UNIVERSITÀ DEGLI STUDI DI CATANIA

FACOLTÀ DI SCIENZE MM. FF. NN.

DIPARTIMENTO DI MATEMATICA E INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA - XXV CICLO

Marco Antonio Aliotta

DATA MINING TECHNIQUES ON VOLCANO MONITORING

Tutor:

Dott.Ing. P. Montalto (INGV-OE)

Dott. Alfredo Pulvirenti (UNICT-DMI)

ANNO ACCADEMICO 2011-2012

Preface

The aim of this thesis is the study of data mining process able to discover implicit information from huge amount of data. In particular, indexing of datasets is studied to speed the efficiency of search algorithm. All of the presented techniques are applied in geophysical research field where the huge amount of data hide implicit information related to volcanic processes and their evolution over time. Data mining techniques, reported in details in the next chapters, are implemented with the aim of recurrent patterns analysis from heterogeneous data.

This thesis is organized as follows. Chapter 1 introduces the problem of searching in a metric space, showing the key applications (from text retrieval to computational biology and so on) and the basic concepts (e.g. metric distance function). The current solutions, together with a model for standardization, are presented in Chapter 2. A novel indexing structure, the K-Pole Tree, that uses a dynamic number of pivots to partition a metric space, is presented in Chapter 3, after a taxonomy of the state-of-the-art indexing algorithm. Experimental effectiveness of K-Pole Tree is compared to other efficient algorithms in Chapter 4, where proximity queries results are showed. In Chapter 5 a basic review of pattern recognition techniques is reported. In particular, DBSCAN Algorithm and SVM (Support Vector Machines) are discussed. Finally, Chapter 6 shows some geophysical applications where data mining techniques are applied for volcano data analysis and surveillance purpose. In particular, an application for clustering infrasound signals and another to index an thermal image database are presented.

Acknowledgments

"First I would like to thank my scientific colleagues for their fundamental cooperation: Dr. Andrea Cannata, Dr. Carmelo Cassisi and Dr. Ing. Michele Prestifilippo, loyal friends as well as colleagues. Without their help this thesis would not have been possible. Many thanks to my tutors Prof. Alfredo Pulvirenti and Dr. Ing. Placido Montalto. A particular mention to Placido, whose friendship since my childhood has supported my personal growth.

In addition, I would like to express my sincere gratitude to Prof. Alfredo Ferro and Dr. Domenico Patanè, Director of "INGV – Osservatorio Etneo" for their constant guidance and encouragement.

In the end, I wish to warmly thank my family for their support and the great help they have given me. Last, but not least, a special thought goes to my girlfriend Elisabetta who has always encouraged me to accomplish this work".

The Author

1. Indexing and searching in metric space

The problem of the search for elements of a set being *close* respect to a *query* element, according to a similarity criterion, has a wide range of applications in computer science, from *pattern recognition* to *information retrieval*, both textual and multimedial.

The case we want to focus on is the one where the similarity criterion defines a Metric Space, rather than dwelling on spatial vectors. For this purpose, a large number of solution for different areas were proposed, in many cases developed without *cross – knowledge*.

Because of this, the same ideas have been reinvented several times and have been given different presentations for the same approach. We now present a unified view of all the proposals to build Metric Spaces, to include their organization through a common model. Many approaches appear to be variations of a few basic concepts. We will organize this work into a taxonomy that allows us to separate the new algorithms from combinations of concepts that were not disclosed earlier because of the lack of communication between the different communities.

Some of the new techniques, created as combinations of the above have proven very competitive

We present experiments that confirm our results and compare them to existing approaches, in order to determine the optimal solution for our problem. At the end of this chapter we show some recommendations for those who engage with this problem and open questions for future developers.

Research is a fundamental problem of computer science, present in virtually all of its applications. In some of these, exposed search problems are very simple, while in others, more complex, it requires a more sophisticated form of search.

Search operation is typically performed on particular *data structures*, e.g. alphabetic or numeric information, which are scanned exactly. In this case, given a query, the set of strings *exactly* equal to the query itself is returned. Traditional databases are built around the concept of exact search: they are divided into records, each with a well-defined and comparable *key*. A query made to the database will return all *records* whose keys match the query.

There are more sophisticated types of research, such as range queries on numeric key or the search for a prefix of alphabet keys, which are based on the concept that the keys may be identical or not, or there is a linear ordering among the total keys. Even in recent

times, when in databases was made possible to storing new data types like images, research has yet been performed on a predetermined number of numeric or alphabetic keys.

With the evolution of information and communication technologies, collections of unstructured information are emerged. Not only different types of data as simple text, images, audio and video can they be asked; also to structure the information in records and keys is no longer possible. Such structuring is very a difficult task (both from human and computational point of view) and narrows in advance the types of queries that can be placed after. Even when a classical structure is possible, new applications such as data mining require access to the database through any field, not just those marked as key. Therefore, you need new models for searching collections of unstructured data.

The scenarios just discussed require more general search algorithms and models than those classically used for simple data. A unifying concept is that search of similarity or proximity, i.e. the search for database elements that are similar or close to a given element (query). The similarity is modeled with a distance function that satisfies the triangular inequality, and the set of objects is called a Metric Space. Since the problem has appeared in different areas, the solutions in their turn have appeared in many fields, also disconnected from each other, as statistics, artificial intelligence, databases, computational biology, pattern recognition and data mining, and others.

With solutions from several fields, it is not surprising that the same solutions have been reinvented many times, as an obvious combination of solutions that had not yet been disclosed, and that had not been against analytical or experimental comparisons. Above all, it has never been attempted to unify all of these solutions.

In many applications, the general problem is traversed from metric spaces in a vector space (the objects are represented as *k*-dimensional points with some interpretations of geometric similarity). It because the concept of similarity search appeared in first place in the spaces vector. This is a natural extension of the problem of finding the point closer to a plan. In this model there are optimal algorithms (on the size of the database) in both the average case that in the worst case [10] for the search of the nearest point.

Search algorithms for vector spaces are called *space access methods* (SAM). Among the most popular, we include the kd-trees [8, 9], R-trees [36] and the most recent Xtrees [11] (see also [57, 35] for an overview). Unfortunately, existing algorithms are strongly affected by the size of the vector space: search algorithms depend exponentially on the

size of the space (this is called “curse of dimensionality”). In practical terms, we consider that the problem becomes intractable when it exceeds the 20 dimensions. For this reason, many authors have proposed the use of a distance based on indexing techniques, which uses only the distance between the points and avoids reference to the coordinates, in order to avoid the curse of dimensionality.

This may be particularly effective when a set with an intrinsically low dimension is “embedded” within an artificial more dimensional set (e.g. a plan within a three-dimensional set). Therefore, generic metric spaces can be used, not only when the problem is not structured in coordinates, but also when the number of these coordinates is very high.

With respect to generic metric spaces, the problem has been addressed by several points of view. The objective in general is to build a data structure (or index) to reduce the number of calculations of distances at the time of query execution, since it is assumed that the distance is expensive to compute.

Some important progresses were made, especially in database communities, where a certain number of indexes and data structures based on distances were developed and studied from different perspectives.

1.1 Key Applications

In the following sections, we present some applications in which the concept of finding proximity appears. Since we have not yet submitted a formal model, we will not try to explain the connections between applications, but will resume the treatment of this in the next section.

1.1.1 Query by Content in Structured Database

In general, a given query to the database presents a fragment of a record information, and we need to find the entire record. In classical approach, the fragment presented is fixed (the key). In addition, the search for a key that is incomplete or incorrect is not allowed. On the other hand, in the most general approach required nowadays, the concept of search with a key is generalized to the search from an arbitrary subset of records, which present or not errors.

This kind of research has earned a great variety of names, e.g. range query, query by content or proximity searching. It is used in data mining (where the interesting parts of the records cannot be predetermined), when the information is not accurate, when we look for a certain range of values, when the search key may contain errors (e.g. a word that contains misspelling), etc.. A general solution to the problem of range queries from any field of a record is the Grid File [46]. The domain of a database is considered as a hyper-rectangle of size *k* (one for each field in the record), where each size has a sort of solidarity to the domain of the field (numeric or alphabetically). Each record in the database is considered as a point within the hyper-rectangle. A query specifies a sub-rectangle (e.g. a interval along each dimension), and returns all the points in the latter.
This does not address the problem of searching of not traditional data types, or mistakes that cannot be recovered with a range query. However, it converts the problem of original research to the problem of obtain, in a given space, all points "close" to a given query point. *Grid Files* essentially consist of an organization technique of the hard disk in order to obtain effective range queries in secondary memory.

1.1.2 Query by content on multimedia objects

New data types like images, fingerprints, audio and video (called "multimedia" data types) cannot be questioned appropriately using the classic way. Not only they cannot be ordered, but they cannot be compared for equality. Applications will never have interest in finding an audio segment that is equal to another given segment. Nonetheless, the probability that two different images are equal "pixel by pixel" is negligible unless they are digital copies of the same source. In multimedia applications, all queries require objects similar to the one given. Some important applications are recognition of well-defined biometric features (face, fingerprint, voice), and multimedia databases in general [1].
An example is a collection of images. Interesting queries could be something like: "find a picture of a lion with the savannah in the background". If the collection is labeled, and each label contains a full description of what the image contains, then our query example can be solved with a classical scheme. Unfortunately, this classification cannot be performed automatically with the image processing technology available today. The recognition of an object in real-world scenes is still in an immature state to perform tasks of this complexity. Moreover, we cannot predict in advance all queries that could

be used to label images for all possible types of requests. An alternative to the automatic classification consists in considering the query as an image of example, so that the system looks for all the images similar to the query. This can be constructed within a more complex system of feedback where the user approves or rejects the found image, and a new query is submitted with the approved images. It is also possible that the query is only a part of an image and the system must recover the entire image.

These approaches are based on the definition of a similarity function between objects. These functions are provided by an expert, but they pose no assumption about the type of query can be absolved. In many cases, the distance is obtained through a set of k "features" that are extracted from the object (e.g. in an image, a useful feature is the average color). Then, each object is represented by its k characteristics (an object becomes a point in a k-dimensional space) and we are once again in the case of a range query on a vector space. There is a growing community of scientists deeply involved with the development of such similarity measures [20, 12, 13].

1.1.3 Text Retrieval

Although it is not properly considered a multimedia data type, the finding of unstructured text poses problems similar to the finding of multimedia data. This is because text documents are generally unstructured, and this does not allow you to easily find the desired information. Textual documents can be interrogated to find strings that are present or not, but in many cases it is asked to find semantic concepts of interest. For example, an ideal scenario would allow the search on a text dictionary for a concept such as “free from obligations”, finding the word “ransom”. Such research problem cannot be properly established with the classic tools. A large number of researchers have worked on this problem for a long time [48, 34, 7]. A certain amount of similarity measures was revealed. The problem is mainly solved finding documents similar to a given query. You can even submit a document as query, so that the system finds documents similar to it.

Some approaches which make use of similarity are based on the mapping of a document into a vector of real values, in so that each dimension is a dictionary word and the importance of word in the document (prepared by some formula) is the coordinate of the document in this dimension. In this space functions similarity are defined. Note however that the number of dimensions of the space is really high (we talk of thousands

of dimensions, one for each word). Another problem related to the discovery of text is typing. Since more and more text databases with low quality control appear (just think about the World Wide Web to get an idea), and typing mistakes or those of using an OCR (optical character recognition) are commonplace both in the text that in the query, we have that document in which appears a word that contains a spelling error is no longer retrievable from a misspelled queries. There are patterns of similarity between words (changes in distance editing [49]), which work very well with the presence of such errors. In this case, given a word, you want to find all the words like it. Another application is related to spell checking, where we try (correct) variations of a mistyped word.

1.1.4 Computational biology

DNA and protein sequences are the basic objects of study in molecular biology. Since they can be modeled as strings, we refer to the problem of finding a given (sub) sequence of characters within a string. However, the fact that an exact overlap occurs is unlikely, and computational biologists are more interested in discovery of parts of longer sequences that are similar to the (small) given sequences. The reason that research is not exact is due to small difference chains that describe genetic creatures of the same species or similar. The similarity measure used is linked to the probability of mutations [56, 49]. Other problems are related to the construction of phylogenetic trees (a tree which graphically shows the evolutionary path of one or more species), the search for pattern of which only certain properties are known, and more.

1.1.5 Pattern Recognition and Functions Approximation

A simplified definition of pattern recognition is the construction of a approximator of functions. In this formulation of the problem we have a certain finite number of data samples, and each of them is labeled as belonging to a certain class. When a new sample of data is given, the system is required to label the sample with one of the labels of the known data. In other words, the classifier can be imagined as a function defined from a space of objects (samples of data) and to values into a set of labels. In this sense, all classifiers can be approximators of considered functions.

If the objects are m-dimensional vectors of real numbers, then a choice are the natural neural networks and fuzzy approximators of functions. Another popular universal approximator of functions, the k-nearest neighbor classifier, finds the k nearest objects to the not labeled sample, and assigns it the "majority" label of the k nearest objects. In contrast to neural networks and fuzzy classifiers, k-nearest neighbor technique has no training costs, but it has linear complexity if it does not use any indexing algorithm [31]. Other applications of the k-nearest neighbor classifier are estimated density [30] and learning "reinforcement" [52]. In general, any problem where it is desired to infer a function based on a finite set of samples of data represents a potential application.

1.1.6 Audio and Video Compression

The audio and video transmission over a narrow band channel is a key problem, for example in the audio / video conferencing based on the Internet. A frame (a static image in a video, or audio fragment) can be thought as formed by a certain amount of (possibly overlapping) sub-frames (16 x 16 sub-images in a video, for example). In a really brief description, the problem can be completely solved by sending the first frame, and for the subsequent frames sending only the sub-frame that has a significant difference with the earlier sub-frame. This description includes the MPEG standard.
The algorithms, in fact, use an under-frame buffer. Each time that a frame is going to be sent, it is searched (with a threshold of tolerance) in the sub-frame buffer and only if that fails then the whole sub-frame is added to the buffer. If the sub-frame is found then it is shipped only the index of similar frames found. This implies, of course, that a fast search algorithm of similarity should be embedded in the server in order to maintain a minimum rate of frames per second.

1.2 Basic Concepts

All the applications presented in the previous section share a common model, which basically consists in finding the nearest objects in according to a suitable similarity function, within a finite set of elements. In this section, we present the formal model that includes all cases discussed above.

We now introduce the basic notation for the problem of the discharge of proximity queries. The set X will denote the set of right or valid objects. A finite subset of them, U of cardinality $n = | U |$, is the set of objects where the search is performed. U will be called the dictionary, database, or simply our collection of objects or elements. The function d: $\mathrm{X} \times \mathrm{X} \rightarrow \mathrm{R}$ denotes the distance measure between objects (in particular the shorter is the distance, the closer or more similar will be the objects). The distance functions have the following properties:

$(p1)\ \forall\ x, y \in X, d(x, y) \geq 0$ *positivity*

$(p2)\ \forall\ x, y \in X, d(x, y) = d(y, x)$ *symmetry*

$(p3)\ \forall\ x \in X, d(x, x) = 0$ *reflexivity*

and in many cases

$(p4)\ \forall\ x, y \in X, x \neq y \rightarrow d(x, y) > 0$ *strict positiveness*

The properties of similarity listed above provide only a consistent definition of the function, and cannot be used to save distance computations in a proximity query. If d is really a distance, that is, if it satisfies

$(p5)\ \forall\ x, y, z \in X,\ d(x, y) \leq d(x, z) + d(z, y)$ *triangle inequality*

then the pair (X, d) is called a metric space.

If the distance function does not satisfy the property of strict positiveness ($p4$) then the space is called pseudo-metric space. Although for simplicity we do not consider pseudo-metric spaces in this work, all the presented techniques are easily adaptable to simply identify all the objects with zero distance as one single object. This process works because if ($p5$) is valid, it is easy to prove that

$d(x, y) = 0 \rightarrow \forall z, d(x, z) = d(y, z)$.

In some cases we can have an *almost – metric* space, where the symmetry property is not valid ($p2$). For example, if the objects are corners of a city and the distance corresponds to the distance a car must travel to move from one corner to another, then the existence of one-way streets makes the distance asymmetric. There are techniques to

derive the new symmetric distance function from an asymmetric one, as $d^l(x, y) = d(x, y) + d(y, x)$.

However, in order to limit the search range of a query when we use a symmetric function, we need a specific knowledge of the domain.

Finally, we relax the triangle inequality ($p5$) to $d(x, y) \leq \alpha d(x, z) + \beta d(z, y) + \delta$, and after some scaling we can search within this space using the same algorithms designed for metric spaces. If the distance is symmetric, then for consistency should be $\alpha = \beta$.

1.3 Proximity Query

There are mainly three types of queries of a certain interest in the metric spaces:

(a) Range Query (or proximity query)

Returns all elements that lie within a certain distance r from q, that is $\{u \in U / d(q, u) \leq r\}$. We denote this query as $(q, r)_d$.

(b) Nearest Neighbor Queries

It finds items nearest to q in U, that is $\{u \in U / \forall v \in U, d(q, u) \leq d(q, v)\}$. In some cases we just find a single item (in continuous spaces still there is normally only one element that satisfies the query).

We may also impose a maximum distance r^* (tolerance threshold), such that if the closest element is at a distance greater than r^* from the query, then it will not return any items.

(c) k-Nearest Neighbor Query

It finds the k closest elements to q in U, which determines a set $A \subseteq U$ such that:

$| A | = k \wedge \forall u \in A, v \in U - A, d(q, u) \leq d(q, v)$. Note that in the case of equal elements, any set of k elements that satisfy the condition will be acceptable.

The most basic type of query is of the type (a). The left part of Figure 1 shows a query on a set of points that we use as an example. For simplicity we will use R^2 as metric space. A proximity query will be so close to a pair $(q, r)_d$, with q element (possibly) not belonging to X and a real number r that indicates the radius (or tolerance) of the query. The set $\{u \in U / d(q, u) \leq r\}$ will be called the *result* of the proximity query. The query

is specified as a query "*d*-type" because for different distance functions there will be different query results (e.g. using the Euclidean distance we will have a different result from what we would get using the Manhattan distance, etc.).
The other two types of queries are usually fixed by using a variant of range queries. For example, the query type (*b*) is usually solved as a range query where the radius is initially placed as infinite, and is decreased at a distance less and less as elements closer to the query are found.

This is usually coupled to a heuristic trying to get items that are close to the query as soon as possible (the problem is increasingly easy for small radii). The queries of type (*c*) are normally resolved as a variant of type (*b*), where at each instant the k-nearest elements are maintained and the current radius is the distance between the query and the farther object from these k elements (given that most distant of these elements become free of interest).
Another widely used algorithm for query of type (*b*) and (*c*) based on those of type (*a*) is to search with fixed radius $r = 2^i \varepsilon$, starting with $i = 0$ and incrementing until the desired number of elements (or more) "falls" in the search radius $r = 2^i \varepsilon$. After that, the radius is redefined from $r = 2^{i-1} \varepsilon$ to $r = 2^i \varepsilon$, until the exact number of elements is included.

The total CPU time for calculating a query can be divided into

$$T = \#\text{ distance calculations} \times d(\,)\text{ complexity} + \text{extra } CPU \text{ time}$$

and we want to minimize T.

In many applications, however, $d()$ computation is so expensive that the extra *CPU* time can be neglected. This is the model that will be used in this thesis, and therefore the number of calculated distances will measure the complexity of the algorithms. We also take into account a certain amount of linear *CPU* (but reasonable) work, as soon as the number of distance calculations remains low. However, we set to some extent also the attention to the so-called extra *CPU* time.
It is clear that all the query types mentioned above may be solved examining the entire dictionary *U*. In fact, if we are not allowed to perform a preprocessing on the data, e.g. building the *index* data structure, then exhaustive investigation remains the only way forward. An *Indexing Algorithm* is an off-line procedure that builds in advance a data

structure (or index), designed to save distance computations during the execution of proximity queries. This data structure can be expensive to build, but the cost spent in the construction process will be amortized saving distance computations on many queries posed to the database. The intention is, therefore, to design efficient indexing algorithms in order to reduce the number of distance calculations. All these structures are working on the basis of discarding elements using the triangle inequality (the only property that actually saves distance computations).

1.4 Vector Spaces

If the elements of the metric space (X, d) are t-uples of real numbers (actually t-uples of any field) then the pair is called a finite-dimensional space vector, or more briefly a vector space.

A k-dimensional vector space is a particular metric space where objects are identified with k real-valued coordinates (x_1,…, x_k). There are many possibilities for the type of distance function to be used, but by far the most used is the family of distances L_s (or Minkowski), defined as

$$L_s((x_1,\ldots,x_k),(y_1,\ldots,y_k)) = \left(\sum_{i=1}^{k}|x_i - y_i|^s\right)^{1/s}$$

The right side of Figure 1 illustrates some of these distances. For example, the distance L_1 returns the sum of the differences between the coordinates. It is called also *block* or *Manhattan* distance, since in two dimensions it corresponds to the distance between two points in a city composed of rectangular blocks. The points at the same distance r from a given point p form a rectangle with the center in p and rotated by 45 degrees, and the distance between two points to the opposite corners of the rectangle is $2r$.

The distance L_2 is better known as the Euclidean distance, since it corresponds to our vision of spatial distance. The points at the same distance r from a given point p form a sphere of diameter $2r$, centered at p.

The most used family member together with L_2 is L_∞, that is equivalent to taking the limit of the formula L_s when s tends to infinity. The result is that the distance between two points is the maximum difference between the coordinates:

$$L_\infty((x_1,\ldots,x_k),(y_1,\ldots,y_k)) = \max_{i=1}^{k} |x_i - y_i|$$

and the points at the same distance *r* from a given point *p* form a rectangle of length *2r* centered at *p*. Distance plays a key role in this study.

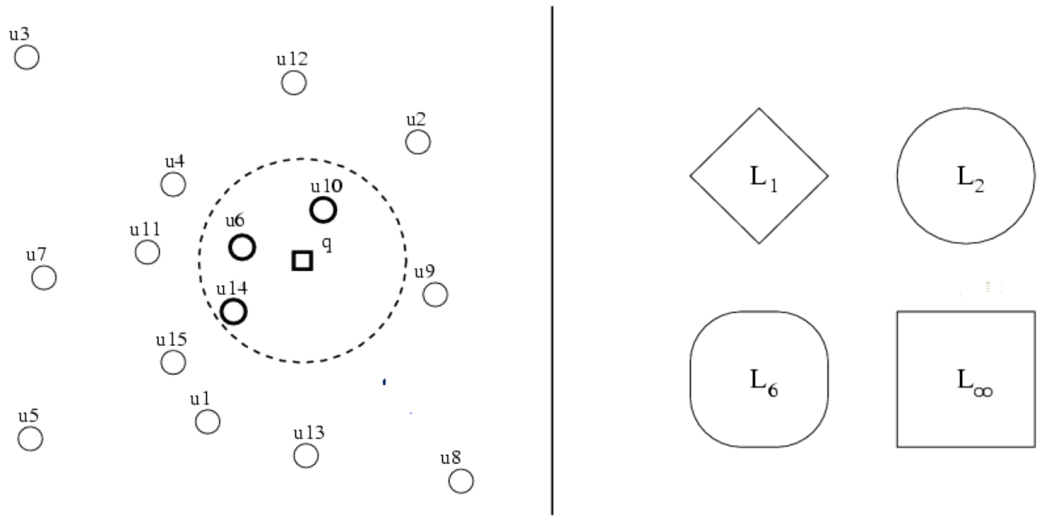


Figure 1 - *At left, an example of a range query on a set of points. At right, the set of points at the same distance from a central point, for different Minkowski distances.*

2. Overview of Current Solutions

In this chapter, we describe briefly the existing indexes to structure metric spaces. Since we have not yet developed the concepts of a unifying perspective, the first description will be maintained at an intuitive level, without any attempt to analyze why some ideas are better or worse than others. We divide the description into four parts: the first deals with data structures based on the discrete distance functions, i.e. functions that produce a set of small values, the second part describes the indexes based on continuous distance functions, where the set of alternatives is infinite or very large. In third section, we consider other methods such as clustering and spaces vector mapping. Finally, we consider briefly specific approximation algorithms for this problem.

2.1 Discrete Distance Functions

- BKT (Burkhard-Keller Tree) [19]: probably the first solution for searching in general metric spaces.

- FQT (Fixed-Queries Tree) [5] is a further development of the BKT. In it, all stored pivot in nodes at the same level are equal.

- FHQT (Fixed-Height FQT) [54] is a variant of the FQT, whose peculiarity is the fact that the leaves are all at the same height h.

- FQA (Fixed Queries Array) [24]: it is not really a tree, but simply a compact representation of a FHQT.

- Hybrids: In [51] is proposed a solution where is made use of more than one element for a node. In practice, this implies a sort of fusion between BKT and FQT.

- Adaptations to a continuous function: In [5] refers to the fact that structures mentioned above can be adapted for continuous distance functions, assigning a range of distances to each branch of the tree. However, it is not specified how to get it. Some approaches for the constant functions distance are discussed below.

2.2 Continue Distance Functions

Although these data structures have been designed specifically for the continuous case, they can be used without modification for discrete spaces.

