the other by moving only one point from one cluster of the partition to another. Using this definitions, it is easy to define the local search and adjacency oracle. For all algorithms, it is implicitly supposed that all possible clusters that can appear in a partition of k clusters has been previously calculated. The calculation of all possible clusters does not affect the total complexity of the algorithm for enumerating partitions, because this calculations is done in linear time in the number of different enumerating partitions. These clusters are kept in a set $\mathcal{C}$.

The local search is defined in a very simple way. The set R of solutions is just the trivial partition, where all the points belong to one cluster. The local search function takes a partition and tries to pass one point of the "smallest" cluster a "bigger" cluster. The algorithm is,

- Take the cluster with the smallest number of elements or clusters if there are several of them,
- for each point of the selected clusters, move this point to the different clusters and check when this form a valid partition, i. e. all the clusters belong to $\mathcal{C}$.
- select the valid partition where the point moved to the cluster with the biggest number of elements. If there are several points which go to clusters with the same number of elements, move the point with smallest index.

This description should be enough to implement. The adjacency oracle works in a similar way. So given a partition, take every point in S, move it to each of the clusters or add it to a new cluster. Again, we can measure the efficiency of this algorithm, thanks to Theorem 1. It is important to notice that any $v \in V$ has at most n neighbors, so $|E|$ is at most $n|V|$ and to check the pertenence of an element to a set of N elements can be done in $\log N$

Theorem 3. *Let P be the number of partitions of S defined by k centroids and N be the number of different clusters which appears in some of these partitions. Then, the implementation of Reverse search algorithm has time complexity $O(Pn^2dk \log n)$ operations and space complexity $O(N)$.*

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[2] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1993.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

[5] David A. Elizondo. The linear separability problem: some testing methods. *IEEE Transactions on Neural Networks*, 17(2):330–344, 2006.

[6] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[7] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[8] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[9] Nora H. Sleumer. Hyperplane arrangements: Construction, visualization and applications. Master's thesis, Swiss Federal Institute of Technology, 2000.

[10] Chen Zhang and Shixiong Xia. K-means clustering algorithm with improved initial center. In *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on*, pages 790 –792, jan. 2009.