- VPT (Vantage-Point Tree): It was the first approach designed to constant function distance. At first called "Metric Tree" [54], was later renamed "Vantage-Point Tree". It is a binary tree built recursively.

- MVPT (Multi-Vantage-Point Tree) is an extension of the Vantage-Point Tree, which uses an m-ary tree instead of a binary one. The authors, in [15, 16] experimentally demonstrate how the use of a tree m-ary instead of binary leads to slightly better results (but not in all cases).

- BST (bisector Tree) [108]: "Bisector Trees" are constructed recursively according to two centers c1, c2, selected each time.

- VPF (Exclude Middle Vantage Point Forest) [60]: another generalization of VPT. In this case there are multiple trees simultaneously.

- GHT (Generalized-Hyperplane Tree) [54]: this is again a tree built recursively. It is shown that, for higher dimensions, it works better than the VPT.

- GNAT (Geometric Near-neighbor Access Tree) [16] is the extension of GHT to an m-ary tree. Experimental analyses show that the GHT is worse of the VPT, which is however beaten by GNAT only for arity of 50 and 100.

- VT (Voronoi Tree) is proposed in [107] as an improvement of BST, where this time the tree has two or three elements (and children) for each node.

- MT (M-Tree) [27]: The purpose of this data structure is to provide dynamic capacity and good I/O performance, in addition to the calculation of a few distances.

- AESA (Approximating Eliminating Search Algorithm) [55]: an algorithm very similar to those presented so far, but surprisingly better than them. The data structure is simply a matrix. The idea is similar to that of FQT, but with some differences. The problem with this algorithm is that it requires $O(n^2)$ space and $O(n^2)$ time of construction. This is unacceptable for any database, except very small ones.

- LAESA (Linear AESA) [42]: is a new version of AESA, using k fixed pivots, in order to lower the space and construction time to $O(kn)$. In this case the differences with the FHQT become minimum.

In [41] another algorithm, that produces a similar structure to GHT using the same pivots, but with sub-linear CPU time, is presented.

- SAT (Spatial Approximation Tree) presented in [44], this algorithm does not use pivot to partition the set, but relies on "space" approximations.

2.3 Other techniques

- Mapping: a natural and interesting reduction in the problem of proximity finding is the "mapping" of the original space into a vector space. In this way, each point of the original metric space will be represented by a point in the objective vector space. The two spaces will be connected by two distances, the original $d(x, y)$ and the distance $d'(\phi(x), \phi(y))$ in projected space. Mapping studies are presented in [33].
A more elaborate version of this idea was introduced by *fastmap* [32]. In this case, a n-dimensional space is mapped in a m-dimensional one, with $n > m$.

- Clustering: this technique is employed in a wide spectrum of applications [39]. The overall objective is to divide a set into subsets of elements that are close to one another within the same subset. There are some approaches that use clustering to index metric spaces.

A technique proposed in [19] is to recursively divide the set U in compact subsets U_i and choose the most representative for each of them.
It calculates the numbers $r_i = max\{d(p_i,u) / u \in U_i\}$ (are the upper limits of the radii of the various subsets). To find the nearest objects to the query, it is compared with all the *pi* and the sets are considered starting from the one closest to the most distant. The r_i are used to determine that there cannot be elements in some interesting subset U_i.

2.4 Approximation and probabilistic algorithms

In order to complete our description we include a brief description of an important branch of the similarity search, where it is allowed a relaxation of the accuracy of the query to get more speed about the complexity of the execution time. In some applications, this is reasonable because the modeling of the metric space already includes an approximation of the right answer, and therefore a second approximation to the search time can be considered acceptable. Additionally to the query you also specify an accuracy parameter ε and to check the outcome of the query away (in some sense) from the correct result. A reasonable behavior for this type of algorithms is to asymptotically reach the correct answer, for ε to strive for zero, and on the other hand speed up the algorithm, losing precision, with ε striving to the opposite direction.

The alternative to the exact similarity search is called fuzzy similarity search, and includes approximate and probabilistic algorithms.

Approximation algorithms are considered in depth in [57]. As example, we mention an algorithm for approximate nearest neighbor search for real-valued vector spaces using any of the metrics of L_s Minkowski [2]. The data structure used by this algorithm is called *BDtree*.

2.5 Summary Table

Table 1 summarizes the complexities of different approaches. These have been obtained or derived from the respective documents, in which it is also suggested that different approaches using different (and incompatible) assumptions and in many cases only coarse or no analysis are provided. Keep in mind that the complexity of the query is always considered in the average case, and that in worst case it is forced to compare all the elements.

Data Structure	Spatial Complexity	Build Complexity	Query Complexity	Extra CPU Time (Per Query)
BKT	n pointers	$O(n \log n)$	$O(n^{\alpha})$	-
FQT	$n..n \log n$ pointers	$O(n \log n)$	$O(n^{\alpha})$	-
FHQT	$n..nh$ pointers	$O(nh)$	$O(\log n)$ if $h = \log n$	$O(n^{\alpha})$
FQA	nhb bits	$O(nh)$	$O(\log n)$ if $h = \log n$	$O(n^{\alpha} \log n)$
VPT	n pointers	$O(n \log n)$	O(log n) (*)	-
MVPT	n pointers	$O(n \log n)$	O(log n) (*)	-
BST	n pointers	$O(n \log n)$	not analyzed	-
VPF	n pointers	$O(n^{2-\alpha})$	$O(n^{1-\alpha} \log n)$ (*)	-
GHT	n pointers	$O(n \log n)$	$O(\text{polylog } n)$	-
GNAT	nm^2 distances	$O(nm \log_m n)$	$O(\text{polylog } n)$	-
VT	n pointers	$O(n \log n)$	not analyzed	-
MT	n pointers	$O(n(m..m^2) \log_m n)$	not analyzed	-
AESA	n^2 distances	$O(n^2)$	$O(1)$	$O(n)..O(n^2)$
LAESA	kn distances	$O(kn)$	$k + O(1)$ if k is big	$O(\log n)..O(kn)$
SAT	n pointers	$O(n \log n / \log \log n)$	$O(n^{1-\theta(1/\log \log n)})$	-

(*) only valid for very small search radii

Table 1: *average complexity of existing approaches, according to their documents. Time complexity only considers n, no other parameters such as their size. Space complexity refers to the more expensive memory unit used. α is a number between 0 and 1, different for each structure, while the other letters are parameters unique to each structure.*

2.6 A Model for Standardization

At first glance, data structures and indexing algorithms seem emerge from a great diversity, and different approaches are analyzed separately, often under different assumptions. Currently, the only realistic way to compare two different algorithms is to apply the same set of data.

In this section, we introduce a formal model of unification. The intention is to provide a common model to analyze all the existing approaches for the proximity search. The conclusions of this section may be summarized in Figure 2. All the indexing algorithms part the set *U* into subsets.

An index which allows to determine a collection of candidate sets whose items are relevant to the query is constructed. At the execution time of the query, the index is scanned in search of relevant subsets (the cost of this is called "internal complexity") and these subsets are checked exhaustively (this corresponds to "external complexity" of the search).

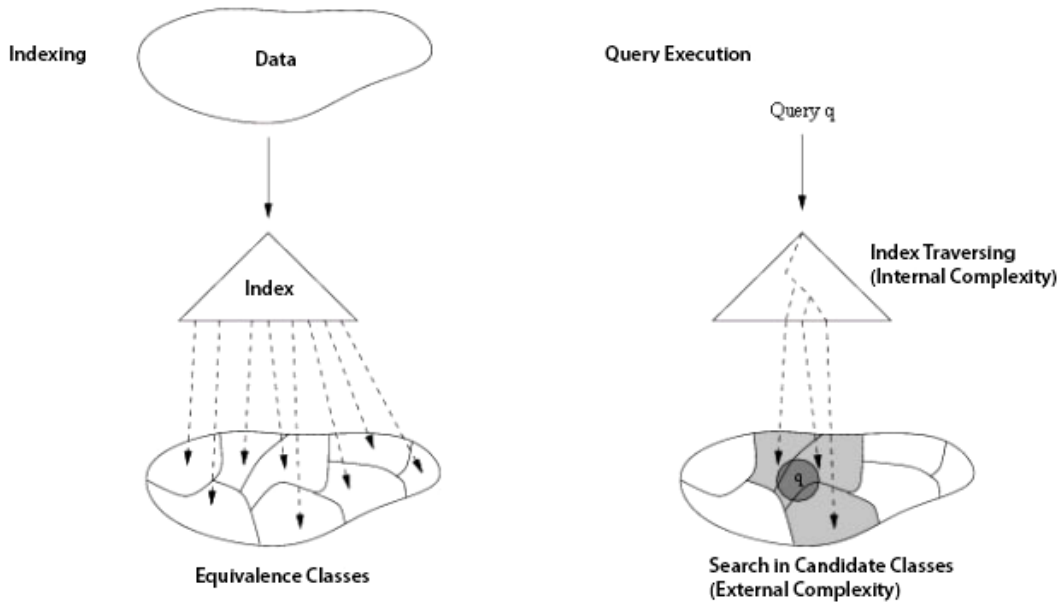


Figure 2 - *A unified model for indexing and query execution of metric spaces, equivalence Relations and Co-sets.*

Given a set X, a partition $\pi(X) = \{ \pi_1, \pi_2, \}$ is a subset of $P(X)$ (parts of X) such that each element belongs exactly to one partition. A *relation*, denoted by $\approx$ is a subset of the Cartesian product $X \times X$ (the set of ordered pairs) of X. Two elements x, y will be

related, and this is denoted by $x \approx y$, if the pair (x, y) is present in the subset. A relation $\approx$ is called *equivalence* if it satisfies, for all $x, y, z \in X$, the reflective $(x \approx x)$, symmetric $(x \approx y \Leftrightarrow y \approx x)$ and transitive $(x \approx y \wedge y \approx z \rightarrow x \Leftrightarrow z)$ property.

It can be shown that each partition induces a relationship of equivalence $\approx$, and vice versa, every equivalence relation induces a partition [25]. Therefore, two elements are related if they belong to the same partition. Each element π_i of the partition is then called class of equivalence.

Given the set X and the equivalence relation $\approx$, the set quotient $\pi(X) = X / \approx$ can be obtained, which indicates the set of equivalence classes or the *cosets*, obtained by applying the equivalence relation to the whole X.

The importance of equivalence classes in this study is represented by the possibility of using them on a metric space so as to derive a new metric space from the quotient set. This new metric space is more *coarse* than the original.

2.7 Indexes and Partitions

The equivalence classes obtained through an equivalence relation of a metric space can be considered as objects of a new metric space, as soon as a distance function on this new metric space is defined.

We then introduce the function $D_0 : \pi (X) \times (X) \rightarrow R$, now established in *quotient* set. Unfortunately this function does not satisfy the triangular inequality, so is not suitable for indexing. However, we can use any distance D that satisfies the properties of metric spaces and which is limited by D_0 (e.g. $D([x], [y]) \leq D_0 ([x], [y])$). In this case we call D an *extension* of d. Since D is a distance, it follows that $(X / \approx, D)$ is a metric space and therefore we can perform queries in a co-set in the same way we perform it in a set. Thus, we can convert a search problem in another, possibly easier to solve. For a given query $(Q, r)_d$, first we find the equivalence class to which q belongs (let it $[q]$. Then, using the new function D, the query is transformed into $([q], r)$ D. Then, we just run an exhaustive search on the candidate list (now using the original distance), to obtain the result of the query $(q, r)_d$. The above procedure is used in virtually all the indexing algorithms.

In other words:

All existing indexing algorithms for proximity search consist in building the equivalence classes, discard some, and exhaustively search in the remaining. As seen briefly above, the most important compromise in the design of the partition of equivalence is to balance the cost to find $[q]$ and check the final list of candidates.

In Figure 3 we can see a schematic example of this idea. We divide the space into different regions (equivalence classes). The objects within each region are indistinguishable. We can consider them as elements of a new metric space. To find the result, rather than examine exhaustively the whole dictionary, we examine only the classes that contain potentially interesting objects. In other words, if a class can contain an element that can be returned from the query result, then the class will examined (see also the rings considered in Figure 3).

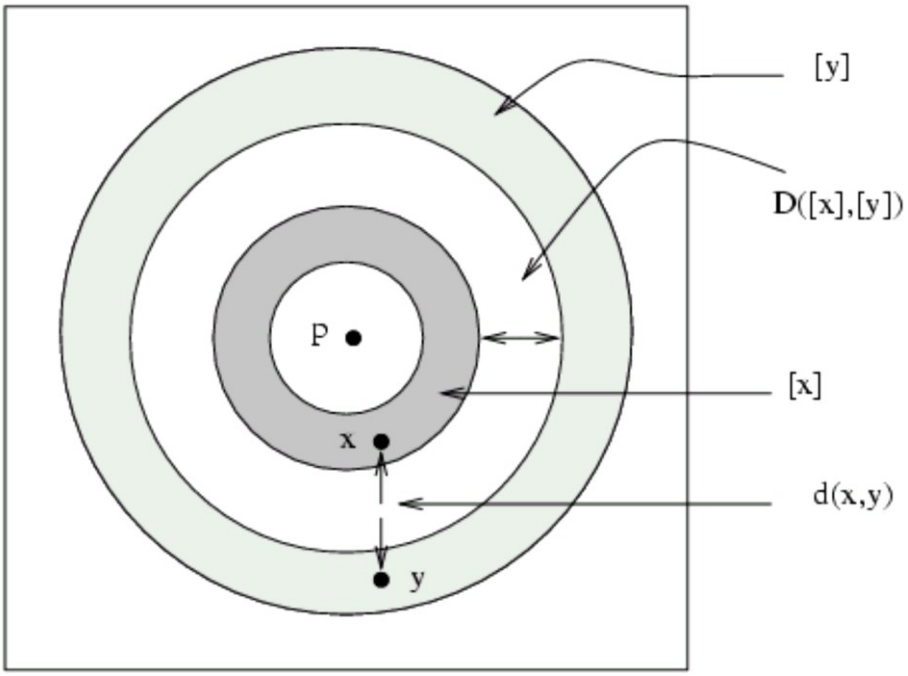


Figure 3 - *Two points x and y, and their equivalence classes (colored rings). D returns the minimum distance between the rings, which represents the lower bound for the distance between x and y.*

2.8 Efficiency measures

As stated previously, many of the ranking algorithms depend on the construction of an equivalence class. The corresponding search algorithms consist of two parts:

1. Find the classes that are relevant to the query

2. Examine in exhaustive mode all the elements of those classes

The first part is linked to the evaluation of certain distances *d*, and can also require extra CPU time. Distance evaluations performed at this stage are internal calls, and their sum defines the external complexity. The second part is about comparing the query with the list of candidates. These evaluations of *d* are called *external* and give rise to external complexity, which is linked to the discriminating strength of distance *D*.
We define the discriminating strength as the ratio between the number of objects in the list of candidates and the actual output of the query *q*, averaged over all $q \in X$.

Note that this depends on the search radius r. Discriminating force serves as an indicator of the efficiency and appropriateness of the equivalence relation (or in correspondence, the distance function *D*). In general, to have a greater discriminating force will result in a greater costs in terms of internal assessments. Indexing schemes need to balance the complexity of the search for the relevant classes and the discriminating force of *D*.

2.9 Location of a partition

Equivalence classes can be imagined as a set of cells that do not intersect in space, where each element within a given cell belongs to the same equivalence class [10]. However, the definition of equivalence class is not confined to a single cell. One consequence of this is that you need an additional property that will be called location, whose meaning is “how the equivalence class resembles a cell”. A partition is nonlocal in cases where classes are partitioned (see Figure 4) embracing a non-compact area in space.
It is natural to expect greater efficiency, e.g. more discriminating power, from a partition that is local to a non-local one. This is because in a non-local partition the list of candidates obtained with the distance *D* will contain elements far enough away from the query.

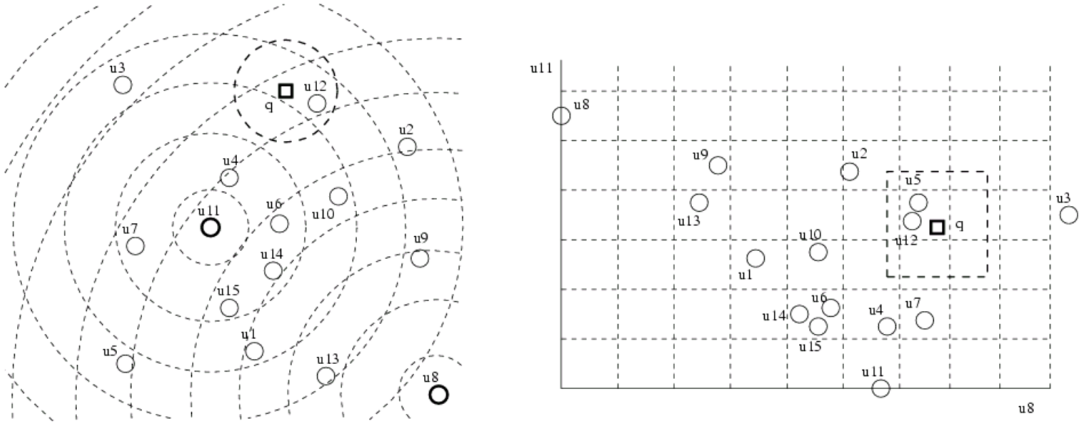


Figure 4 - *An equivalence relation induced by the intersection of rings centered in two pivots, and how the query is processed.*

Note that in Figure 4 the fragmentation disappears as soon as we insert the third pivot. In a vector space with k dimensions, it suffices to consider $k + 1$ pivot in general position[1] to obtain a local partition. This applies also for general metric spaces. However, to obtain local partitions is not enough: even considering local partitions assuming uniformly distributed data, we will verify that inevitably some cells remain empty, whose number grows exponentially with the space size.

2.10 Intrinsic Dimension

As explained, one of the reasons for the development of search algorithms for general metric spaces is the existence of the so-called high dimensionality. It because the traditional indexing techniques for vector spaces depend exponentially on the size of the space. In other words, if a space vector has a large number of coordinates, then an indexing algorithm that uses explicit information on the coordinates (such as *kd*-tree) requires exponential time (in terms of size) for answer the query. This has motivated research on the so-called indexing algorithms based on distance, which does not use explicit information on coordinates. This works very well in some vector spaces of apparently high dimension but actually small. An example is a 5-point set with the last three coordinates that are always equal to zero, or a more sophisticated case where the points lie in a 2-dimensional plane immersed in a 5-dimensional space. However, it is

[1] That is, not lying in a (k-1) dimensional hyper-plane

also possible that data may have a high intrinsic dimensionality. The concept of high dimensionality is not exclusive of vector spaces, but it can be characterized in metric spaces. If a vector space has an intrinsically low dimension, then considering it as a metric space can be advantageous, while for a high-dimensional vector space, the corresponding metric space in each case will be intractable. The following describes the effect of high dimensionality, and why it makes the problem intractable [22]. Let us consider the distance histogram between objects in the dictionary. This information is mentioned in many articles, in particular [16, 27]. Fixed a u element of the dictionary, let us consider the histogram of the distances $d(u, u_i)$, $u_i \in U$. It is clear that the number of elements within $(u, r)_d$ is proportional to the area under the histogram, from 0 to r included. Now, if we calculate the histogram considering all possible pairs $d(u_i, u_j)$, the area described above is proportional to the average number of elements provided for a query of radius r.

It is reasonable to assume that the queries are distributed according to the same laws which rule the elements of the dictionary. In other words, the distribution of the distances between the query and the elements of the dictionary appears as the histogram calculated for the data set. For the search range algorithm, if we select a ring having its center in the element u^*, with radii $d(u^*,q) - r$ e $d(u^*, q) + r$, the ratio of elements captured in the ring is, on average, approximated by the area under function density in the range $[d(u^*,q) - r, d(u^*,q) + r]$. For each *pivot-based* algorithm, the value of the above described area governs the behavior of the algorithm itself. At each step the number of deleted items is proportional to $1 - n_r$. If $n_r = 1$ then there is no elimination. We now show that in some cases we can infer that the discriminating force of the algorithm may be very low. Let us assume that the interval where the function density is greater than zero is $[r_a, r_b]$. We note that:

1. r_a is the average distance between a given element and its closest element.

2. r_b is the average distance between a given element and its farthest element.

3. If $r < r_a$, then the expected number of items in the dictionary within a sphere centered on the query with radius r is zero.

Moreover one can prove the following property:

If $2r_a > r_b$ then $n_r = 1$. In other words, on average, no elements are deleted.

This property is a coarse measure of the dependence on the size in algorithms based on the pivot. In the Euclidean spaces R^k, the distribution of distances between two random points becomes more "average" and has less variance in k increases. Therefore, the histogram of distances is "stretched" towards right (Figure 5), consequently stretching upwards. You can define the intrinsic dimensionality of a given metric space, considering the generic histogram shape.

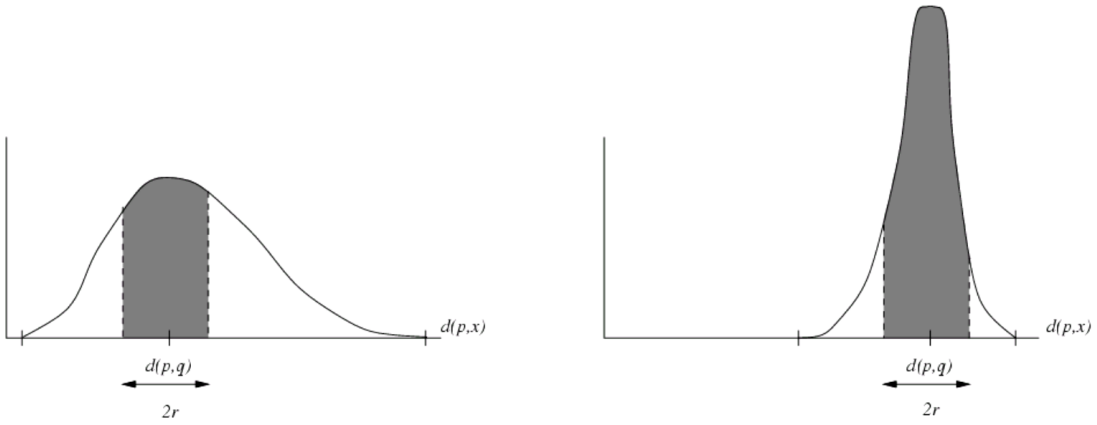


Figure 5 - *Histogram of distances at low number of dimensions (left) and at high scale (right), showing that for high size, virtually all elements become candidates for exhaustive evaluation.*

This gives us a direct and general explanation of the problem of the so-called “curse of dimensionality”. Many authors insist on an extreme case that is a good illustration: a distance such that $d(x, x) = 0$ and $d(x, y) = 1$ for all $x \neq y$. In this way, we do not obtain any information from a comparison except that the considered element is whether or not our query. It is clear that we cannot avoid, in this case, a sequential search. The histogram of distances is totally concentrated at its maximum value. The problem of intrinsic dimensionality is also discussed in [59].

3. Taxonomy of Searching Algorithms: K-Pole as a case of study

In this chapter, we first organize all known approaches in a taxonomy. This will help us to identify the essential characteristics of all existing techniques, in order to find possible combinations of algorithms not yet known so far, and find out what are the most promising areas for optimization. First, we will realize that almost all algorithms are based on an equivalence relation. The intention is to group the elements of the set in *cluster*, so that the partitions are as local as possible and have good discriminating power. In practice, it is out that many of the algorithms used to construct the equivalence relations are based on obtaining, for each element, up to k values (also called coordinates), so that the equivalence classes can be regarded as points of a k-dimensional vector space. Many of these algorithms obtain the k values as the distances between the object and k different (and possibly independent) pivots. For this reason they are called *pivoting* algorithms. The algorithms differ in their pivot selection method, in when the selection is made, and how much information is used for comparisons. Figure 11 summarizes methods classification based on their most important features.

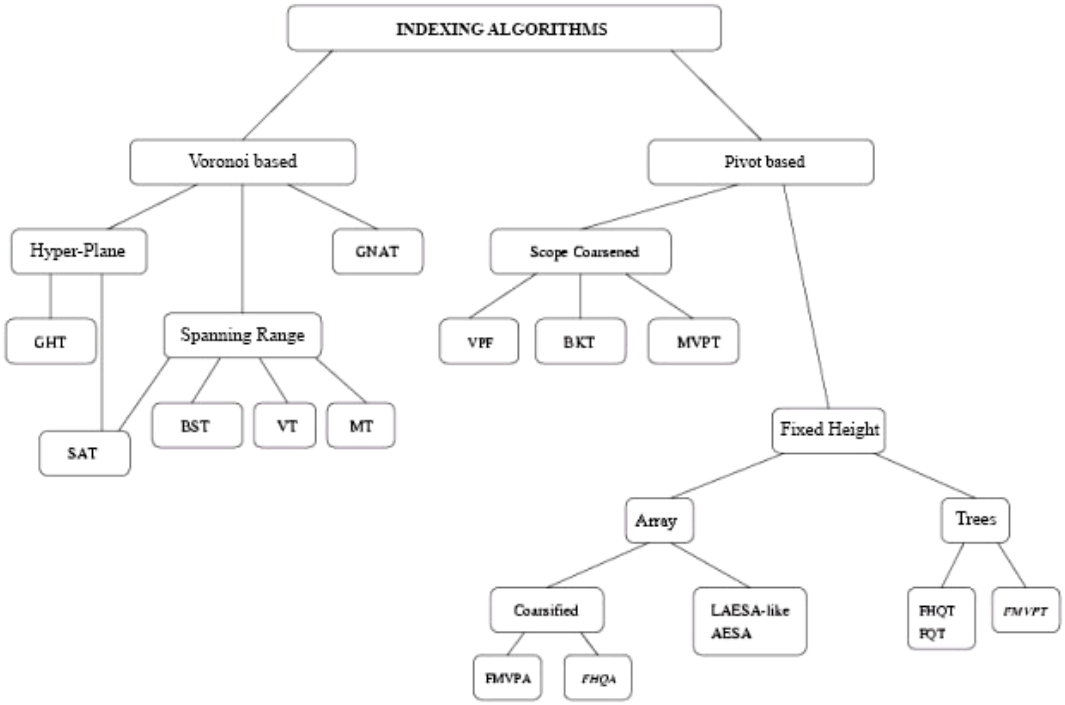


Figure 6 - *Taxonomy of existing algorithms. Methods written in italics are combinations of already existing ones.*

3.1 The Relation of Equivalence of Pivots

Many of the existing algorithms are built on this equivalence relation. The distances between an element and a number of preselected pivot (e.g. elements of the universe, also known in the literature as *vantage points*, keys, queries, etc..) are considered. The equivalence relation is defined in terms of distances between the elements and pivots, so two elements are equivalent when they are at the same distance from a pivot. If we consider a pivot p, then this equivalence relation is

$x_p \approx_p y \Leftrightarrow d(x, p) = d(y, p)$.

The equivalence classes correspond to the intuitive notion of a family of spherical shells centered at p. Points that fall within the same sphere are those equivalent to or indistinguishable from the point of view of p. The equivalence relation above is easily generalized to *k pivots* or *reference points* p_i to obtain

$x \approx_{\{p\}} y \Leftrightarrow \forall\, \imath,\ d(x, p_i) = d(y, p_i)$

and in the general case a graphical representation of the partition corresponds on intersection of different spheres centered at the points p_i. Alternatively, we can consider the equivalence relation as a projection of the R^k vector space, being k the number of used pivot. The i-th coordinate of an element will be the distance between the element and the i-th pivot. Once that is done, we can identify points in R^k with elements in the original space with the L_∞ distance. The indexing algorithm then consist in searching the set of equivalence classes such that they fall within the search radius. A third way to see the technique, less formal but more intuitive, is the following: to check whether an item $u \in U$ belongs to the query result, we prove a number of random pivot p_i. If, for each of these p_i, we have $|d(q, p_i) - d(u, p_i)| > r$, then the triangle inequality assures us that $d(q, x) > r$ without the need for calculate the values of $d(q, p_i)$. The only u elements that cannot be discarded through the use of *pivot* will be the only ones compared directly with the query object q.

3.2 Selecting Pivots

We must find the optimal number for the k pivot. If k is too small, then finding the classes will be inexpensive, but the partition will be very coarse and probably not local, and we will pay a high price during the exhaustive search. If k is too large then the partitions will be less expensive to traverse, but the cost for calculate them will be high. The number of pivot we need to obtain a good partition is linked to the intrinsic dimension of the data set. In [33] is formally proved that if the size is constant, then after selecting a proper constant number k of pivot, exhaustive search costs $O(1)$ (but their theorems don't show how to select the pivots). In AESA [55], we show empirically that $O(1)$ pivot are required for this result, then we will have a total $O(1)$ search time (their algorithm is not practical). On the other hand, in [6], it is shown that $O(\log n)$ pivots are needed to have a comprehensive final cost of $O(\log n)$. This difference is due to different models of the space structure (e.g. finite volume) and statistical behavior of the distance function. The correct answer probably depends on the considered particular metric space. A related topic is how to select the pivot. All current schemes select the pivots randomly from the set of objects U. This is done for simplicity, although addressing this issue could have notable improvements. For example, in [51] is recommended to select the pivots outside the cluster, while in [5] is suggested to use a pivot for each cluster. All authors are agreed that the pivot should be

away from each other, and this is obvious since pivot close to each other would lead almost the same information. On the other hand, pivot randomly selected are probably far enough in a high dimensional space. To discriminate a compact set of candidates, a good idea is to select a pivot between these same candidates. This makes it more likely to select an element next to them (ideally select the centroid). In this case, distances tend to be few. For example, LAESA[42] does not use pivots in fixed order, but the next to be selected will be the one with minimum distance L_1 from the current candidates.

3.3 Search Algorithms

After determining the equivalence relation to use (e.g. k pivot), the set undergoes a preprocessing where there are stored, for each element of U, its k coordinates (e.g. the distances from the k pivots). It involves a preprocessing of cost $O(kn)$, both spatial and temporal. The index can be seen as a table of n rows and k columns as shown in the left part of Figure 7.

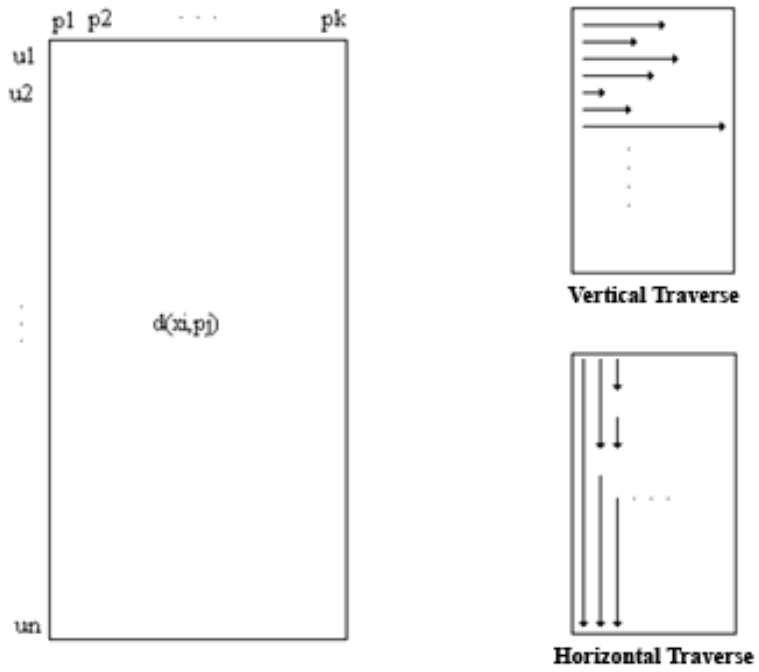


Figure 7 - *Schematic view of a pivot based indexing, in vertical and horizontal traverse*

When query is executed, it is first compared with the k pivots, thereby obtaining its k coordinates (y_1, ..., y_k) in the target area. It corresponds to determine which equivalence class the query belongs. The cost of these k distances evaluation corresponds to the

complexity of the search. It must be then determined, in the target space, which equivalence classes may be relevant to the query (e.g. which are at a distance *r* or less in the metric L_∞, which corresponds to the distance *D*). This does not involve additional distance evaluations, but produces an extra cost of CPU time. Finally, the elements that belong to the qualifying classes (e.g. those who cannot be discarded after considering the *k* pivot) are directly compared with q (external complexity). The simplest search algorithm proceeds line by line: it considers each item in the collection (e.g. each row (x_1, ..., x_k) of the table) and sees if the triangle inequality allows to discard that row, i.e. if $\max_{i=1..k}\{|x_i - y_i|\} > r$. For each row that is not rejected by applying this rule, the elements are directly compared with *q*. Although this type of crossing does not perform most of the necessary evaluations of *d*, is not the best choice. First, note that the amount of CPU work is O(*kn*) in the worst case. However, if we leave a line when we find there is a difference bigger than *r* scanning the coordinates, the average case is closer to *O*(*n*). The first improvement consists in processing the whole set column by column. Therefore, we compare the query with the first pivot p_1. Now, consider the first column of the table and discard all the elements that satisfy $|x_1 - y_1| > r$. Then consider the second pivot p_2 and repeat the procedure only on not previously discarded items. An algorithm that implements this idea is the LAESA. It is not difficult to see that the number of distance evaluations and the total CPU work remain the same as the line by line case. Anyway, now we can do better because each column can be sorted so that the interval of the qualifying rows can be traversed in binary mode instead of sequential mode[45, 23]. This is possible because we are interested, at column *i*, in values [$y_i - r$, $y_i + r$]. This is not the only allowed “trick” from a column for column evaluation (which cannot be done in the line by line case). A very important trick is that it is not necessary to consider all the *k* coordinates. As soon as the set of remaining candidates is small enough, we can stop considering the remaining coordinates and directly verify the candidates using the distance *d*. This point is difficult to determine in advance: despite the (few) existing theoretical results [33, 6], usually the application cannot be understood well enough to predict the optimal number of pivots (e.g. the point where it is better to switch to the comprehensive assessment). Another trick that can be used with the column for column evaluation is that the selection of the pivot can be done "on the fly" rather than before. In this way, once you select the first pivot p_1 and discard all the not interesting elements, the second pivot p_2 may depend on which has been the result of p_1. However, for all potential pivots, we must preserve the coordinates of all

the elements with respect to this pivot. In this way, we select k potential pivots and operate a preprocessing work on the table like before, but now we can choose in which order to use the pivots (according to the state current research) and where to stop the using of pivots and go to compare directly. An extreme case of this idea is AESA, where $k = n$, i.e. all the elements are potential pivots, and at each iteration the new pivot is randomly selected among the remaining elements. Despite its inapplicability in practice because of the time (and space) preprocessing of order $O(n^2)$ (because distances between all the known elements are precomputed), the algorithm calculates a surprisingly low number of distances, even better when the pivot is fixed at first. This shows that it is a good idea to select the pivots from the current set of candidates. Finally, we note that instead of a sequential search in an mapped area, we could use a search algorithm for k-dimensional vector spaces (e.g. *kd-trees* or *R-trees*). According to their ability to handle large values of k, we may be able to use more pivots without significantly increasing the CPU cost. It should be remembered also that, when many pivots are used, research facilities working worst for vector spaces. This is a very interesting problem that has not been studied further, that considers the balance between distance computations and CPU time.

3.4 Comparisons between existing Data Structures

Comparisons performed on various existing data structures show that, using the same amount of memory, BKT is the most efficient data structure, followed for higher dimensions by other similar schemes. In addition it was found that, for areas of high size, techniques based on Voronoi partitions [3, 58] have proved more efficient than those based on pivots. On the other hand, FHQA, FMVPA and LAESA have the characteristic of improving their performance by using multiple pivot. LAESA requires an amount of memory 128 times greater and FHQA and FMVPA require 32 times more memory than the LAESA amount, in order to be capable of beating BKT in 20 dimensions spaces. It is interesting to note that, with increasing size, pivoting algorithms need more pivots to beat the algorithms based on Voronoi partitions. On the other hand, there are no good methods that allow Voronoi-like algorithms to use more memory to increase their efficiency, but if there were, they likely would require less space to obtain the same results of other algorithms in spaces of higher dimensions. Note also that only the implementations for arrays of FHQT and FMVPT make possible

in practice the amounts needed for the calculations. We believe that a data structure that combines the idea of Voronoi partitions with an k-pivoting algorithm may represent the best compromise.

3.5 Considerations and Open Problems

Many of the existing algorithms are in fact based on variations of a few ideas, and identifying them, have appeared combinations not known before, and some of they were particularly efficient. We can summarize the path showed so far with the following statements:

1. The factors which affect the efficiency of a search algorithm are the intrinsic dimension of space and recovered proportion of the whole set (in percent).

2. We have identified the use of equivalence relations as the common ground that underlies all the indexing algorithms, and classified the search cost in terms of internal and external complexity.

3. A broad class of search algorithms depend on the selection of k pivots and the mapping of the metric space on R^k, using the distance L_∞. Another important class of algorithms use the Voronoi partitions.

4. Although there is an optimum number of pivots to use, this number is too high in terms of space requirements. Therefore a compromise between the number of pivots and requirements the system.

5. Algorithms based on Voronoi partitions are more resistant to the increase of the intrinsic size of the space.

6. Among the considered structures, experimental results show that the best one is the version of BKT adapted to continuous spaces of arity 2. This structure "degenerates" into a very efficient clustering scheme. If the availability of memory increase, at the end other structures will be better of BKT. Among these, those which best use memory are FHQA and FMVPA.

But there are a number of open issues that require further attention. The main are listed below:

- Working more on the clustering schemes in order to designing new algorithms, finding new ways to reduce construction time (which are extremely high in almost all practical cases) and allow the use of more memory so as to reduce the search time.

- Finding good hybrid between clustering and pivoting algorithms. The first act better with higher dimensions while seconds improve when more memory is available. After that the space is clustered, intrinsic size of the cluster is low, so a to top-level clustering structure combined with a pivoting scheme for the clusters represents an interesting alternative. Pivots could be selected starting from the cluster since it has been made compact in space.

- Better understanding of how the space structure affects the efficiency of search algorithms. Parameters such as the distance histogram allow us to find bounds to the worst or the average case of the complexity of the problem.

- Better understanding of the effect of pivot selection, conceiving methods to choose *effective* pivot. The problem of the appropriate number of pivots and its relation to the intrinsic size of the space plays a key role here.

- Taking into account the extra CPU complexity, which we have already considered. In some applications, distance is not so expensive and then other CPU costs can be considered. The use of research facilities specialized in the mapped space (especially R^k) and the resultant compromise in the complexity deserves more attention. Another area concerns inputs / outputs costs.

- Focusing on finding the *nearest neighbor*. Many current algorithms are based on the range query to solve this problem, and despite the existing heuristics are difficult to improve, there may be other ways to address the problem regardless of the range query.

- Considering approximate and probabilistic algorithms, which may give better results at a cost, especially for this problem, it seems acceptable.

3.6 The Antipole Tree

The Antipole Tree [14] belongs to the class of "bisector trees" [73, 69, 95], which are binary trees where each node represents a set of elements to cluster. The process begins by partitioning the input set according to proximity of the elements with two centers c_1

and c_2 chosen randomly. Partitions are recursively treated as new sets of input. The process continues until there are no more elements to be treated. Each node of the tree contains the centers and radii of the respective subset of elements. In order to produce a good pair of (c_1, c_2) elements, a simple randomized tournament among the elements of the input is used, related to the one described in [65]. The tournament is played as follows: in each game, the winners of the previous game are randomly partitioned into subsets of a fixed t size. Then a procedure finds the "1 – median"[2] and discards it. Matches are played up to less than $2t$ elements remain. The pair of further points in all the remaining elements is the Antipole (antipodes) pair of elements.

There are three main ideas from which Antipole tree draws its inspiration:

a) The FQ-tree [5] organizes the elements of a metric space among the leaves of a tree data structure. Slightly abstracting from its published description, an FQ-tree consists of an array of reference objects r_1, ..., r_k and a distance vector v_o associated with each object o such that $v_o[i] = dist\ (o, r_i)$. Given a query object q, the distance to each object reference is calculated, thus obtaining a v_o. The object o cannot be closer than a threshold distance t from q if, for each i, $v_q[i] > v_o[i] + t$. Therefore, even if o is closer to the query than r_i, q cannot be at a distance from or that is less than t. In Antipole Tree a similar idea is used except for the fact that our reference objects are the centroids of the clusters.

b) The M-tree [26, 76] is a dynamically balanced tree. The nodes of M-tree retain various elements of the collection doing so they are "neighbors" and "not too many." If any of these conditions is violated, the node is split and a subtree that originates in the node is recursively created. In an M-tree to each parent node corresponds a cluster with a certain radius and to each child node corresponds a sub-cluster with a smaller radius. If a centroid x is located at a distance $dist\ (x, q)$ from the query and the radius of the cluster is r, then the entire cluster corresponding to r can be discarded if $dist\ (x, q) > t + r$. The Antipole Tree takes from the M-tree the feature to make dynamic updates and the idea that a parent node corresponds to a cluster and its children nodes represent sub-cluster of the parent node. The main difference resides in construction methods and in the fact that the clusters in the M-tree should contain a limited number of elements.

[2] It is recalled that the 1-median of a set of points in a metric space S is the element of S whose average distance from all the other elements of S is minimized.

Also search strategies are different since the Antipole algorithm produces a binary tree data structure.

c1) The VP-tree [54, 59] organize the elements that come from a metric space into a binary tree. The elements are stored both in the leaves and the internal nodes. The elements stored in the internal nodes are called vantage points. Processing a query requires the calculation of the distance between the queries, and some of the vantage points. The construction of a VP-tree partitions a dataset according to the distances between the elements and reference points. The average value of these distances is used as a separator to partition the objects into two balanced subsets (those closer to or equal to the mean value and those more distant from it). The same procedure is applied recursively to each of the two subsets.

c2) The MultiVantage-Point tree [66] is an intellectual descendant of the vantage point tree and the GNAT structure [16] and appears to be superior to the earlier methods. The main idea is that, given a point p, we can partition all the elements in m partitions based on their distances from p, where the first partition consists of those points whose distance is greater than d_1 and less than or equal to d_2, etc.. Given two points p_a and p_b, partitions $a_1 \ldots a_m$ based on p_a and partitions $b_1 \ldots b_m$ based on p_b a1 ... am are created. All the possible a-partitions and b-partitions can intersect to obtain m^2 partitions. In a *MVP-tree*, each node in the tree corresponds to two objects (*vantage points*) and m^2 children, where m is a parameter of the construction algorithm and each child corresponds to a partition. When searching for objects that are within distance t respect to the query q, the algorithm proceeds as follows: given a parent node with vantage points p_a and p_b respectively, if any partition Z has the property that for each object $z \in Z$, $dist(z, p_a) < d_z$ and $dist(q, p_b) < d_z + t$, then Z can be discarded. There are other reasons for rejecting the clusters, also based on the triangle inequality. The use of multiple vantage points, together with the preprocessing of distances calculations reduces the number of distances evaluation at query time. As the MVP-tree, the Antipole Tree does an aggressive use of the triangle inequality. Other sources of inspiration include [19, 79, 75, 85, 50, 51, 92].

3.7 Calculation of Diameter (Antipole)

Let (M, d) a metric space with distance function d: $(M \times M) \rightarrow$ R. The *problem of calculating the diameter* or the *problem of the farthest pair* consists in finding a pair of points A, B in S such that $d(A, B) \geq d(x, y)$, $\forall\ x, y$ in S. As reported in [90], one can construct a metric space where the distances between the objects are all set to 1 except for one (randomly chosen), which is set to 2. In this case each algorithm that attempts to provide an accuracy factor greater than ½ shall examine all pairs, so a randomized algorithm does not necessarily find the pair. Nevertheless, we expect a good result in almost all cases. Here is reported a *randomized* algorithm inspired from that proposed for the calculation of the 1-median [70]. In this case each subset X_i is processed locally through a LOCAL_WINNER procedure that computes its exact 1-median x_i and returns the set $*X* = Xi \setminus \{x_i\}$. The elements obtained from $X_1 \cup X_2 \ldots X_k$ are used for the next step. The tournament ends when we obtain a single set X, from which we extract the final winners A, B. The winning pair, called *Antipole pair*, represents the approximate diameter of S. One possible implementation of the general method discussed above is to partition the elements in each game into subsets of the same dimension t, possibly with the exception of a single subset whose size must be between t and $2t - 1$. In addition, we can assume that the iteration stops when the number of elements fall below a certain given threshold. Finally, quadratic "brute force" calculation returns the exact diameter of the whole residue set.

3.8 How Antipole Tree works

Let $(S, dist)$ a finite metric space and suppose that the intention is to divide it into the minimum possible number of clusters whose radius should not exceed a given threshold σ.

This problem was studied by Hochbaum and Maass [89] for the Euclidean spaces. Their approximation algorithm has been improved by Gonzalez [86]. Similar ideas were used by Feder and Greene [81] (see [96] for an extensive study on clustering methods in Euclidean spaces).The clustering via Antipole of radius limited to σ is done by a top-down recursive procedure that begins from a finite set of points S and controls at each iteration if a particular splitting condition Φ is met. If Φ is not satisfied, no division is performed, current subset is a cluster and a centroid whose approximately distance from

any other element in the cluster is less than σ is calculated. Centroid calculation is performed using the following *tournament*: during each game, the winners of the previous game are randomly partitioned into subsets of fixed size t and only the 1-median of each subset will participate in the next game. The tournament ends when less than $2t$ elements remains. At this point the centroid we needed is just the1-median of the remaining winners. The centroid is stored, together with its corresponding radius, in the same leaf. Otherwise if Φ is satisfied, a couple of points $\{A, B\}$ of S, called *Antipole pair*, is created , which is used for partitioning S into two subsets S_A and S_B (with $A \in S_A$ and $B \in S_B$), obtained by assigning each point of S to the subset containing its nearest reference point between the two of the *Antipole pair* $\{A, B\}$.

3.9 Antipole Tree in generic metric spaces

The Antipole Tree data structure acts on a metric space (X, *dist*), where *dist* is the distance metric. The elements of the metric space are inserted into a data type called *object*. An *object O* in the *Antipole Tree* contains the following informations: an element x, an *array* D_V that stores the distances among D_V and all its ancestors (*Antipole pairs*) in the tree and a variable D_C containing the distance between x and the cluster centroid C. A set S of data is a collection of *objects*. Each cluster is a set of objects in which are stored the following information:

- The centroid, called C, which is the element that minimizes the sum of the distances to other objects;
- The radius, *Radius*, containing the distance to the farthest object from C;
- The list of items, *CList*, which stores the catalogue of the objects in the *cluster*;· The number of objects, Size, in the cluster.

The Antipole Tree has internal nodes and leaf nodes.

- An internal node stores (i) the identity of two Antipole objects A and B, called *Antipole pair*, with distance at most 2σ, (ii) the radii *RadA* and *RadB* of the two subsets (S_A and S_B respectively obtained dividing S according to the proximity of objects from A or B, respectively), and (iii) the pointers to the left and right subtrees, respectively called *left* and *right*;
- A leaf node stores a *cluster*.

3.10 Construction of the Tree

To construct such a data structure, the procedure BUILD takes as input the dataset *S*, the radius σ of the cluster and a set *Q*, initially empty. The algorithm starts by checking if *Q* is empty. If so, the procedure ADAPTED_APPROX_ANTIPOLE, which returns the *Antipole pair*, is invoked. The pair is then inserted in Q. This algorithm works as the one presented above, with the only difference that as soon as a pair with distance greater than 2σ is generated, the algorithm stops and returns it. The next step is to check if the *splitting condition* is verified. In positive case, the *S* is divided into S_A and S_B according to the distances from the *Antipole pair*. Otherwise a cluster is generated, using the procedure MAKE_CLUSTER: it calculates the centroid of the cluster using the randomized algorithm APPROX_1_MEDIAN, after which for each object *O*, the distance between it and the centroid *C* of the cluster is calculated. The data structure produced by BUILD is a binary tree whose leaves contain a set of cluster with approximate centroids whose radii are less than σ.

3.11 The K-Pole Tree

The K-Pole Tree is a generalization of the Antipole Tree, designed to use an arbitrary number *k* of *pivots*. This leads to some extent re-engineering the approach, providing new solutions for the implementation of the data structure, the for selecting the *pivots*, the tree building procedure and the routines for absolving queries. In the current literature are not a few examples of search algorithms that are extension of a previous idea: for example, the Multi-Vantage-Point Tree whose structure is a *m*-ary tree which extends the binary tree on which is based the Vantage-Point Tree, or the *GNAT*, which is a *n*-ary extension of the binary *GHT*.

Why *k* instead of 2? In section 3.2 we clearly refer to the fact that there is an optimum number of *pivot*, and that these should be chosen according to specific criteria. The *K-Pole Tree* was developed to make more complete the approach of the Antipole Tree (which is based on the selection of a pair of pivot, the distance of which approximates the whole diameter), by selecting *k pivot* to approximate the *k* centers of the set. The objective is to obtain some partitions (and consequently some equivalence classes, which at the end of the tree construction process are other than our cluster) more local possible, in order to dramatically reduce the number of distances calculated during the

search process. It is feasible only if the *k pivot* chosen by the selection algorithm are quite different from each other. In metric spaces we can consider *k* points quite different from each other when they are *well distributed* and sufficiently *distant* from each other. For example, if our set approximates a cube (i.e. is composed of points evenly distributed in some interval of a three-dimensional space), then the optimal choice of 8 *pivot* could be the following: we decompose the cube into 8 equal sub-cubes (Figure 8) and each pivot is located at the center of each sub-cube. If we consider each pivot as placed in the centers of convenient spheres all of equal radius *r*, we realize that each one captures a well-certain portion of space, local enough, well-distributed and different from the portion of space captured by any other of the remaining 7 pivot. This is what it takes to produce a good search algorithm. Conversely, the choice of close pivot is far from being considered satisfactory, due to the low diversity of information that each pivot lead than the other.

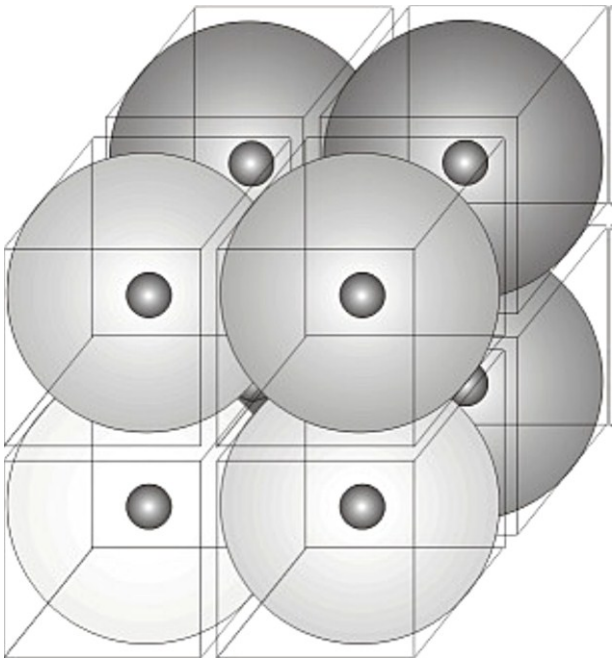


Figure 8 - *Optimal selection of 8 pivots in a uniform cubic space*

3.12 The calculation of the k centers of a set (*K-Pole*)

The *K-Pole Tree* provides several criteria for selecting the pivot, based on approximation algorithms that, for a given *k*, calculate the *k* centers in a set. The

problem of finding the best possible position for networks of calculators, or vertices in graphs, abound in common situations. One of the most famous problems of this type is called the *k-centers problem*, where given *n* cities and the distances between all pairs of cities, the goal is to choose *k* cities (called *centers*) such that the maximum distance between a city and the closest *center* is minimal. More formally, the problem of k-centers vertex can be defined as follows: let $G = (V, E)$ a not direct complete graph, whose arcs costs satisfy the triangle inequality, and let *k* a positive integer less than or equal to $|V|$. For each set $S \subseteq V$ and each vertex $v \in V$, we define $d(v, S)$ as the length of the shortest arc (i.e. the less expensive) from *v* to each vertex of S. The problem then is to find that particular $S \subseteq V$ that minimizes max $d(v, S)$. The problem of k-centers is NP-*hard*. [82]. One popular way to solve this problem consist in solving a series of *set coverage problems* [67, 78, 100, 101]. At each step a threshold for the distance of coverage is chosen and it is checked that all vertices can be *covered* (i.e. reached) not exceeding this distance, using at most *k* centers; if so the threshold decreases, otherwise it grows up (you can also use the *set domination problem* rather than the coverage one [97]). For example, Minieka [101] has solved the problem of the k centers as a series of set coverage problems.

More elaborate versions of this approach were presented by Daskin [67, 68], where it was used also the problem of maximum coverage, Ellumni [78], and Ilhan [100], which have implemented more efficient definitions of the problem. Usually these set cover problems have been solved by *entire* programming. Another way to solve the problem of *k* centers has been given recently by Mladenovic [103], where tabu search, neighborhood search and various other greedy methods were used. Other methods of this latter type were also applied by Gonzalez [85], Hochbaum and Shmoys [98], and Shmoys [104].

Below are details of two approximate solutions of the problem of calculation of k centers of a set: the algorithm of Gonzalez and the algorithm K-Median [106] (the latter was conducted by the University of Catania), both algorithms are currently used by K-Pole Tree in order to create the *index* data structure. Note that we could even operate a totally random selection of the *pivots*, but this choice would lead to a seriously detrimental indexing. It is rather a lot more efficient if it relies on a robust algorithm for the approximate calculation of the *k* centers of the set.

3.13 Gonzalez algorithm

Gonzalez algorithm [85] (or the farthest point clustering) is one of the known fastest algorithms for the approximate determination of the k centers in a finite metric space. Let (S, d) a metric space and k is the number of items you want to determine. Shown in Algorithm 1 is a procedure in pseudo-code that summarizes the main steps:

GONZALEZ (S, k)

Let C_1 an arbitrary point in S.

for $2 \leq i \leq$ k

for each p **in** S, let $f(p) = d(p, closest)$, such that closest is the point, among the C_i found so far, closest to the given point, i.e. $d(p, closest)$ is minimal.

end for each

C_i = **max**($f(p)$)

end for

Algorithm 1 - *How Gonzalez Algorithm works*

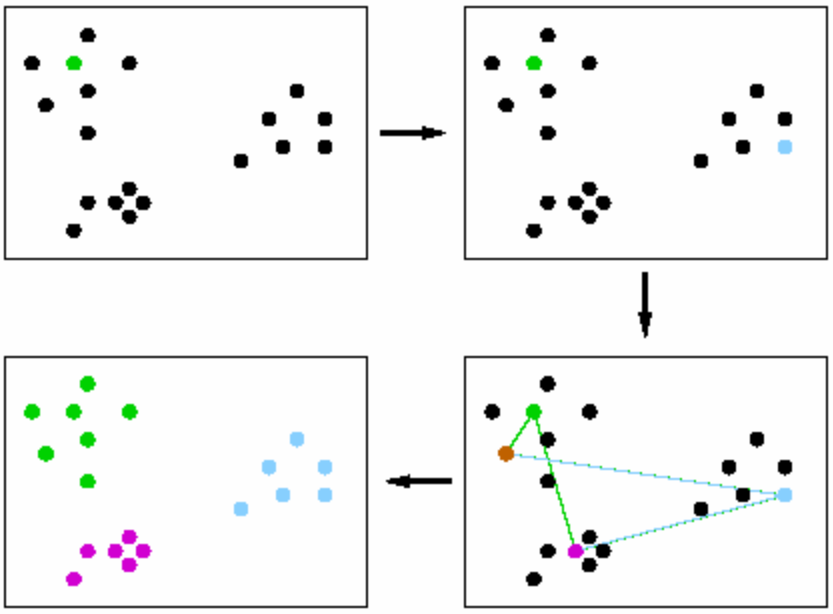


Figure 9 - *Graphical representation of the Gonzalez Algorithm*

After choosing the first point at random, the algorithm determines at each step the farthest object from all those chosen so far. At the next step, the just selected object will be part of the final set of the centers, and new calculations will take account of this new object.

The Algorithm of Gonzalez has the following properties:

- The execution time is $O(n)$ for each iteration: so, in order to find k centers you will have a time complexity of order $O(kn)$.
- The radius of the produced centers is at most double that best possible.
- The diameter of the produced centers is at most double that best possible.
- It produces the centers one by one, successively.
- It works for any distance function that is a metric.
- Returns always exactly k objects (unless the starting set is formed from less than k elements).

Experimentally it is seen that the algorithm of Gonzalez produces in most cases k objects quite different from each other, which can be considered a good approximation of the k centers of the set. This algorithm has been chosen as method for selecting the *pivots* for the K-Tree Pole since, though it is not a purely greedy algorithm, it finds a good approximation of the k centers of the set with a cost of only $O(kn)$ iterations. Other algorithms, like that of Shmoys [104], or that of Hochbaum and Shmoys [98], cost respectively $O(kn^3)$ and $O(kn^5)$ in terms of iterations.

3.13.1 Possible optimizations of Gonzalez Algorithm

A possible improvement to the Gonzalez algorithm comes from the idea of choosing the first point following a certain criterion, rather than choose it essentially at random. In particular, the step of optimization consists in matching the first point chosen by the algorithm with the 1-median of the set. Once the first point is obtained in this way, we proceed as in the original algorithm. Experimental studies [105] confirm that this version of the algorithm of Gonzalez possesses a better degree of approximation than the original. It is clear, however, that the cost of the calculation of the 1-median affects the implementation of the algorithm itself, producing a worse performance in terms of space-time.

3.14 The Data Structure

In order to treat a variable quantity *k* of *pivots*, some of the construction parameters of the tree have been reviewed. The substantive changes mainly relate to the structure of pointers of the tree and the mode of storing information in the nodes of the tree itself, in order to reflect the new implementation. The data structures used to store the objects and clusters have not detailed changes (Figure 10). Here are the details of the salient changes.

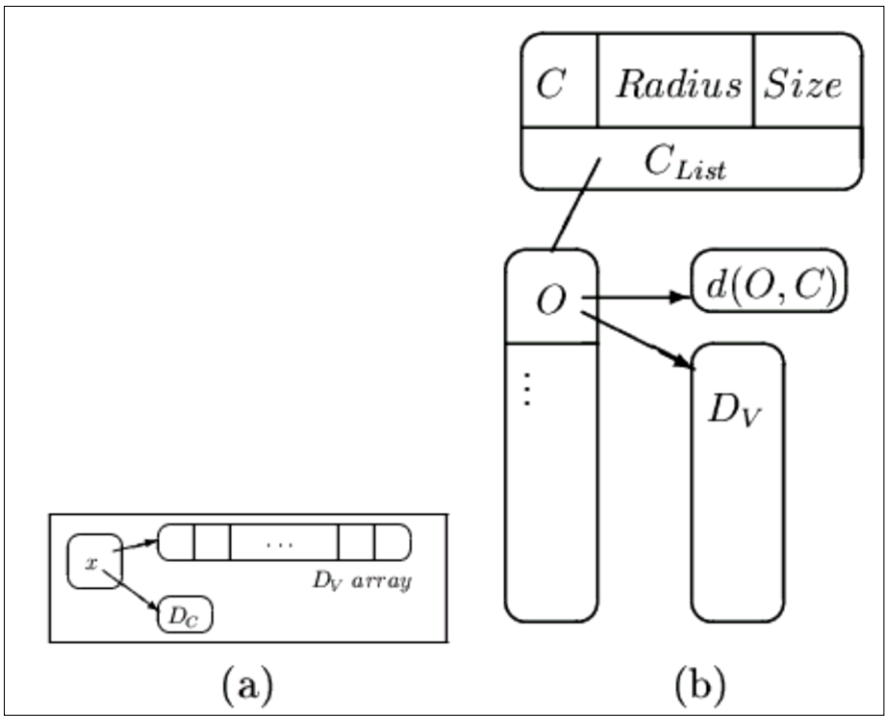


Figure 10 - *(a) Object Data Structure, (b) Cluster Data Structure*

3.14.1 From Binary Tree to k-ary Tree

The indexing algorithm on which the Antipole Tree relies, provides the building of a binary tree (Figure 11), in which each node permits to identify, thanks to two pointers, the left child and the right child respectively. To each of them one of the pivot resulting from the calculation of *Antipole pair* is associated.

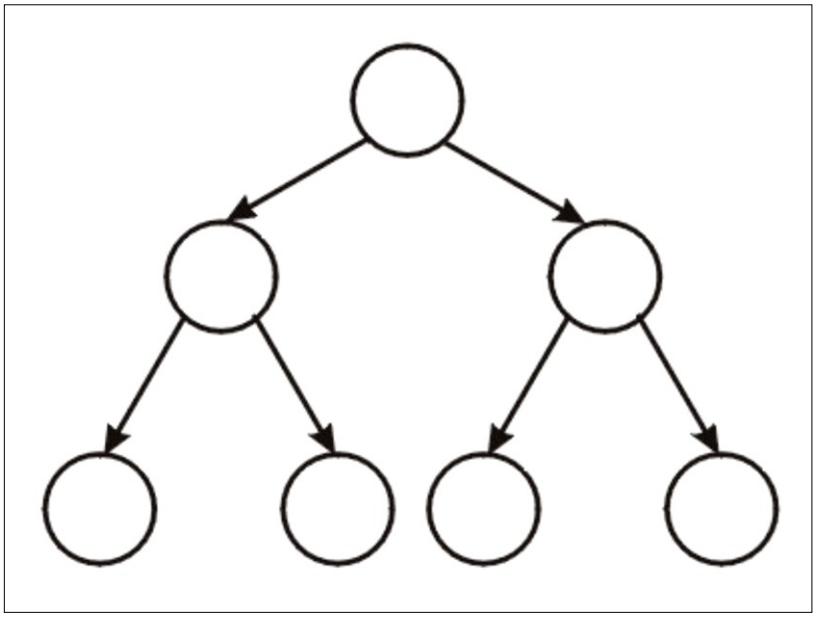


Figure 11 - *Example of Binary Tree*

In the case of K-Tree Pole, in order to treat any number *k* of pivots, the data structure "Tree_Node" of the original Antipole Tree was modified in the following way: each tree node maintains two pointers, one to the first of his children (*leftchild* pointer), the other to his brother (*leftbrother* pointer). In this way is possible to create a tree structure where each node can have an arbitrary number of children (and brothers): from zero (in the case of the *cluster*) until to *k* ([3]). Wanting to treat the subject from a purely technical point of view, we can say that the structure is still a binary tree, organized to function as a *k-ary* tree (Figure 12).

There are other possibilities for implementing a *k-ary* tree, which mainly involve the use of arrays: these appear to be limited by the fact that the array has, however, a limited (and fixed) number of elements and are also difficult to manage. The implementation used for the K-Tree Pole is simple but very powerful.

[3] The number of children for each node cannot in any way be greater than *k*. This is due to the algorithm for selecting the *pivots*, which returns each time (possibly) *k* objects of the set. Actually it can return even less: in this case the node will have a number of children less than *k*.

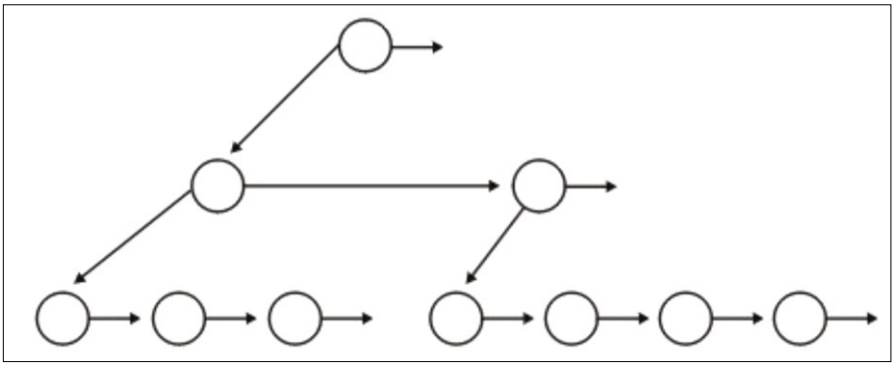


Figure 12 - *Example of k-ary Tree (note that in this particular case each node has an arbitray number of children (2,3,4) and siblings (0,1,2,3)*

3.14.2 Inner nodes and clusters

Even the inner nodes have been changed to reflect the new implementation: first, we cannot store a fixed number for the *pivot*, and then instead of the fields *A* and *B* of the *Antipole Pair*, we have introduced a linked list of pointers to objects, *K-Pole_Seq*, which stores the sequence of *k* (or less) objects which identify the K-Pole of the set. In parallel, in place of fields *RadA* and *RadB* has been introduced a linked list of real numbers, *Radius_Seq*, which stores the sequence of *k* (or less) radii of the subsets in which the set is divided, according to the proximity of objects to the members of the K-Pole. 4[4]

Finally, the pointers to the left and right subtrees have been replaced by *leftchild* (for the first child) and *leftbrother* (for the first sibling) pointers. A schematic representation of an inner node is offered in Figure 13.

[4] Note that each radius is the distance between the *K-Pole* member and the farthest object from it in the subset where the K-Pole member is the representative.

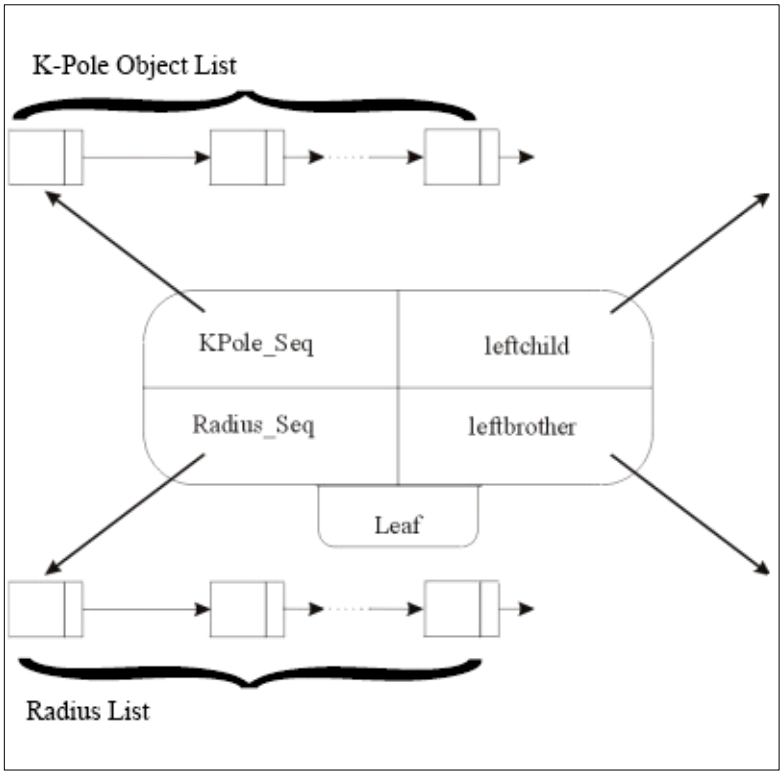


Figure 13 - *Structure of an inner node of the K-Pole Tree*

Clusters have not been much influenced from new implementation, therefore it will store the *centroid* of the *cluster*, *Centroid*, the catalog of objects in the *cluster*, *MList*, the radius *Radius*, and finally the number of elements, *Size*. (Figure 10 b). For each node an attribute that allows to distinguish an internal node to a leaf, called *Leaf*, is used. In fact, only the leaf nodes will have an associated cluster (*Leaf* set to *TRUE*), whereas this is not true for the inner nodes (*Leaf* set to *FALSE*).

3.14.3 The construction of the Tree

To construct such a data structure, the procedure BUILD takes as input the dataset S, the radius σ of the cluster, the number *k* of the pivots and the metric *m*[5]. In addition, locally to the procedure, the set *Q*, initially empty, is of particular importance. It is designed to contain the objects of K-Pole. The algorithm begins by recalling the procedure for the selection of *k pivots*, which will be included in Q. After that, the

[5] The K-Pole Tree works with any distance function that is a metric. In particular, in its current implementation, it is possible to choose from a variety of distance functions belonging to the family of Minkowski metrics [2].

splitting condition is checked, which in our case is equivalent to the fact that Q is empty or not. Q can be empty in the following cases:

- the cardinality of the input is less than k.
- the subset has a radius less than σ.
- the diameter of the set of the found k pivots is less than σ.

In all cases discussed above a cluster is created from the set of input, using the procedure MAKE_CLUSTER. It calculates the *centroid* of the *cluster* via the randomized algorithm APPROX_1_MEDIAN, after which for each object *O* the distance between it and the *centroid C* of the cluster is calculated. Otherwise, if Q is not empty (i.e. contains at least two elements, *k* at most[6]), we partition the set. This is done by the function MAKE_SPLIT_SET, which in addition to the task of partitioning the set is concerned with updating D_v fields of the objects and the radii of the subsets identified by the partitioning process. The set *S* is thus divided into (S_1 ... S_K) according to distances from the pivot *k*. Finally, BUILD procedure is recursively called on each subset identified by MAKE_SPLIT_SET, starting from the left child and then continuing to allocate the various brothers of the left child. It is important to note the fact that when we call the BUILD procedure on various brothers of the first child node, we must somehow scroll through the list of the brothers until we find a *free* pointer (i.e. we reach the last allocated sibling). This situation is managed by the REACH_LAST_BROS procedure, which returns the pointer to the last non-null sibling in the list of siblings. The data structure produced by BUILD is a *k-ary* tree whose leaves contain a set of cluster with approximate *centroids*, whose radii are bounded by σ.

Below is the BUILD procedure in pseudo-code:

BUILD (*S*, σ, *Q*, *k*, *m*)
Q = K-POLE_DETECT (*S*, σ, *k*, *m*);
if *Q* = ∅ **then** / / Splitting Condition fails
    T.Leaf ← *TRUE*;

[6] It never happens that *Q* contains only one element because in that case we cannot calculate the diameter.

T.Cluster ← MAKE_CLUSTER (*S*);

return *T*;

end if;

{ P_1 ... P_K } ← *Q*;

T.K-Pole_Seq ← { P_1 ... P_K };

for each $S_i \in \{ S_1 \ldots S_K \}$

$S_i \leftarrow \{O \in S \mid dist(O, S_i) \leq dist(O, S_m), m = 1 \ldots k\}$;

end for each;

for each $O \in S$

$O.D_v \leftarrow O.D_v \cup \{(S_1, dist(O, S_1)), \ldots, (S_k, dist(O, S_k))\}$;

end for each;

$T.Radius\_Seq \leftarrow \{(\max_{o \in S1} dist(O, S_1), \ldots, (\max_{o \in Sk} dist(O, S_k)\}$;

T.leftchild ← BUILD (S_1, σ, *Q*, *k*, *m*)

for each $S_B \in Q \setminus \{S_1\}$

Brother ← REACH_LAST_BROS (*T.leftchild*);

Brother ← BUILD (S_B, σ, *Q*, *k*, *m*)

return *T*;

END BUILD.

Algorithm 2 – *The BUILD procedure for K-Pole Tree*

3.15 Dynamic *k* cardinality evaluation

The index construction algorithm takes as input the number of clusters, *k*, which must be produced representing also the number of starting pivots. Once the dataset is partitioned into *k* subsets we recursively establish the more suitable number of pivots that are needed to index the subspace. We heuristically determine such a number in terms of the pseudo-diameter of the subset and the residual number of objects. The idea that relies behind this heuristic is that if *k* is a suitable number for splitting a dataset with a certain diameter and a number of objects, it is reasonable that to index the subspace in a highly discriminant is enough to choose a $k^* < k$. Intuitively, we can infer that a bigger value of k^* is needed for a subset with a higher diameter, or that contains

more elements. On the other hand, when dealing with subsets having a high diameter but with few elements, or with many elements in a smaller space needs less pivots to have still a good pruning gain at query time. More formally, let S be the current set, S_{diam} the pseudo-diameter, let k be the starting number of pivots. In each recursive call we empirically re-evaluate the number of cluster needed for a subset S_i with the following function:

$$k^* = \max\left(\frac{\left\lfloor \frac{S_{i_{diam}} \times k}{S_{diam}} \right\rfloor + \left\lfloor \frac{|S_i| \times k}{|S|} \right\rfloor}{2,3}\right)$$

We establish the k^* value averaging the diameter and cardinality ratios of the superset S with the one of the set that we are going to partition S_i. In this way we avoid worst cases of using only one of the fraction values, producing more balanced partitions. Moreover, our preliminary tests showed that maintaining $k^* \geq 3$ yields better building and search performances. Our preliminary experimental results, in chapter 4, show that such a heuristic provides at searching time better results than the static one. Nonetheless, our preliminary analysis demonstrates that this heuristic yields better results than using diameter ratio only (or cardinality ratio only), avoiding the worst cases related to each method, due to "unbalanced" subsets.

3.16 The Range Search Algorithm

The *range search* algorithm takes as input the K-Pole Tree T, the object query q, the threshold t, and returns the result of the *range search* on the database with threshold t. The search algorithm recursively descends all branches of the tree until it finds a leaf node that represents a *cluster* to visit or finds a subtree whose elements are certainly outside the search range and therefore can be pruned. These branches are filtered by applying the triangle inequality, which is used in this case to degree inclusion or exclusion. The first use states that an object can be excluded, avoiding the calculation of the distance between the object and the query. The second states that an object must be inserted in the result, since it is *close* to the *centroid* of its *cluster*, which in turn is located so near to the query object to fall within the range of research. At implementation level, the search algorithm consists of two procedures, RANGE_SEARCH and VISIT_CLUSTER.

3.16.1 The procedure RANGE_SEARCH

The procedure starts by checking that the node is internal. In case the node is a leaf, the control is passed to the VISIT_CLUSTER procedure, whose task is purely the visit of the *clusters*. If the node is internal, first the distances D_1, ..., D_K between the query object and elements of the *K-Pole* of the current node are calculated. Having just completed the explicit calculations of distances, we check immediately that some of them result less than the threshold t. If so the element of *K-Pole* of the node corresponding to that distance is inserted in the result of the query. At the next step the distances D_1, ..., D_K are appended to the field D_v of the query object, so that the query simulates in every way an object that has been previously treated in that branch during the process of creation of the tree. Once the field D_v has been updated, for each radius in the rays list of the current node the following property is checked:

$$D_i \leq t + T.Radius\_Seq\ [i]$$

The recursive call will take place *only* on subtrees whose associated subset has a radius that satisfies this property. On the other hand, the subtrees that do not satisfy this property are permanently pruned (and in this way the calculation of distances with the elements of *K-Pole* and *clusters* associated with them should be avoided). When all the recursive calls are executed, the distances were added to those already present in D_v object query are removed, in order not to distort further (if any) recursive descents in the tree. Finally, we note that unlike the BUILD procedure, in this case we do not need any support function to scroll properly the brothers of the left child of a node, since they were all previously allocated during the BUILD procedure.

The following is the procedure RANGE_SEARCH in pseudo-code:

RANGE_SEARCH (T, q, t, OUT)
if ($T.Leaf = FALSE$) **then**
    for each $D_i \in D_1, \ldots, D_K$
        $D_i \leftarrow dist\ (q, T.K\text{-}Pole\_Seq\ [i])$
        if ($D_i \leq t$) **then**
            $OUT \leftarrow OUT \cup \{T.K\text{-}Pole\_Seq\ [i]\}$;
        end if;

end for each;

$q.D_v \leftarrow q.D_v \cup \{D_1, \ldots, D_K\}$;

if ($D_1 \leq t + T.Radius\_Seq$ [1]) **then**

RANGE_SEARCH (*T.leftchild*, *q*, *t*, *OUT*);

end if;

for each $D_i \in D_2, \ldots, D_K$

if ($D_1 \leq$ t + *T.Radius_Seq* [*i*]) **then**

RANGE_SEARCH (T.*leftchild.leftbrother* [*i*], *q*, *t*, *OUT*);

end if;

end for each;

$q.D_v \leftarrow q.D_v \setminus \{D_1, \ldots, D_K\}$;

return;

else

OUT ← *OUT* {VISIT_CLUSTER (*T.Cluster*, *q*, *t*, *OUT*)};

end if;

END RANGE_SEARCH.

Algorithm 3 - *The RANGE_SEARCH procedure*

3.16.2 The procedure VISIT_CLUSTER

The procedure begins by calculating the distance between the query object and the centroid of the cluster. Having carried out an explicit calculation of distance, we use it for verify that the centroid of the cluster is at a distance such as to fall in the threshold *t*, and therefore should be included in the query result. Then two checks are carried out: one for determining whether the cluster can be discarded of all, the other for possibly include everything. We note that in both cases the procedure ends without having calculated any distance if not between the query object and the centroid of the cluster. Subsequently a cycle is carried out on all the elements of the Member List of clusters, in which we try to include or exclude an object from the query result based on the triangle inequality, which is supported by the use of *Dc* and *Dv* fields of the current object and the query object. Note that only in one case, within this cycle, the distance between the

object and the query object is explicitly calculated. Here is procedure VISIT_CLUSTER in pseudo-code:

```
VISIT_CLUSTER(Cluster, q , t, OUT)
 q.Dc ←dist(q, Cluster.Centroid);
 if (q.Dc ≤ t) then
        OUT ←OUT ∪{Cluster.Centroid};
end if;
if (q.Dc ≥ t + Cluster.Radius) then
        return;
end if;
if (q.Dc ≤ t - Cluster.Radius) then
        OUT ←OUT ∪{Cluster.MList};
        return;
end if;
for each O ∈ Cluster.MList do
        if (q.Dc ≥ t + O.Dc) then
                continue;
        end if;
        if (q.Dc ≤ t - O.Dc) then
                OUT ←OUT ∪{O};
                continue;
        end if;
        if (not ( ∃ ( d_q ∈ q.Dv ∧ d_o ∈O, Dv) | d_q ≥ t + d_o ∨ d_q ≤ t − d_o ) then
                if (dist(q,O) ≤ t) then
                        OUT ←OUT ∪{O};
                end if;
        else
                 if (d_q ≤ t − d_o) then
                        OUT ←OUT ∪{O};
                end if;
```

end if;
end for each;
return *OUT*;
END VISIT_CLUSTER.

Algorithm 4 – *procedure VISIT_CLUSTER*

3.17 The *k*-Nearest Neighbor Search algorithm

The *k*-Nearest Neighbor Search algorithm takes as input the *K- Pole Tree T*, the query object *q* and the parameter *k* indicating the number of requested objects. It returns the *k* objects of the set *S* that are closer to *q*. The search algorithm recursively descends all the branches of the tree until it finds a leaf that represents a *cluster* that must be visited, or finds a subtree that contains objects that are certainly outside of the search interval and therefore can be safely discarded from the search. In order to perform this pruning, a threshold *t* is dynamically maintained, initially placed at ∞, indicating the distance between *q* and the most distant object from it so far among those included in the temporary result of the *k*-nearest neighbor search. The following will be treated detail of the procedures, similar to those previously analyzed for the range search, which involved a visit of the internal nodes and the *clusters*. In order to optimize the dynamic operations such as insertion, deletion and update, all the current *k*-nearest neighbors should be stored in a heap[7].

3.17.1 The procedure KNN_SEARCH

The procedure starts by checking that the node is internal. In case the node is a leaf, is called the procedure KNN_VISIT_CLUSTER, which deals purely of the visit of the cluster. If the node is internal, the distances D_1, ..., D_K between the query object and the elements of the K-Pole node in question are first calculated. This time, these distances are calculated using the special procedure CHECK, which in addition to explicitly calculate the distance also concerns to progressively update the threshold *t*. After the

[7] The heap is a data structure whose main feature is the constant maximum (minimum) recovery time of all elements contained therein.

calculation of each distance, it is appended to the *Dv* list of the query object, analogous to what happened in the procedure RANGE_SEARCH. The original procedure of the Antipole Tree includes two calls to another recursive function, KNN_VISIT, which in turn makes use of a mutual recursion with the main proceedings KNN_SEARCH. The order of calls is discriminated based on the distance to the closest object to the query in the Antipole pair (e.g. if the shortest distance is that relating to the object A, then KNN_SEARCH is first called on his left child, and right on the contrary if the closer object is B). In the case of *k* children you have to order the *k* distances D_1, ..., D_K and then perform *k* calls to KNN_VISIT. For this purpose a data structure ad hoc has been implemented, which takes the name of *TN_List*, which is nothing other than a linked list of nodes of the tree (which also stores the distances with the elements of *K-Pole* and the radii of subsets). Its entries are sorted based on the distance with the query object. This will create a call list, with the sort of case, which is passed to the procedure KNN_VISIT instead of making *k* calls. Obviously this method remains valid in the case that a node contains less than *k* children. Finally, after calling KNN_VISIT through the call list, the additional values inserted in the Dv field of the query are removed, similarly to what happened in RANGE_SEARCH. We note that the procedure KNN_SEARCH does not make any discrimination able to prune any subtree, but only cares about making the calls to KNN_VISIT following an order based on the proximity of the query object by the various elements of the *K-pole*: it is just KNN_VISIT, as we shall see later, that discards the appropriate branches during the search. The following is the procedure KNN_SEARCH in pseudo-code:

KNN_SEARCH (*T*, *q*, *t*, *OUT*, *k*)

if (*T.Leaf* = *FALSE*) **then**

for each $D_i \in D_1,\ldots,D_K$

$D_i \leftarrow$CHECK_ORDER (*q*, *T.K-Pole_Seq*[i], *t*, *OUT*);

$q.Dv \leftarrow q.Dv \cup \{D_i\}$;

Call_List ← *addElemOrd*(D_i, *T.Radius_Seq*[i]);

end for each;

KNN_VISIT(*Call_List*, *q*, *t*, *OUT*, *k*);

$q.Dv \leftarrow q.Dv \setminus \{D_1,\ldots,D_K\}$;

else

KNN_VISIT_CLUSTER(*T.Cluster*, *q*, *t*, *OUT*, *k*);

end if;

END KNN_SEARCH.

Algorithm 5 – *The procedure KNN_SEARCH*

3.17.2 The procedure KNN_VISIT

The procedure KNN_VISIT cycles the *Call List* that is passed as parameter, taking care to mutually invoke the procedure KNN_SEARCH on the elements of the Call List, but only upon least one of two conditions:

1. | OUT | < *k*

Meaning: *k* elements that are part of the result not yet been found.

2. D_i < t + T.Radius_Seq [i]

Meaning: the current item in the Call List has a distance from the associated element of the K-Pole with it such that it is less than the sum the threshold current and the radius of the subset corresponding to the element of Call List. The following is the procedure KNN_VISIT in pseudo-code:

KNN_VISIT (*CL*, *q*, *t*, *OUT*, *k*)

for each $C_i \in CL$

if |*OUT*| < *k* **then**

KNN_SEARCH (C_i.*Tree_Node*, *q*, *t*, *OUT*, *k*);

else

if (C_i. D_i < t + $CL.Rad$[i])

KNN_SEARCH ($C_i.Tree\_Node$, q, t, OUT, k);

end if;

end if;

END KNN_VISIT.

Algorithm 6 - The procedure KNN_VISIT

3.17.3 The procedure KNN_VISIT_CLUSTER

The procedure KNN_VISIT_CLUSTER is quite similar to the procedure VISIT_CLUSTER used in the *range search*. The only difference is that when the distance is calculated explicitly (lines 1 and 21 of VISIT_CLUSTER procedure), is not used the usual *dist* function, but rather is invoked procedure CHECK_ORDER[8], which is described below.

3.17.4 The procedure CHECK_ORDER

CHECK_ORDER returns the distance between the object taken as input and the query object; Furthermore, the above object is inserted into the temporary result of the query, subject to two conditions: a) have not been found so far k objects in the list that contains the temporary result of the query, or b) calculated distance is less than the *current* threshold t. In the case that the object is inserted, the threshold value is dynamically updated, taking as new value the distance between the query and the farthest object among those belonging to the temporary result of the query. The following is the procedure CHECK_ORDER in pseudo-code:

CHECK_ORDER (q, O, t, OUT)

8 The use of CHECK_ORDER, in this case, is mainly to update the appropriate value of the threshold t.

```
Do ←dist(q, O);
if |OUT| < k then
    ORDERED_INSERT (O, OUT);
    t ←EXTRACT_MAX (OUT);
else
    if (Do < t) then
        ORDERED_INSERT (O, OUT);
        t ←EXTRACT_MAX (OUT);
    end if;
end if;
return Do;
END CHECK_ORDER.
```

Algorithm 7 - *The procedure CHECK_ORDER.*

Note to the procedure CHECK_ORDER: procedures ORDERED_INSERT and EXTRACT_MAX deal respectively with the ordered inclusion in a list of items of limited maximum number (in this case *k*) and returning the value of the distance between the query and the last element of the list.

3.18 Caching Distance Calculations

Although the main feature of the algorithms of *range search* and *k-Nearest Neighbor Search*, presented in the previous two sections is to reduce to the minimum calculations of new distances, it may happen to recalculate a distance that had been previously calculated. This is very frequent especially in the *k-Nearest Neighbor Search*, because the ordered insertion expects an explicit comparison between the distances of various objects of the result set with the query (*ORDERED_INSERT* procedure). To avoid having to recalculate a previously calculated distance is made use of a *cache*[9], which in our case has been achieved by the use of an additional field dq in the *Object* data structure. dq fields store the distance between the object and the query. Initially each dq

[9] Typically, a portion of memory suitable for storing temporary data used frequently

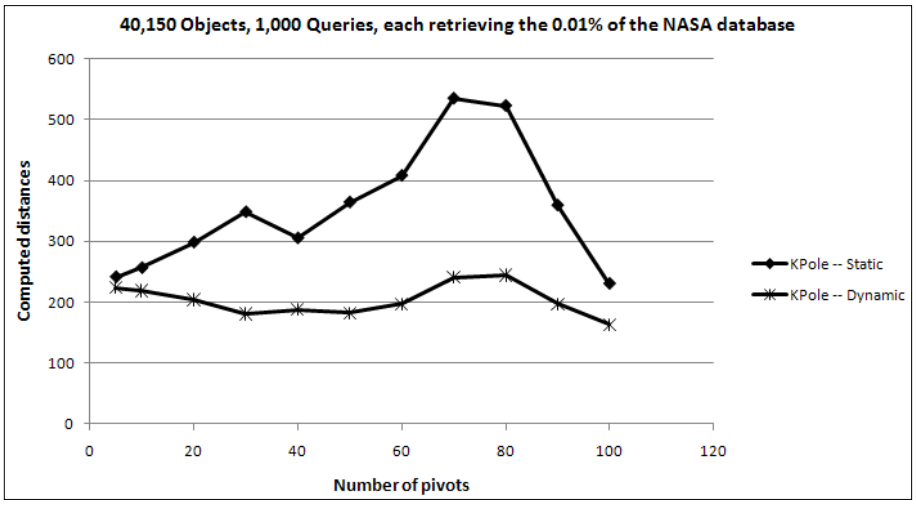
is set to an indefinite value (e.g. any negative real number, because the property ($p1$) of the metric distance function provides a positive return value, so when we go to watch the *dq* field of an object and find any negative value, we conclude that the distance between the object and the query has not yet been calculated). In this way the distance is really calculated only the first time, and as soon as the value of the distance is obtained, the *dq* field is set with the value of distance. If we need again to compare the object with the query (and this is very common especially for the *k-Nearest Neighbor Search*), we use the (positive) value stored in *dq*, without the need to recalculate the distance. If the cache had not been used, some unnecessary distance calculations were carried out (sometimes many), that would be burdened efficiency of our search algorithms.

4. Experimental Results (in Range Search and KNN Search)

In this chapter we evaluate the efficiency of K-Pole Tree at query time. We have implemented our data structure using C programming language under Linux operating system. We have conducted a very preliminary experimental analysis to establish the performances of the proposed method. For our experimental evaluation we used a collection of 40,150 images extracted from the video and image archives of NASA[10], each of them represented by a feature vector of 20 components. The Euclidean distance has been used to compare objects. We measured the performances of the system using the number of computed distances.

4.1 Static K-Pole VS Dynamic K-Pole

In the first experiment, we compared the static version of the K-Pole Tree with the dynamic one. As stopping criteria we set a threshold equal to the 25% of the diameter set for both version of the data structure. We compared the structures by performing 1,000 of range search queries randomly draw from the dataset, capturing the 0.01% of the entire data. We compared also the two structures performing 1,000 *knn* queries with k equal to 5, 10, 20 and 30 respectively.



[10] This dataset has been obtained from the Metric Space Library: http://www.sisap.org/Metric_Space_Library.html

Figure 14 – *Static version of K-Pole is compared with the dynamic one in range search*

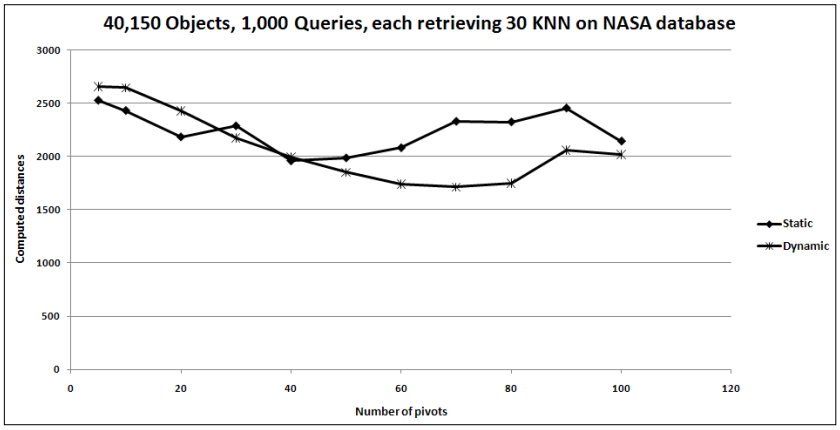


Figure 15 - *Static version of K-Pole is compared with the dynamic one in KNN search*

From Figures 14 and 15, we can easily conclude that the dynamic version of K-Pole is more efficient of the static one. In range search experiments, dynamic K-Pole results are always better than static, for all tested pivots. In *knn* experiments we observed similar results using fewer pivots to partition the space. However we found better results with the dynamic version by using a higher number of pivots (80 pivots).

4.2 K-Pole VS Other Index Structures

We compared our method with the results presented in [111]. We conducted the range search experiment using the same condition presented in [111]. We can see that the results produced by the Dynamic K-Pole tree are better than those presented in [111]. We found that our method is better for almost any value of k, saving from 25% of distance computations to almost 50% in the best case. We conducted also a comparison with the Antipole Tree data structure, setting as stopping criteria a threshold equal to the 25% of the diameter for the Dynamic K-Pole Tree, and equal to the 45% for Antipole Tree (this threshold value is the one producing the best Antipole results for this dataset). We observed that Dynamic K-Pole Tree performance is far better than the Antipole one, saving average 75% of distance computations in range search, and average 80% in *knn* search queries.

We conducted a comparison between the Dynamic K-Pole Tree and Pivots [42] in order to inspect its effectiveness. As parameters, we used for Dynamic K-Pole Tree a threshold value equal to the 25% of the dataset diameter. For Pivots we tried a set of pivots ranging from 20 to 400. For the examined dataset, we can assume that the best results for Pivots fall inside such a range. To compare the efficiency of the two indexing techniques, we performed 1,000 range queries on random objects taken from the dataset, capturing the 0.01% of the whole dataset, and 1,000 *knn* queries, on random objects too, with k equal to 5, 10, 20 and 30 respectively. We compared our best results with the best results of Pivots structure. Range search experiments state that Pivots best case is better than the Dynamic K-Pole Tree one, although Dynamic K-Pole Tree average performance on a large number of tests is better than Pivots one. The outputs for the *knn* search experiments of Pivots and Dynamic K-Pole Tree are shown in figures 16 and 17 respectively.

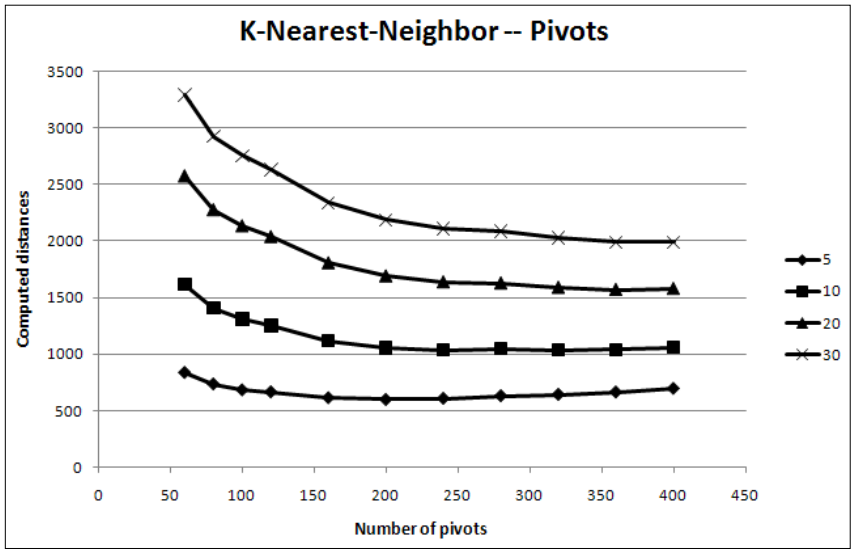


Figure 16 – *Results produced by Pivots in KNN experiments*

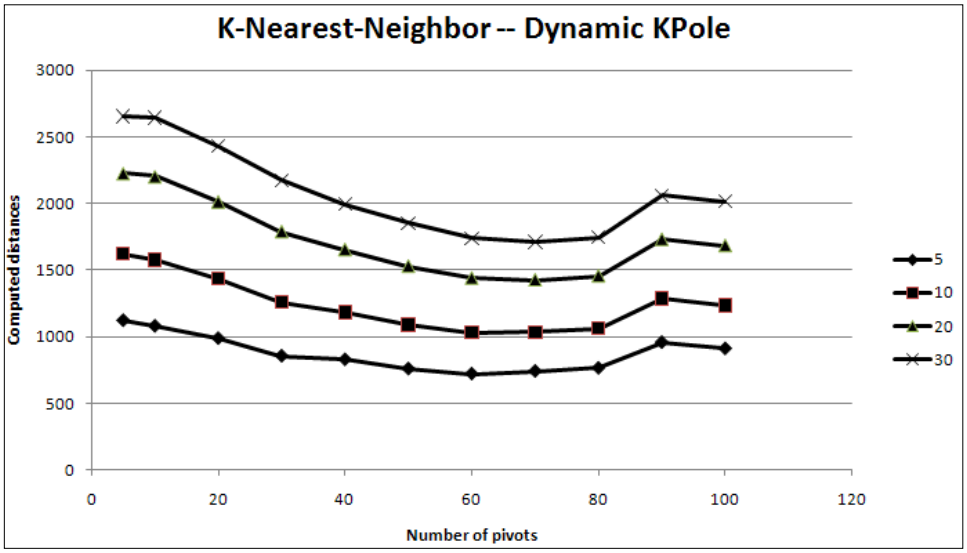


Figure 17 - *Results produced by Dynamic K-Pole in KNN experiments*

As these two figures reveal, Dynamic K-Pole Tree performs better than Pivots in almost all the observed scenarios, using only a fraction of the pivots needed by the Pivots structure. Although for *knn* = 5 Pivots in its best case saves average 20% of distance computations than Dynamic K-Pole Tree, it is remarkable that Pivots achieve this result using 200 pivots, while the best case of Dynamic K-Pole Tree is obtained with only 60 starting pivots. For *knn* = 10, 20, 30 K-Pole Tree performs better than Pivots, saving respectively the average 5%, 15% and 20% of distance computations. Nonetheless, Dynamic K-Pole Tree reaches these results using a number of starting pivots that is 50% less than the number of pivots used by Pivots.

4.3 Pivots decreasing rate

From our experimental analysis we observed that the number pivots selected dynamically by the algorithm rapidly decrease (see Figure 18).

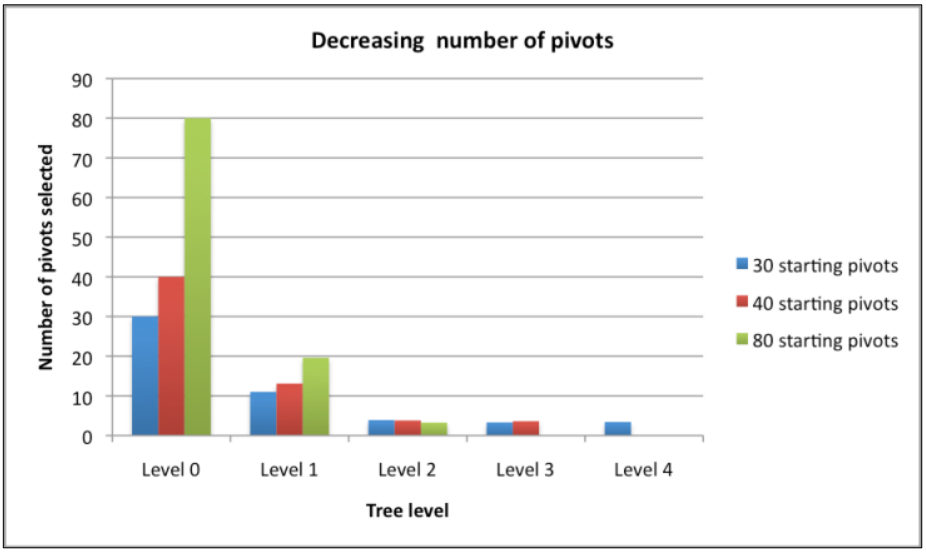


Figure 18 – *Decreasing rate of pivots in K-Pole Tree*

5. Pattern recognition analysis methods

Automatic extraction, recognition, description and classification of patterns extracted from images and signals are the most important tasks in several scientific disciplines. In the last years pattern recognition techniques together with time series analysis constitute a novel research field in active volcano modeling and can provide useful information for monitoring purposes. There are many definitions of pattern recognition (hereafter referred to as PR). In [138] PR is defined as the problem of estimating density functions in a high-dimensional space and dividing the space into regions of categories of classes. In [31] PR is defined as a field concerning machine recognition of meaningful regularities in noisy or complex environments. In [162] PR is defined as the process of classifying input data via extraction of significant features from a set of noise data. Other definitions of the term pattern are given by several authors. The main aspect of PR is the definition of a set of peculiar features or descriptive elements of the analyzed objects. Given a pattern, the recognition process consists of one of the following tasks:

i) Supervised classification in which the patterns are classified on the basis of a known learning rule

ii) Clustering that is the process of grouping sets of objects into classes called clusters with no a priori knowledge

In the first case the classifier design can be implemented by several techniques, implying the definition of a metric based on template matching or the minimum distance between pattern and class prototype. There are different methods for metric definition, such as vector quantization and learning vector quantization [139, 140]. For instance, in statistical pattern recognition the classification process is based on probabilistic approaches: the pattern-class association is computed on the basis of a probability density function. The optimal Bayes decision rules assign a pattern to a class with the maximum posterior probability [141]. Other classification techniques are based on geometric approaches. These kinds of classifiers are based on a training procedure that minimizes an error (such as the mean square error, MSE) computed by comparing classification output and target value. Typical examples are Fisher's linear discriminant (LDA) and single layer perceptron (SLP). The former minimizes the MSE between the classifier output and the desired label, the latter iteratively updates the decision surface

in the form of hyperplane between classes. Fisher's discriminant can be extended in non-linear classification where original observations are mapped into a higher dimensional non-linear space. Applying linear classification on this higher dimensional space is equivalent to non-linear classification in the original space. The most commonly used technique is the kernel Fisher discriminant [143]. The nonlinear extension of SLP is the Multi-Layer Perceptron (MLP) that allows nonlinear decision boundaries and overcomes many of the limitations of single layer perceptron. A powerful method in classifier design is the Support Vector Machine (SVM) introduced in [90]. This algorithm is different from other hyperplane-based classifiers such as SLP. The problem of estimating hyperplane separating two classes is not unique. The SVM algorithm is able to find the optimal hyperplane that separates the classes. Beyond supervised classification, the other important task in PR is the clustering problem, that, as aforementioned, is the process of grouping data without any a priori information.

5.1 Clustering: an overview

Objects belonging to the same cluster will be more similar than objects belonging to different clusters with respect to some given similarity measure. An object could be referred to a physical object or, in general, to a physical recording, such as a waveform, on which a set of quantitative descriptive elements can be identified. A pattern is a set of features describing an object and generally is represented as a vector. Therefore, a set of patterns is represented by a pattern matrix. The problem of clustering arises in many different scientific fields, and, thus, a vast amount of literature has been produced on the subject. Existing clustering methods can be broadly divided into hierarchical and partitioning [113]. Hierarchical algorithms gradually (dis)assemble objects into clusters. On the other hand, partitioning algorithms learn clusters directly trying to discover clusters either by iteratively relocating points between subsets or by identifying areas heavily populated with data. This second type of partitioning algorithms attempts to discover dense connected components of data. Examples of algorithms belonging to such a category are: DBSCAN, OPTICS, DENCLUE [113].

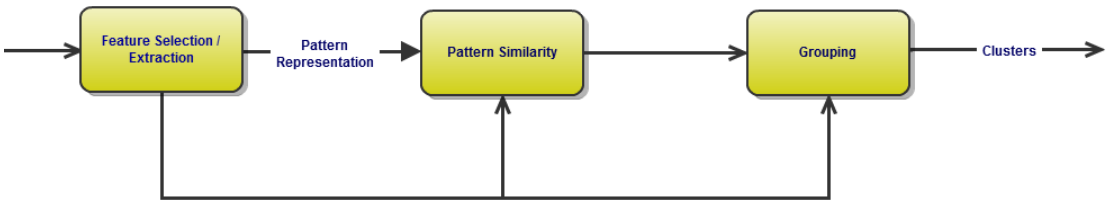


Figure 19 – *Clustering stages*

In figure 19 a general clustering task is shown [39]. It can be synthesized as follows:

- pattern representation
- definition of an appropriate pattern proximity measure to the data domain
- clustering algorithm
- data abstraction (if needed)
- assessment of clustering result

As reported in [137], the pattern representation refers to the number of classes, number of available patterns, and number, type and scale of the features available for the clustering algorithm. While the feature selections task identifies the most effective features for discriminating available patterns, the features extraction can be defined as the process transforming the available features into new features with the purpose to better describe characteristics usable in the clustering process. For instance, considering a simply decaying waveform, the descriptive characteristics may be the frequency peak, a measure of decay and the amplitude. Pattern proximity is usually evaluated by a distance function defined on pairs of patterns. Given a set of n patterns with d features the pattern set can be expressed in matrix form using an $n \times d$ matrix. The j^{th} feature of the generic i^{th} pattern is x_{ij}. Using this notation, the i^{th} pattern is described by the column vector $x_i = [x_{i1}x_{i2} \ldots x_{id}]^T$. The clustering process requires proximity indices between all pairs of patterns [144]. A possible distance index is the Minkowski metric defined as:

$$d(p,q) = \left(\sum_{j=1}^{d} \left| x_{qj} - x_{rj} \right|^m \right)^{1/m} = \left\| X_q - X_r \right\|_m$$

where q and r are pattern matrix indices. For $m = 2$ the Minkowski metric produces the Euclidean distance between two patterns, while for $m = 1$ results in the Manhattan distance [145, 137]. In particular, the Euclidean distance is commonly used to evaluate

proximity of objects in two or three-dimensional space and it is a good choice when the data set has compact or isolated clusters [142].

To avoid features of large degree dominate the others, normalization process of the feature is required. A simple normalization can be performed by a mapping process of the original feature space into the range [0,1] by dividing the distance for each feature by the feature's range. The features distance measures can be distorted by the presence of a linear correlation. To overcome this problem a whitening process of the data can be applied or the squared Mahalanobis distance can be used:

$$d_M(X_q - X_r) = (X_q - X_r)\Sigma^{-1}(X_q - X_r)^T$$

where the patterns X_q and Xr are row vectors, $\sum$ is the covariance matrix. This distance definition assigns different weights to different features based on their variances and pair wise linear correlation [137]. It is noteworthy that distances computation between pairs of patterns exhibiting some or all noncontinuous features, may be problematic. In order to overcome this problem, a proximity measure for heterogeneous types of pattern should be used. For instance, in [147] has proposed a combination of a modified Minkowski metric for continuous features and a distance based measure for nominal attributes. A variety of other metrics can be found in [148] and [149]. Another way of representing patterns is by using string or tree structures [150]. In particular, patterns string representation is commonly used in syntactic clustering [151]. Several measures of similarity between strings can be found in [161], while similarity measures between trees are reported [146]. Once the indices of dissimilarity or similarity have been computed, the outliers can be identified based on high distances from other patterns. Depending on the data domain and the intended goal of the clustering process, such outliers may be removed prior to applying a clustering algorithm. However, outliers sometimes provide useful information that would otherwise be lost if removed prematurely [145].

5.1.1 Hierarchical vs. partitional algorithms

Hierarchical clustering builds a cluster hierarchy or a tree of clusters, also known as a dendrogram. Each node in the dendrogram represents a cluster. The root of the dendrogram is a cluster that includes all objects, while each child contains a sub-cluster

of its parent node. Hierarchical clustering methods are categorized into agglomerative (bottom-up) and divisive (top-down) [39, 152]. The agglomerative clustering process starts with one-point clusters, called singleton, and recursively merge two or more of the most similar clusters. A divisive hierarchical clustering, by beginning with the root node, starts with a single cluster containing all data points and recursively splits the most appropriate clusters. The process continues until a stopping criterion (for example the requested number *k* of clusters) is achieved. The height of each node is proportional to the measure of similarity or dissimilarity between its sub-clusters. Typically, the leaf nodes in the dendrogram represent individual patterns. The hierarchical clustering approach allows data exploration on different levels of granularity (figure 21).

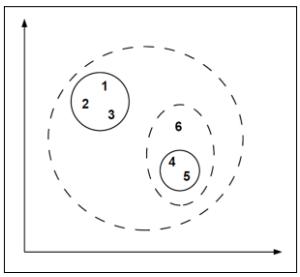


Figure 20 - *A possible set of clusters for six patterns*

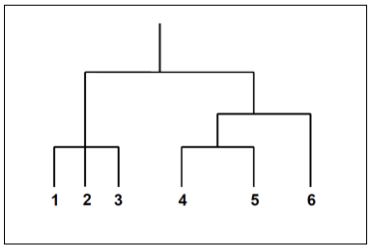


Figure 21 - *Dendrogram corresponding to the clusters of figure 20*

The creation of a hierarchical clustering may involve excessive time and space constraints, because the algorithm used must make an iteration for each level in the hierarchy [153].

Another clustering approach is the partitional clustering algorithm producing a single partition of the patterns. This kind of clustering process requires less time and space than hierarchical clustering. However, partitional methods require the user to choose a specific number k of clusters, in general known a priori, to be created. Additionally, while a partitional method produces only one final set of clusters, the algorithm may

create the final set iteratively, in the form of iterative optimization, beginning each time with a different starting configuration and choosing the best result from all the runs [137]. Unlike traditional hierarchical methods, in which clusters are not revisited after being constructed, relocation algorithms can gradually improve clusters. With appropriate data, this results in high quality data [113]. Another kind of partitioning algorithm is the Density-Based Partitioning. As the name suggests these clustering approaches rely on a density-based notion of clustering and are capable of discerning clusters of arbitrary shape in spatial database with noise. One of the best known density based clustering approach is the Density-Based Spatial Clustering of Applications with Noise (*DBSCAN*).

5.1.2 Cluster assessment

Process of cluster assessment is multi-faceted. Indeed, this process can be involved in the assessment of data domain itself or as a validation of clustering results. The former should be verified if the data set contains a reasonable number of clusters prior to performing cluster analysis. In this process the assessment procedure should be used to estimate if the data set has a low cluster tendency, meaning there is or no a benefit in attempting to perform clustering. In this case, data which do not contain clusters should not be processed by a clustering algorithm. [137, 145].

Assessment process as validation task is the goodness estimation of the clustering algorithm results. Let K_m as cluster of N points $\{t_{m1},t_{m2},...,t_{mN}\}$, three typical measurements are proposed in [153] and are centroid, radius and diameter respectively. The centroid can be viewed as the middle of the cluster and may not coincide with a point of cluster itself. It can be expressed as:

$$C_m = \frac{\sum_{i=1}^{N} t_{mi}}{N}$$

Alternatively, a medoid can be defined as the centrally located point of the cluster. The radius represents the averaged mean squared distance from any point to the cluster's centroid. It is defined as:

$$R_m = \sqrt{\frac{\sum_{i=1}^{N} (t_{mi} - C_m)^2}{N}}$$

The diameter is defined as the square root of the averaged mean squared distance between all pairs of points in the cluster. The diameter of a cluster *m* is then defined as:

$$D_m = \sqrt{\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (t_{mi} - t_{mj})^2}{N(N-1)}}$$

Common clustering evaluation involves the clusters distance by utilizing the centroid of cluster. In [144] three types of validation studies are reported. In particular, an external assessment of validity compares the recovered structure to an a priori structure and attempts to quantify match between the two. An internal examination of validity tries to determine if the clustering structure is intrinsically appropriate for the data. This assessment considers whether a given cluster is unusually compact or isolated compared to other clusters of the same size in random data [145]. A relative test compares two structures and measures their relative merit [137]. Indices used for this comparison are discussed in detail in [39] and [144]. An internal cluster validation measure is the Davies-Bouldin (DB) index [120]. Such an index is function of the number of clusters, the inter-cluster and within-cluster distances. Formally it is defined as follows:

$$DB = \frac{1}{n} \sum_{i=1}^{N} \max_{i \neq j} \left\{ \frac{S_n(Q_i) + S_n(Q_j)}{S_n(Q_i, Q_j)} \right\}$$

where S_n is the average distance of all cluster objects to their cluster centers, $S(Q_i, Q_j)$ is the distance between clusters centers. Small values of DB correspond to compact clusters whose centers are far away from each other. In the light of it, the number of clusters that minimizes DB is taken as the optimal number of clusters.

5.1.3 Squared error clustering and k-means algorithm

Partitional clustering algorithms perform a subdivision of data set into clusters based on some distance metrics. The most common used criterion function is the squared error criterion, which tends to work well with isolated and compact clusters. This kind of

methods is iterative and the clustering process terminates when a stopping criterion is met. For example, a common stopping criterion is when squared error between successive iterations ceases to decrease significantly. Let a cluster C of size k and a pattern set X, the squared error is defined as:

$$e^2(X,C) = \sum_{j=1}^{k} \sum_{i=1}^{n_j} \left\| x_i^{(j)} - c_j \right\|^2$$

where j represents the cluster number, n_j denotes the number of elements, $x_i^{(j)}$ represents the ith pattern of cluster j and c_j is the centroid of jth cluster [137]. General squared error clustering can be summarized as follows: 1) Set an initial partition of the data into k clusters and calculate initial k cluster centers. 2) Assign each pattern to its closest cluster center and compute the new cluster centers based on the new cluster assignments; compute the squared error; repeat this step until a stopping criterion is met (i.e., until the cluster membership is stable). 3) Merge and split clusters based on some heuristic information, optionally repeating step 2. The most common square error clustering method is the *k*-means clustering algorithm [154]. It differs from the general squared error clustering method only in the initialization phase where k cluster means are chosen and patterns are assigned to its closest mean. In this case the cluster mean is defined as:

$$c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_i^{(j)}$$

The *k*-means algorithm is easy to implement, and its time complexity is O(n), where n is the total number of patterns. This algorithm converges to a locally but not globally optimal solution and depends on the choice of the initial cluster means. *K*-means algorithm can be summarized as follows [137]:

1) Choose k cluster centers to coincide with k randomly-chosen patterns or k randomly defined points inside the hypervolume containing the data set.

2) Assign each pattern to the closest cluster center.

3) Recompute the cluster centers using the current cluster memberships.

4) If a convergence criterion is not met, repeat step 2. Typical convergence criteria are: no (or minimal) reassignment of patterns to new cluster centers, or minimal decrease in squared error.

Since *k*-means algorithm is based on the concept of centroid, it is sensitive to noise and outlier and a data filtering procedure is required. One solution is the use of feature weighting and a distortion measure to produce a partition that will minimize average within-cluster variance while simultaneously maximize the average between cluster distances [155]. Several variants [156] of the k-means algorithm have been reported in the literature.

For instance, another distance measure may be [158]:

$$d(x,y)=1-e^{-\beta\|x-y\|^2}$$

β is a positive constant. The measure provided by this equation is robust to data outlier and noise. Other methods are based on the idea of down weight data outlier respect to its cluster centroid [157]. As aforementioned, k-means algorithm assumes that the number k of clusters is known a priori. In practical applications the optimal value of *k* is, in general, not known. To overcome this limitation, many k-means clustering algorithms are executed with many values of k. The best *k* value is then determined on the basis of some cluster assessment criterions (section 5.2.). A possible approach may be the use of validity algorithms such as Davies-Bouldin (DB) index. As explained in section 5.2, the number of clusters that minimizes DB is taken as the optimal number. An example of 3-class k-means together with DB index is shown in figure 22. In particular, figure 22a shows the best 3-clustering structure of the data set, while figure 22b shows the value of DB index for increasing value of k. The best cluster number is chosen on the basis of minimum value of DB index. As aforementioned, k-means algorithm is very sensitive to the choice of the initial partition leading to a locally optimal solution instead of a globally optimum. To alleviate this problem, different initialization strategy have been proposed. One of the common initialization method is the Random Approach (RA) where the initial partitioning is computed in random way. Other methods that apply different strategy can be found in [159, 160].

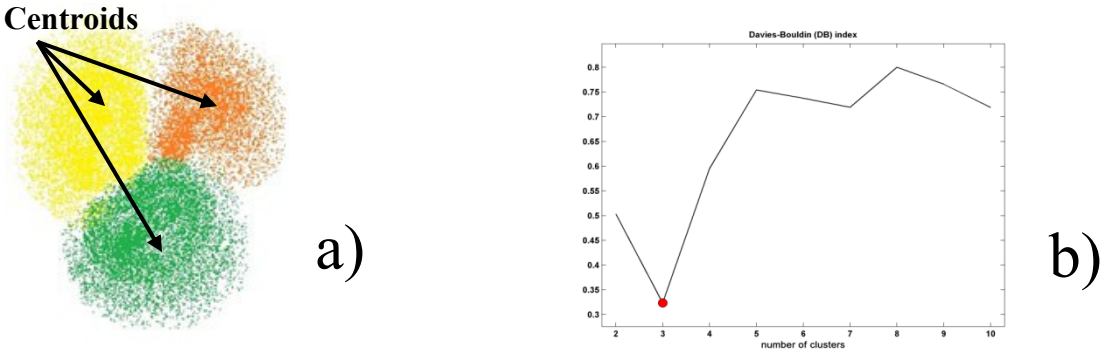


Figure 22 - *a) A features plane with three clusters; b) best clustering structure computed on the features plane using Davies-Bouldin index.*

5.1.4 Clustering algorithm based on DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise [80]) is a density-based clustering algorithm able to discover clusters of arbitrary shape in spatial databases with noise (figure 23). Clusters are defined as maximal sets of density-connected points. Usually DBSCAN runs on datasets drawn from multidimensional or metric spaces and uses a distance function to compare objects. Given a dataset D of objects DBSCAN makes use of the following structures and definitions:

i) ε - neighbourhood
ii) core point
iii) directly density-reachable
iv) density-reachable
v) density-connected

The ε-neighbourhood of a point p, denoted by $N\varepsilon(p)$, is a subset of points q in D, such that a distance measure dist(p,q) (such as the Euclidean distance) is lower than ε. The point p is called *core point* or core object if its ε-neighbourhood has cardinality above a minimum threshold called MinPts. Each point q which lies in the ε-neighbourhood of a point p is called directly density-reachable from p (figure 24a). A point q is density-reachable from a point p with respect to ε and MinPts if there is a chain of points $q_1,...,q_n$ such that $q_1 = p$, $q_n = q$ and $q_i + 1$ is directly density-reachable from q_i for each i (figure 24b). A point q is density-connected to a point p with respect to ε and MinPts if there is

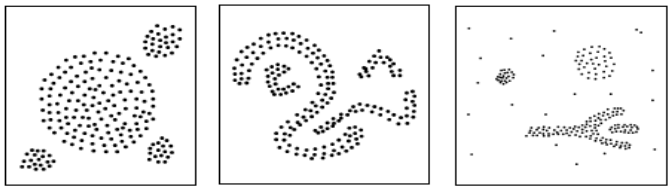


Figure 23 - *Different clusters shape*

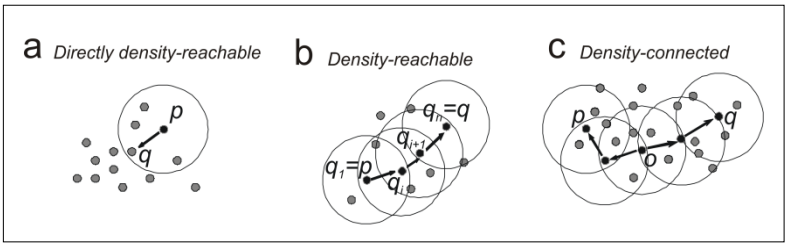


Figure 24 - *Examples of (a) directly density-reachable, (b) density-reachable, (c) density connected in density-based clustering*

a point *o* such that both *p* and *q* are density reachable from *o* with respect to ε and MinPts (figure 24*c*). Given *D*, *ε* and MinPts as input parameters, DBSCAN clusters *D* by checking the ε-neighborhood of each object in *D*. If the *ε*-neighborhood of an object *p* contains more than MinPts, a new cluster with *p* as core object is created. DBSCAN iteratively collects directly density-reachable objects from these core objects. The process terminates when no new objects can be added to any cluster. In such a case the algorithm will return the set of clusters and a special cluster containing outliers.

5.1.5 Features classification using SVM

Support Vector Machines(SVMs) are a popular machine learning method for solving problems in classification and regression, able to guarantee high classification quality [115]. In recent years, novel applications of SVM have been performed in several research areas such as biology [76, 119] and volcano seismology [64, 47]. The SVM algorithm can be summarized as follows. It first uses a non-linear mapping to transform the original data set into a higher dimension space. Next, it identifies a hyperplane able to maximize the margin of separation among the classes of the training set. Such a hyperplane is called maximum marginal hyperplane (MMH). The margin in SVMs

denotes the distance from the boundary to the closest data in the feature space (Fig. 25). With appropriate mapping, data from two classes can always be separated by a hyperplane.

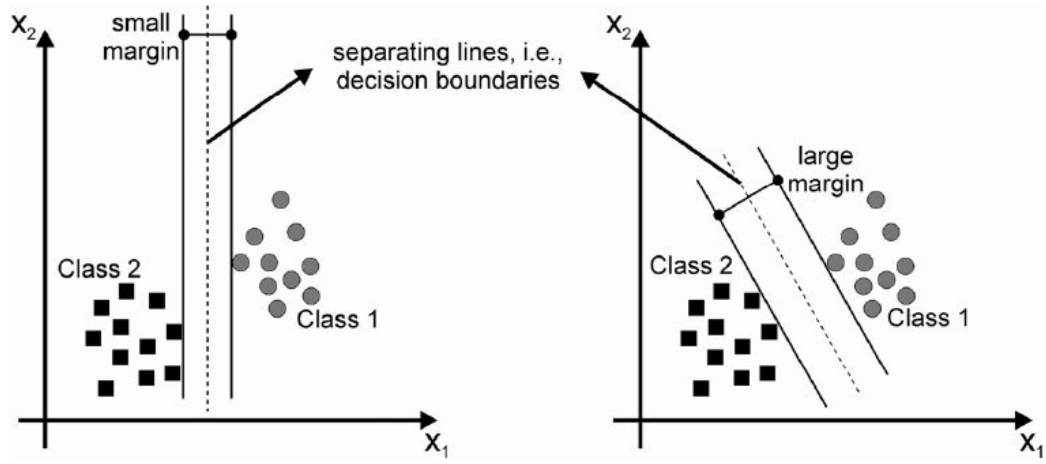


Figure 25 – *Two-feature planes each of which with two classes of data (black squares and grey circles) and a separating line (dashed lines): the left one shows a small margin between clusters, the right one a larger margin (redrawn from [37])*

The problem of computing the MMH can be formulated in terms of quadratic programming in the following way [134].

$$W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

subject to

$$\sum_{i=1}^{l} y_i \alpha_i = 0, \forall i : 0 \leq \alpha_i \leq C$$

The number of training data is denoted by l, α is a vector of l variables, where each component α_i corresponds to a training data (x_i, y_i). C is the soft margin parameter controlling the influence of the outliers (or noise) in training data. The kernel for linear boundary function is x_i y_i, a scalar product of two data points. The non-linear transformation of the feature space is performed by replacing $k(x_i, y_i)$ with an advanced kernel φ, such as polynomial kernel $(x^T x_i + 1)^p$ or a radial basis function kernel $\exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$.

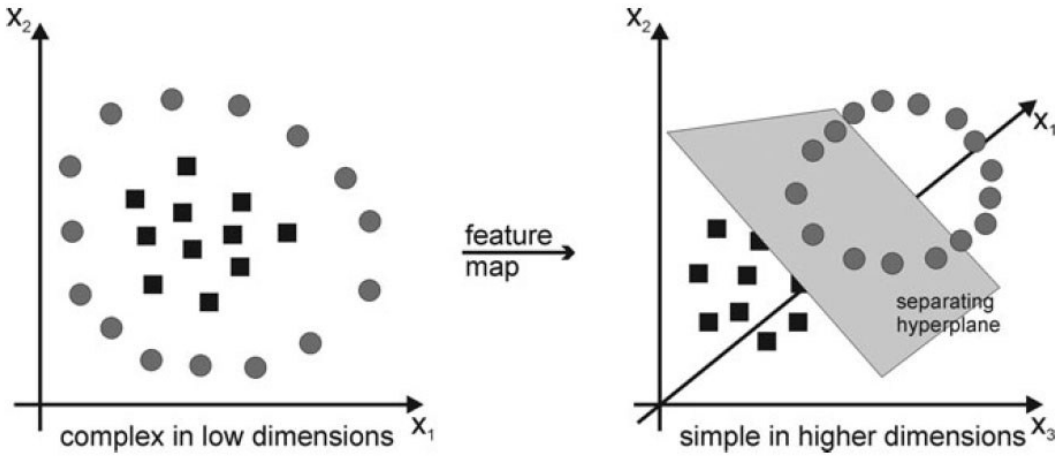


Figure 26 – *Two classes of data in the original 2-D space (left) and in a higher-dimensional feature space (right)*

The use of an advanced kernel is an attractive computational shortcut, which avoids the expensive creation of a complicated feature space. An advanced kernel is a function that operates on the input data but has the effect of computing the scalar product of their images in a usually much higher-dimensional feature space (or even an infinite-dimensional space), which allows one to work implicitly with hyperplanes in such highly complex spaces (Fig. 26). The extension of SVM to multiclass problems can be performed using two different methods called one-against-one and one-against-all. The former constructs k (k − 1) / 2 classifier where each one is trained on data from two classes. The latter constructs k SVM classifier. In this last case, the i^{th} SVM is trained using all training patterns belonging to i^{th} class with positive labels and the other with negative labels. A point is assigned to the class for which the distance from margin is maximal. Finally, the output of one-against-all method is the class that corresponds to SVM with highest output value [93, 133].

5.2 Model selection

Typically our model will have a set of tuning parameters α that varies the complexity of our model, and we wish to find the set of α that minimizes error, that is, produces the minimum of the average test error. As mentioned the choice of such parameters is a key step in model learning because their values determine classification performance. It is important to note that there are two separate goals in a classification modeling:

1) model selection that is the task of estimating the performance of different models in order to choose the best one;

2) model assessment: having chosen a final model, estimating its prediction error (generalization error) on new data.

As a consequence, model selection is applied with the aim of finding the best set of parameters that minimizes the error rate estimated as the ratio between misclassified and hit patterns. More in detail, in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The first set is used to fit the models; the second set is used to estimate prediction error for model selection; the third set is used for assessment of the generalization error of the final chosen model [130]. In practical application it is difficult to give a general rule on how to choose the number of observations in each of the three parts. In particular, it depends on the signal-to-noise ratio and the training sample size.

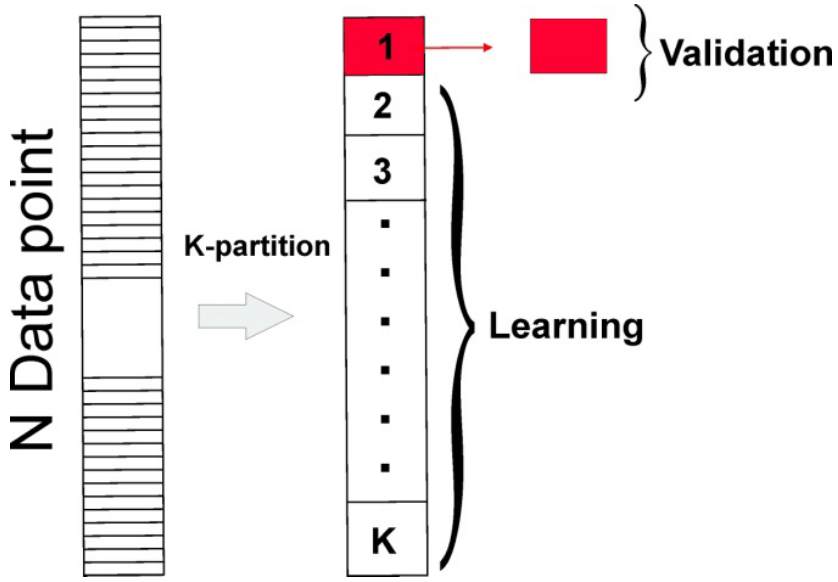


Figure 27 - *Basic scheme of K-Fold Cross-Validation*

A typical split might be 50% for training, and 25% each for validation and testing. In order to overcome this limitation, model parameters can be chosen using a *cross-validation* (CV) approach, which is a statistical method for learning algorithms evaluation and model selection. In particular, in *K-fold* CV the available dataset is

partitioned into K subsets or "folds": K-1 folds are used for classifier learning purpose, and the remaining fold for model validation. Thus, K iteration of learning and validation are performed and for each i^{th} iteration the training process is carried out using K-1 folds and the i^{th} fold for validation (figure 27).

The model selection procedure can be summarized as follows:

1) for each set of tuning parameters a mean error rate is computed averaging the error rate values obtained by the K classifiers $\{C_1,C_2,..,C_k\}$;

2) the set of parameter α with the minimum error rate are selected;

3) such parameters set is used to train the final classifier with the whole dataset, comprising all the K folds.

The model selection scheme is shown in figure 28.

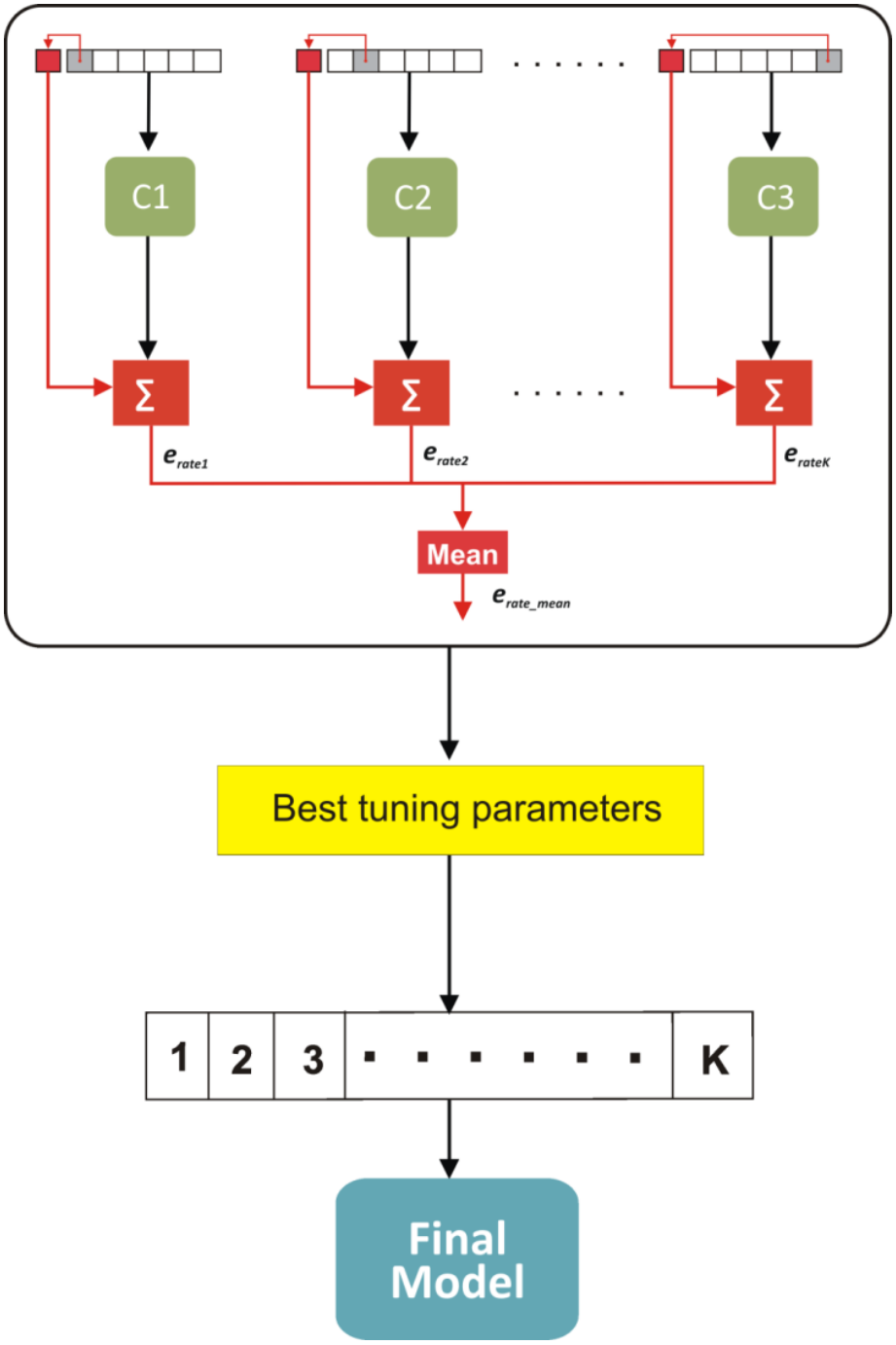


Figure 28 - *Best model selection using K-Fold cross-validation*

6. Data Mining in Geophysics

This chapter contains a collection of works, concerning the application of data mining on geophysical data, that became object of publication [163, 164]. This was made possible thanks to the collaboration with the INGV (Istituto Nazionale di Geofisica e Vulcanologia), Osservatorio Etneo, which makes available its data for this kind of research.

6.1.1 Clustering and classification of infrasonic events at Mount Etna using pattern recognition techniques

Two of the fundamental tasks of volcano monitoring are to follow volcanic activity and promptly recognize any changes. To achieve such goals, reliable field measurements and advanced data analysis methods are required. Different geophysical techniques (i.e. seismology, ground deformation, remote sensing, magnetic and electromagnetic studies, gravimetry) are used to obtain precise measurements of the variations induced by an evolving magmatic system. In recent years, useful information to monitor the explosive activity of volcanoes, as well as to investigate its source processes, have been provided by studying infrasonic signals [92, 82, 117, 118, 61]. The location of the source of the infrasonic signals, generally coinciding with active vents, is of great importance for volcanic monitoring. Thus, different techniques, generally based on the comparison of the infrasonic signals using cross-correlation or semblance functions, have been developed [82, 126, 135, 136, 21, 71]. Over the last decades, Mt. Etna volcano (Italy) has been characterized by a remarkable increase in the frequency of shortlived, but violent eruptive episodes at the summit craters. Between 1900 and 1970, about 30 paroxysmal eruptive episodes occurred at the summit craters, while there have been more than 180 since then [112]. The summit area of Mt. Etna is currently characterized by four active craters: Voragine, Bocca Nuova, Southeast Crater and Northeast Crater (hereafter referred to as VOR, BN, SEC and NEC, respectively; see Fig. 29). These craters are characterized by persistent activity that can be of different and sometimes coexistent types: degassing, lava filling or collapses, low rate lava emissions, phreatic, phreato-magmatic or strombolian explosions and lava fountains [116]. At Mt. Etna in 2006, a permanent infrasound network was deployed providing useful information to

monitor the explosive activity [117, 118, 122]. Unfortunately, sometimes during the winter season owing to bad weather conditions, the lack of signals from some summit stations prevents applying the aforementioned location algorithms. Here, we propose a new system, based on pattern recognition techniques, able to identify at Mt. Etna the active summit crater from the infrasonic point of view using only the signal recorded by a single station.

6.1.2 Infrasound features at Mt. Etna

Some recent studies have shown that the infrasonic signal at Mt. Etna is generally composed of amplitude transients (named 'infrasonic events'), characterized by short duration (from 1 to over 10 s), impulsive compression onsets and peaked spectra with most of energy in the frequency range 1–5 Hz (Fig. 30; [127, 117, 118]. Similar features are also observed at several volcanoes, though characterized by different volcanic activity, such as Stromboli [83], Klyuchevskoj [124], Sangay [17], Karymsky [17], Erebus [86], Arenal [128] and Tungurahua [87]. Since the deployment of the infrasound permanent network at Mt. Etna in 2006, two summit craters have been recognized as active from the infrasonic point of view: SEC and NEC [117, 118]. The former has been characterized by sporadic explosive activity with different intensity, from ash emission to lava fountaining, while the latter mainly by degassing. According to [117, 118], these craters generate infrasound signals with different spectral features and duration: 'SEC events', showing a duration of about 2 seconds, dominant frequency mainly higher than 2.5 Hz and higher peak-to-peak amplitude than the NEC events (Fig. 30a); 'NEC events', lasting up to 10 s and characterized by dominant frequency generally lower than 2.5 Hz (Fig. 30b).

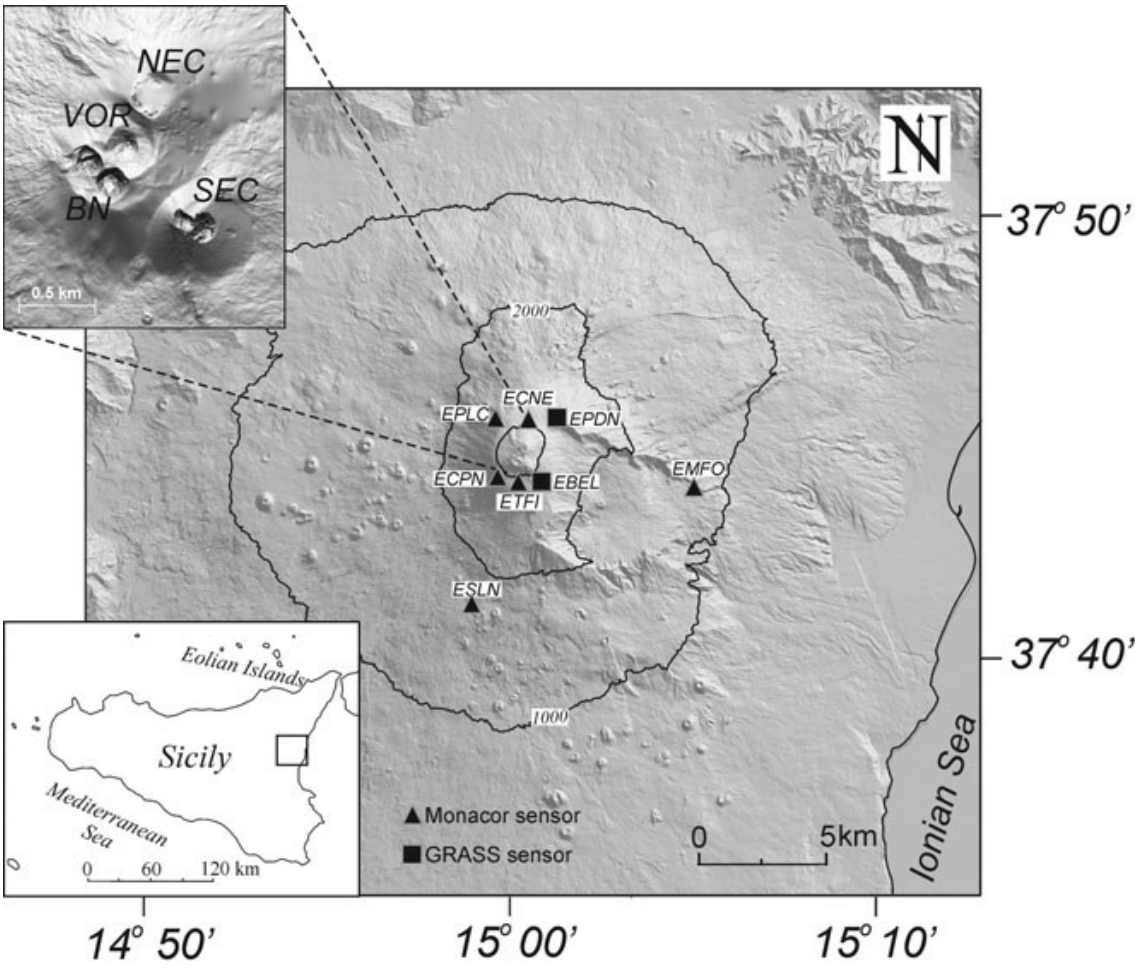


Figure 29 - *Digital elevation model of Mt. Etna with the location of the infrasonic sensors (triangles and squares), composing the permanent infrasound network. The upper right inset shows the distribution of the four summit craters (VOT, Voragine; BN, Bocca Nuova; SEC, Southeast Crater; NEC, Northeast Crater).*

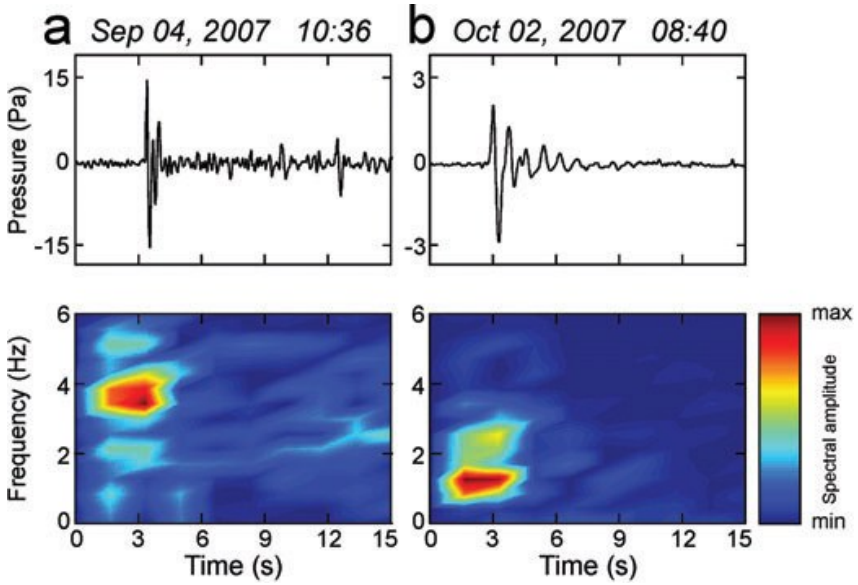


Figure 30 - *Infrasonic events recorded by EBEL station and the corresponding Short Time Fourier Transform, obtained by using 2.56 s long windows overlapped by 1.28s. The event in (a) is a typical 'SEC event', the one in (b) a typical 'NEC event'.*

6.1.3 Data Acquisition and Event Detection

Since 2006, the permanent infrasound network run by Istituto Nazionale di Geofisica e Vulcanologia – Osservatorio Etneo, has been composed of a number of stations ranging from one to eight depending on the considered period, located at distances ranging between 1.5 and 7 km from the center of the summit area (Fig. 29). Today, some stations are equipped with Monacor condenser microphones MC-2005, with a sensitivity of 80 mV Pa^{-1} in the 1–20 Hz infrasonic band, while others with GRASS 40AN microphone with a flat response with sensitivity of 50 mV Pa^{-1} in the frequency range 0.3 – 20000 Hz. The infrasonic signals are transmitted in real-time by means of radio link to the data acquisition center in Catania where they are acquired at a sampling rate of 100 Hz. At Mt. Etna we use EBEL as reference station, because it generally shows a very good signal-to-noise ratio and, unlike the other summit stations, its maintenance is generally feasible even during the winter season. Once the infrasound signal is recorded, the signal portions of interest, that are the infrasonic events, have to be extracted. Then, the root mean square (*rms*) envelope of the infrasonic recordings is calculated by a moving window of fixed length. Successively, we calculate the percentile envelope on moving windows of rms envelope. For a given time-series, the p_{th} percentile can be defined as the value such that at most (100 × p) percent of the

measurements are less than this value and 100(1 – p) per cent are greater. In light of this, the estimation of percentile enables us to efficiently detect amplitude transients and estimate background signal level. The percentage threshold should be chosen on the basis of both the amount of transients in the signal that have to be included or excluded in our calculations and the signal-to-noise ratio. The performance of this method was compared with the short time average/ long time average (STA/LTA) technique [98, 101]. The lengths of short and long windows, mainly depending on the frequency content of the investigated signal, were fixed respectively to 2.5 and 12.5 times the dominant period of the signal (equal to roughly 0.3 s), considered a reasonable compromise between sensitivity and noise reduction [98], and the detection threshold to 1.7. As shown in Fig. 31, the trigger results obtained by the two methods were similar; nevertheless, the technique based on percentile was also able to detect transients very close to each other.

6.1.4 Infrasonic signal features extraction

Often the decomposition of a time-series into purely harmonic components (Fourier transform case) can be impractical. In fact, the actual oscillations observed in geophysics often decay (or grow) exponentially with time, due to some mechanisms of energy dissipation (or supply), as if the frequency were complex [40]. Therefore, the spectral structure will be reasonably represented in the complex frequency space [40]. Since infrasonic events can be represented as decaying complex exponential functions, to determine their complex frequency the Sompi method can be used [40]. This is a high-resolution spectral analysis method based on an autoregressive (AR) filter. By this method, a given time series is resolved into a number of 'wave elements' that consist of decaying harmonic components, and additional noise. Each wave element is specified by two complex parameters z and α [40]

$$z = \exp(\gamma + \mathrm{i}\omega), \quad \alpha = Ae^{\mathrm{i}\theta}$$

where γ and ω are the real and imaginary parts of the complex angular frequency, A and θ correspond to the real amplitude and phase of the wave element referred to some origin point and finally i is $\sqrt{-1}$. Another two parameters, ordinary real frequency and 'gradient' or 'growth rate', referred as to f and g, respectively [40], are given by

$$f = \omega / 2\pi, \quad g = \gamma / 2\pi.$$

Finally, the ‘dissipation factor’ or ‘quality factor’ Q is defined as

$$Q = -f / 2g.$$

Generally, to represent a set of complex frequencies, their locations are plotted on a 2-D plane with f and g axes. The wave elements scattering widely in the plot, as the AR order changes, are considered noise. It is also possible to identify some wave elements densely populated on the theoretical frequency lines that remain mainly stable as the AR order changes. They are considered dominant spectral components [131]. An example of frequency-growth rate domain for an infrasound event recorded by EBEL station is reported in Fig. 31. Therefore, in summary, the spectral features of an infrasonic event can be described by the two parameters Q and f. Further, in addition to frequency and quality factor, the third feature used to characterize the infrasound events is the peak-to-peak amplitude, depending on both distance source-station and energy of the infrasonic source.

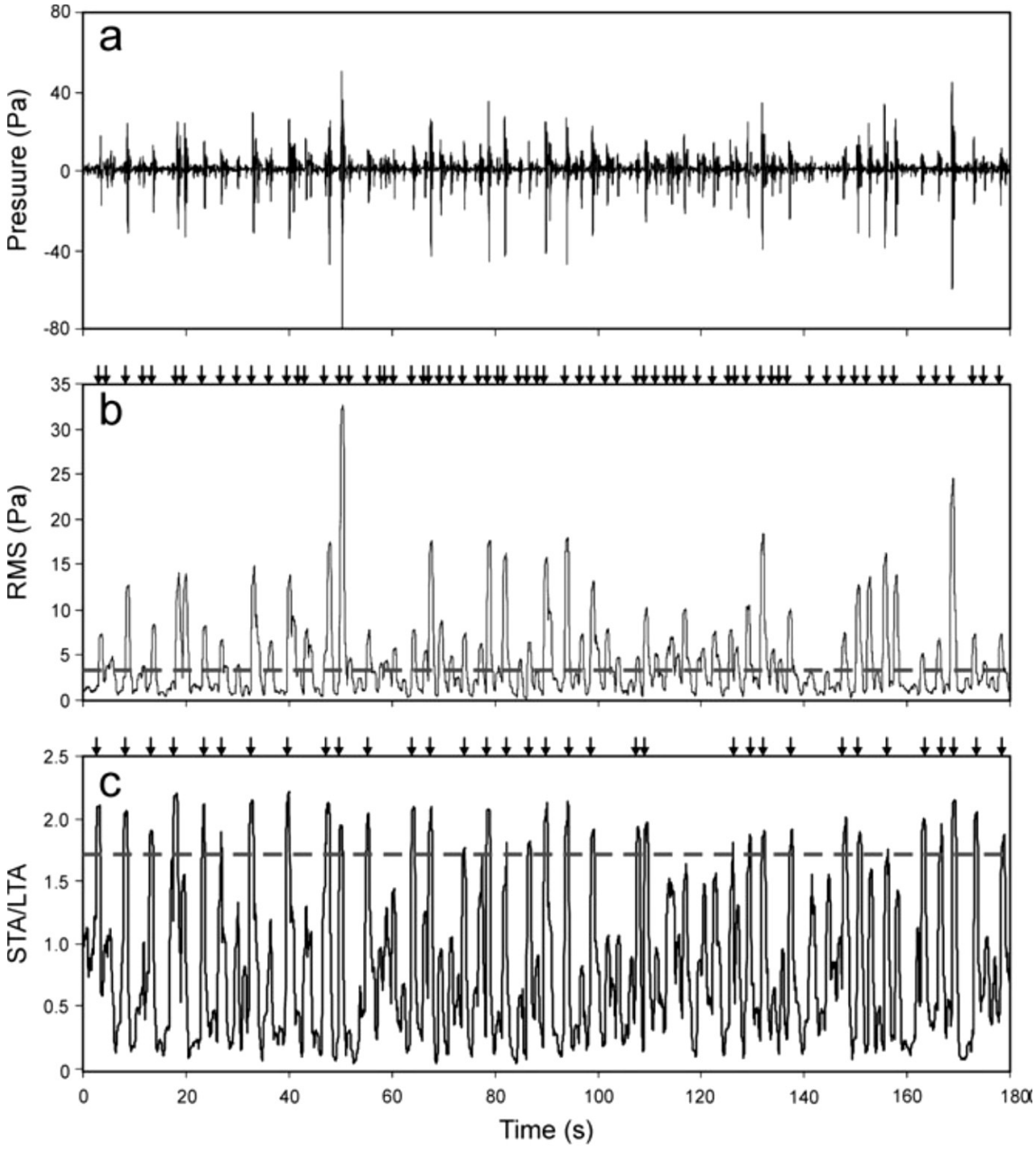


Figure 31 – *(a) Three-minute long infrasound signal recorded by EBEL station, (b) corresponding rms envelope (black line) calculated by using a moving window of 0.7 s and (c) STA / LTA values. The horizontal grey dashed line in (b) indicates the detection threshold calculated by a percentile value of 5 multiplied by 5. The horizontal grey dashed line in (c) indicates the detection threshold fixed at 1.7. The arrows at top of (b,c) indicate the onset time of the detected events.*

6.1.5 Semblance Algorithm

The location of the source of the infrasonic events, generally coinciding with active vents, is of great importance for volcanic monitoring. Therefore, different location

techniques, generally based on grid searching procedures, were developed [82, 21, 18, 71]. The semblance technique is based on the semblance function that is a measure of the similarity of multichannel data [73]. For infrasonic events this method applies a 2-D grid searching procedure over a surface covering the summit area and coinciding with the topographic surface. The infrasonic source is assumed to be in each node of the grid, and for each node the theoretical travel times at the sensors are first calculated. Then, infrasonic signals at different stations are delayed and compared by the semblance function. Finally, the source is located in the node where the delayed signals show the largest semblance value. Therefore, the semblance function is assumed representative of the probability that a node has to be the source location (further details about the method are reported in [71]).

6.1.6 Learning phase

In the proposed system, the learning phase merges together results of clustering and classification analysis (Fig. 32). The techniques described in Section 5.1 and 5.2 are applied on infrasound event features together with geophysical information used to "label" the recognized clusters. About 665 events, recorded during 2007 September–November at EBEL station, were detected and filtered in frequency range 0.5–5 Hz. The feature extraction from the detected events was performed by Sompi method (Section 5.2) using 2-s long windows of infrasonic signal recorded at EBEL station and AR order equal to two. The sharply monochromatic nature of the investigated signals justifies the choice of this low order [53]. Frequency and quality factor of the events, together with peak-to-peak amplitude, constituted the feature space and are plotted in Fig. 33. Then, to discover clusters in this space, "data clustering" techniques based on DBSCAN algorithm (Section 5.1) were applied. Using such an algorithm we found three main clusters (called cluster 1, 2 and 3) and other outlier points that can be considered as noise (Fig. 34). Points belonging to each cluster are related to infrasonic events that were located using Semblance location method (Section 5.3). In accordance with [118], during 2007 September–November, two infrasonic sources were found, NEC and SEC. In particular, a cluster was composed of events generated by NEC (cluster 1) and the other two by SEC.

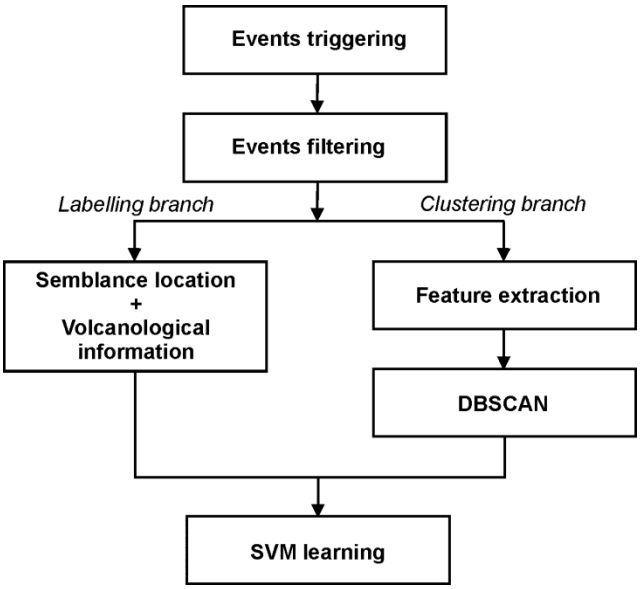


Figure 32 – *Scheme of the learning system*

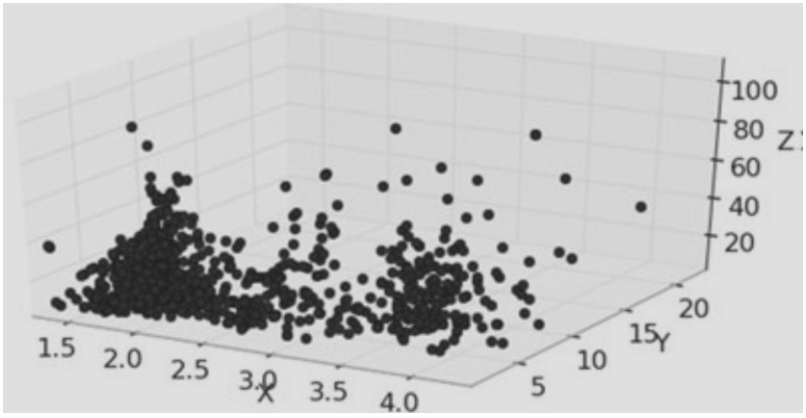


Figure 33 – *Feature space with frequency, quality factor and peak-to-peak amplitude of the infrasound events recorded at EBEL station during September – November 2007*

Such last two clusters were related to different kinds of explosive activity at SEC. In particular, the events belonging to cluster 3 were coincident with 'more visible' explosions, characterized by a relevant presence of ash, whereas the events of cluster 2 were hardly visible in the monitoring video-camera recordings [118]. Features clustering together with labels provide the patterns for SVM learning process.

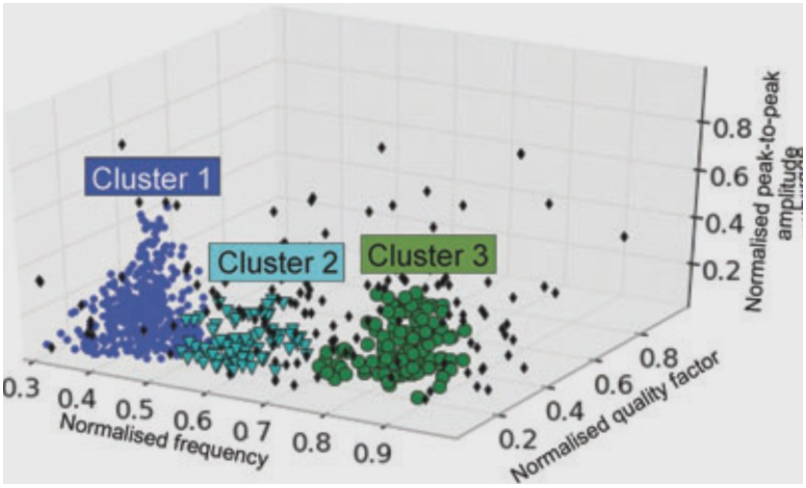


Figure 34 – *Clustering of the feature space reported in Fig. 10. The clusters are indicated with blue (cluster 1) and green circles (cluster 3) and light green triangles (cluster 2), the outliers with black diamonds*

As mentioned in Section 5.2, optimization of parameters C (regularization parameter) and σ (radial basis function kernel parameter) is a key step in SVM learning because their values determine classification performance [121]. As a consequence, model selection is applied with the aim of finding the best pair of parameters C and σ that minimizes the error rate estimated as the ratio between misclassified and hit patterns. These parameters can be chosen using a cross-validation (CV) approach [130], which is a statistical method for learning algorithms evaluation and model selection. In particular, in K-fold CV the available data set is partitioned into K subsets or 'folds': K–1 folds are used for SVM learning purpose, and the remaining fold for model validation (Fig. 35). Thus, K iteration of learning and validation are performed and for each ith iteration the training process is carried out using $K-1$ folds and the i^{th} fold for validation (Fig. 35). All SVM training algorithms are computed using one-against-all method (see Section 5.2). Since we worked on a small data set, a simple exhaustive grid search can be performed [132]. In particular, C was systematically changed in the range [1–100] with a step of 10, σ in the range [0.1–10] with a step of 0.5 and a K-fold CV with $K = 10$ was used. The entire procedure can be summarized as follows (Fig. 36): (1) a grid value of C and σ is defined; (2) for each pair of C and σ values, a mean error rate is computed averaging the error rate values obtained by the K SVM models; (3) the pair of C and σ with the minimum error rate is selected; (4) such a pair is used to

train the final SVM model with the whole data set, comprising all the K folds. Here, the best parameter values were C = 1 and σ = 0.1, for which mean CV error minimized to 0.6 per cent.

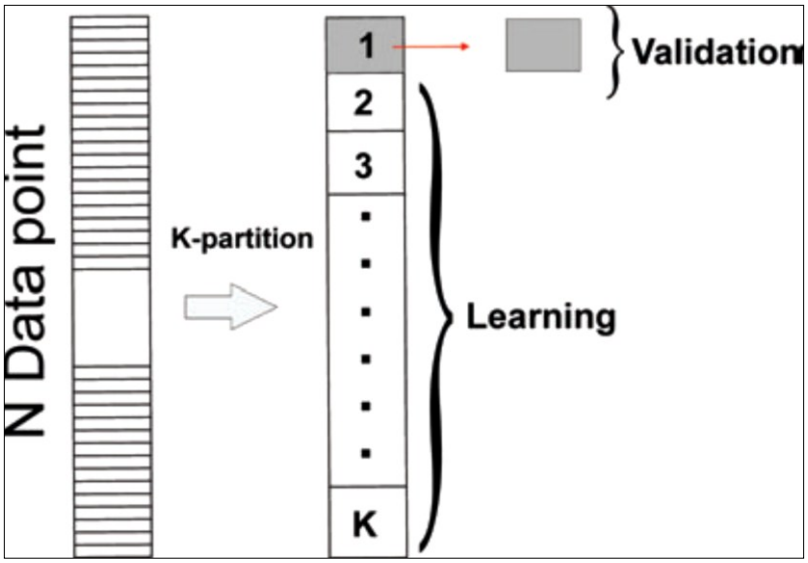


Figure 35 – *Basic scheme of K-Fold Cross-validation (see Section 5.3 for details)*

6.1.7 Testing phase and final system

To verify the system, the trained SVM is tested by classifying new unknown infrasonic events and then assigning them to their source crater. The reliability is verified using events not analyzed during the previous learning phase (Section 5.3). To this end, a new test data set of about 610 events, recorded during 2 months, 2007 August and December, was used and labeled by location algorithm based on semblance method.

Moreover, the events belonging to cluster 2 and cluster 3 were labeled using information related to the intensity of the explosive activity [118]. The quality of classification is quantified using confusion matrix (Table 2), where each column represents the instances in the predicted class (based on the SVM model), while each row represents the instances in the actual class (based on the previously attributed labels). Thus, the entries on the diagonal count the events in which prediction agrees with known labels, whereas the other entries the misclassified events. 63 elements were wrong assigned, providing an error rate of about 11.97 per cent. Misclassifications were mostly concentrated in the second and third classes that are related to the two different

explosion activities of SEC crater. Indeed, such a distinction is qualitative and not clear cut, hence many halfway events can be misclassified. If we do not take into account the distinction between clusters 2 and 3, and consider them as a single cluster, the error decreases to 5.25 per cent. Finally, the proposed system can be summarized as follows (Fig. 37): (i) triggering procedures is performed on buffer of acquired signal; (ii) then, if events are found, the system evaluates whether there is a sufficient number of stations for semblance location algorithm; (iii) if the number of stations is not sufficient, alternative 'single station' location is performed by extracting signal features and classifying them using the trained SVM.

		Predicted		
		Cluster 1	Cluster 2	Cluster 3
ACTUAL	Cluster 1	**476**	9	6
	Cluster 2	9	**15**	8
	Cluster 3	8	33	**46**

Table 2. *Confusion matrix calculated in the testing phase. Each column represents the instances in the predicted class (based on the SVM model), while each row represents the instances in the actual class (based on the previously attributed labels). Thus, the entries on the diagonal (bold numbers) count the events in which prediction agrees with known labels, whereas the other entries the misclassified events.*

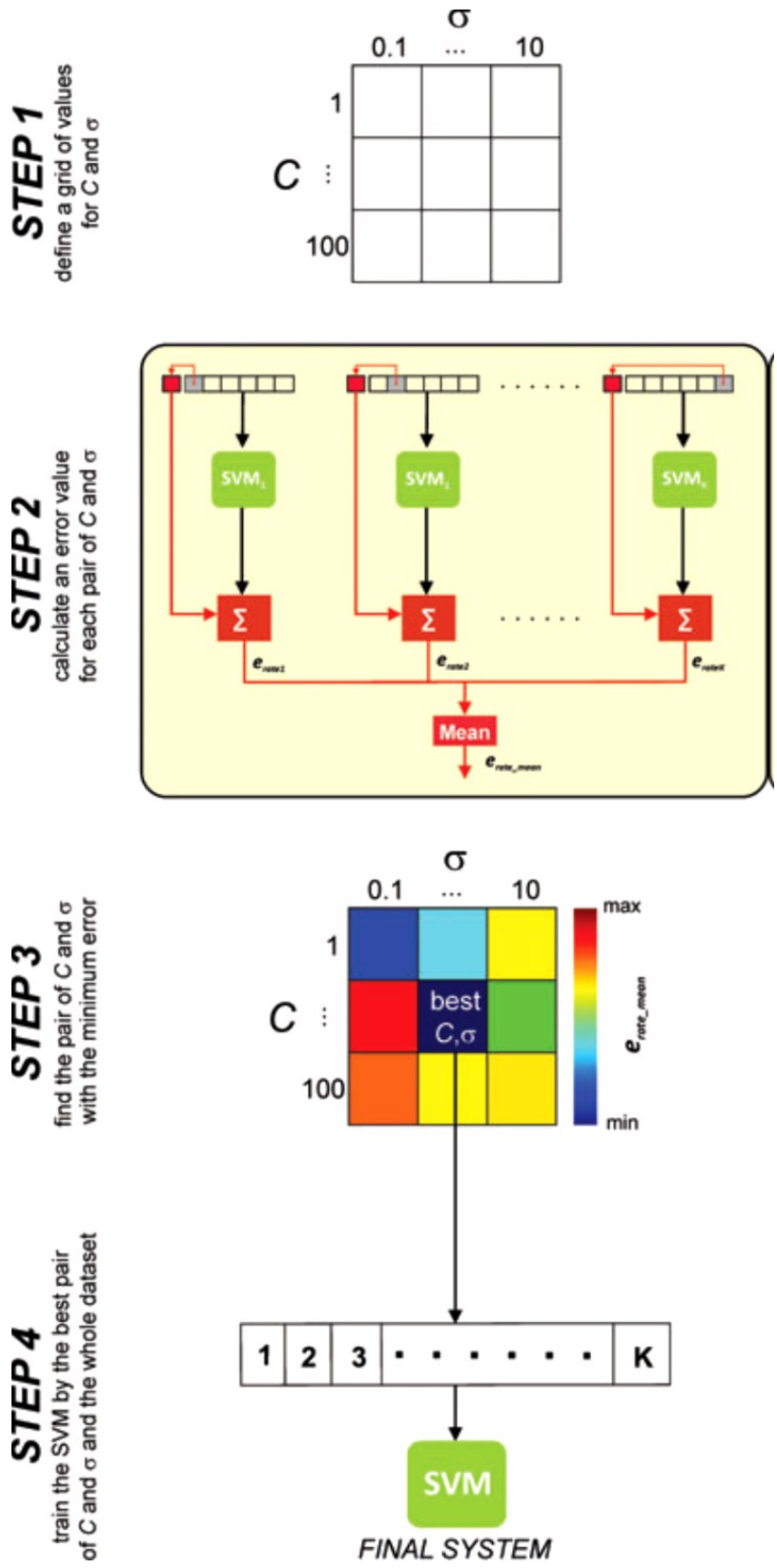


Figure 36 - *Best SVM model selection using K-Fold Cross-validation (see Section 5.3 for details)*

It is also worth noting that SVM classifier is also applied offline on localizable events to evaluate its performance in distinguishing NEC events (cluster 1) from SEC events (clusters 2 and 3). In this case, events belonging to clusters 2 and 3 are simply considered SEC events and then labeled based on the source vent, with no further distinction depending on the type of explosive activity. This task is carried out by comparing the results of the classifier with the location parameters provided by the semblance algorithm. By the inspection of the obtained error rate, a new clustering execution is necessary when classification of new signals is not aligned with that of infrasonic network classifier. This may be caused by the creation of a new active vent or by the changing activity of a pre-existing vent; in such a case the system must be updated.

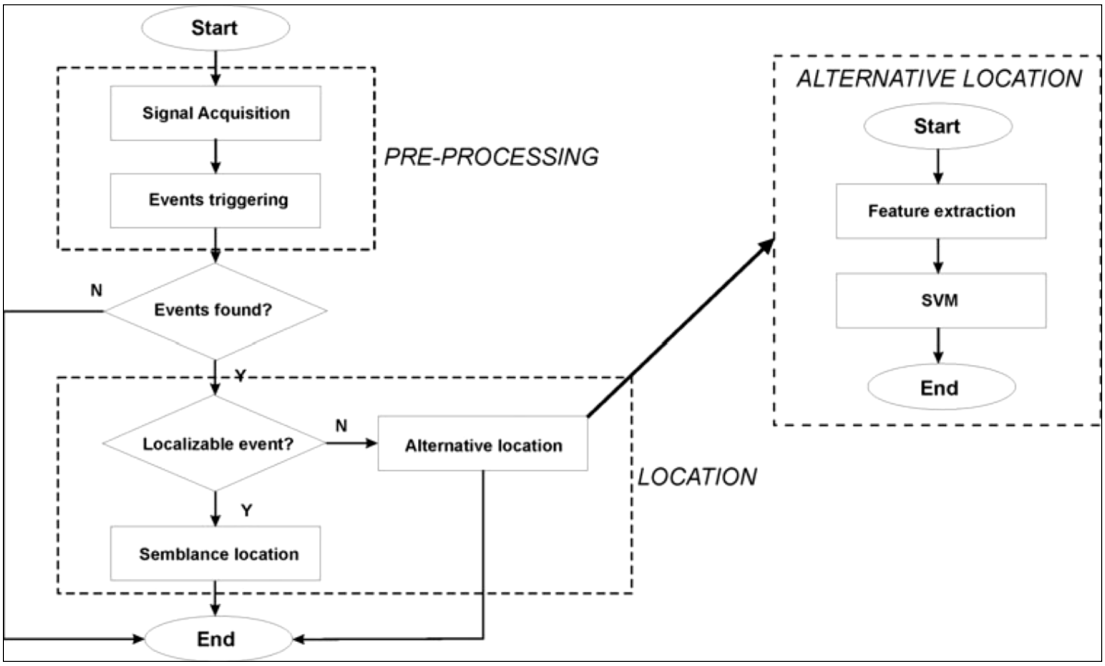


Figure 37 – *Flow chart of the proposed location system*

6.2 Data mining in image processing: Stromboli as a case of study

For a scientist, an image is an information carrier, but the information of interest may not be perceivable with the human eye only. It may corrupted by noise or simply be tied up with details and information of no interest so that it is not useful for purposes. In such cases we may have to develop techniques to extract that information from the image. This area of image processing is called image analysis.

The first and most the important step in image analysis is to segment the image (often referred to as boundary estimation, boundary detection or object localization). Segmentation subdivides an image into its constituents parts. Segmentation algorithms for monochrome images generally are based on one of two basic properties of grey-level values: discontinuity and similarity. The first method partitions an image based on abrupt changes in grey-level (isolated points, lines and edges) while the second method is based on thresholding, region growing and region splitting and merging [109].

The problem with these methods is that the edges found do not necessarily correspond to the boundaries of objects. With the exception of high-quality images from controlled environments, these methods produce spurious edges and gaps. The limitation of these methods is due to their complete reliance on information contained in the local neighborhood of pixel in the image. They ignore both model-based information and higher order organization of the image.

Another problem associated with these methods is edge grouping. After extracting edges from the image, they have to be grouped in order to determine boundaries. This is usually done by first associating edge elements into edge segments and then by associating segments into boundaries. These methods often resort to arbitrary interpolation in order to complete boundary gaps.

In this section we illustrate two methods for boundaries detection in image: active shape model, called Snake, and Morphological image analysis method. Now we introduce these models that we will be using all along this work.

6.2.1 Morphological image analysis

Morphological image analysis (MIA) [110], is based on same morphological operations for extracting image components that are useful for representation and description. These apply a structuring element to an input image, creating an output image of the same size.

In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. This technique is a set-theoretic method of image analysis providing a quantitative description of geometrical structures. Morphology can provide boundaries of objects, their skeletons and their convex hulls. Generally speaking most of morphological operations are based on simple expanding and shrinking operations. The primary application of morphology occurs in binary images; it can also be useful on range images. The two basic morphological set transformations are erosion and dilation. These transformations involve the interaction between an image A (the object of interest) and a structuring set B, called the structuring elements. Typically the structuring element B is a circular disk in the plane, but it can be any shape. A basic scheme of MIA process is reported in figure 38.

MIA tool can be divided in different steps:

- ✓ Identification threshold and converting image in a binary image (black and white);
- ✓ Fill image regions and holes;
- ✓ Remove small object from an image while preserving the shape and size of lager objects in the image: this technique is called morphological opening;
- ✓ Extraction contour;
- ✓ Dilation: adds pixels to the boundaries of object in an image;

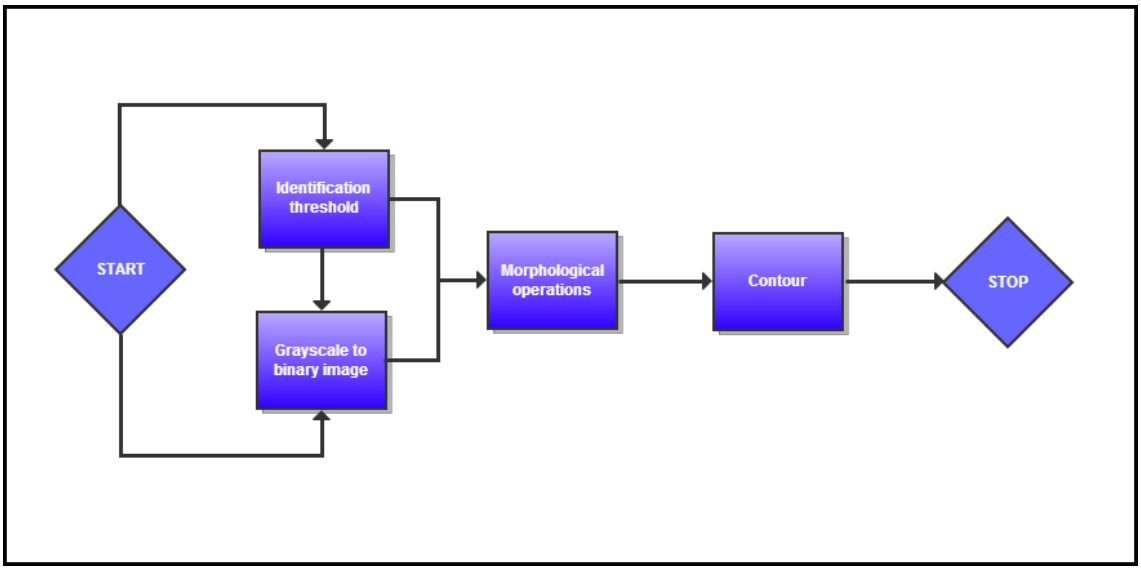


Figure 38 – *Overview of process*

All these operation are based on a process of convolution between a binary image and a structuring element, called also mask or kernel. Generally, this is a square that have an uneven number of pixel of sides (3x3, 5x5…), and which origin is defined how the pixel correspondent to the center of the square (fig 39).

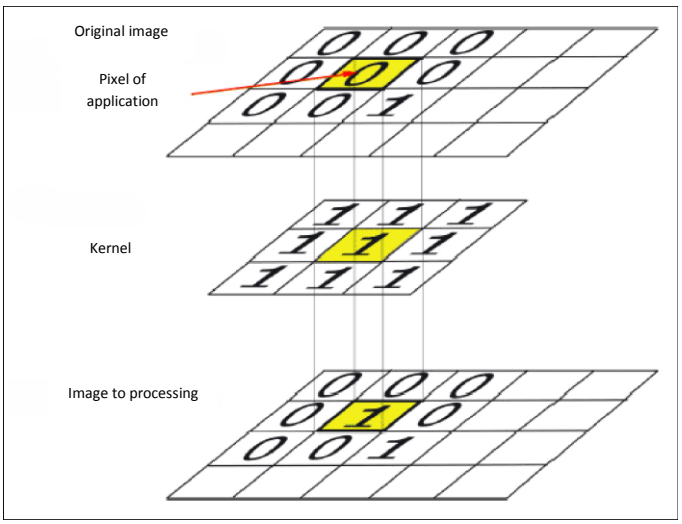


Figure 39 – *Operation of dilation with a square kernel of size 3x3*

Morphological opening is an erosion followed by a dilation, using the same structuring elements for both operations. The opening operation has the effect of removing objects that cannot completely contain the structuring element. The extraction of contour

constitutes the fulcrum of the algorithm, because provide for determine the shape of the objects.

6.2.2 Thermal Surveillance at Stromboli volcano

The explosive activity at Stromboli volcano is routinely analyzed by *live* monitoring cameras, in order to follow the eruptive dynamics. Since the style of strombolian activity can be monitored using the frames acquired by video surveillance, an efficient system able to require information from a huge amount of frames is needed. One of aims of this thesis was the development of a novel system, capable of fast data retrieval based on similarity concepts. In the light of it, an indexing algorithm was developed. As explained in chapter 3, basic concept is searching for elements of a set being close respect to a query element, according to a similarity criterion.

Figure 40 – *Map of thermal cameras monitoring the activity of Stromboli Volcano in Stromboli Island (Aeolian Islands, Italy). There are 3 thermal cameras, located at Pizzo (S400 and S800) and Vancori (SVAN)*

6.2.3 Mapping Explosions into a Metric Space

The developed system is shown in figure 41. Basically, each video frame is processed using morphological processing techniques to extract the image area of the explosion. Each closed curve, which represents the contour of the explosion, is approximated using an ellipse. The approximating ellipses undergoes a parameterization process, using six coefficients of its general equation, and then the evolution over time of each coefficient is expressed by the coefficients of their linear regression. All the coefficients of the linear regression, together with the six coefficients of the ellipse with respect to the maximum expansion of the explosion, are the characteristics that describe the explosive sequence (Fig. 42). These characteristics are the metric space in which the similarity between objects can be evaluated by calculating their "distances". This approach suffers from an inherent problem related to the number of distances calculated on a huge amount of data (in fact feature vectors are immersed in a 18-dimensional space).

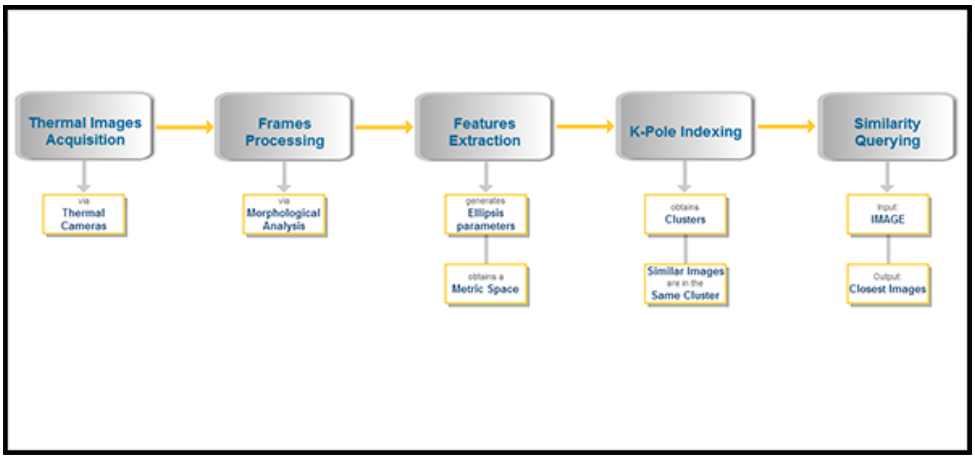


Figure 41 - *Building blocks of the framework*

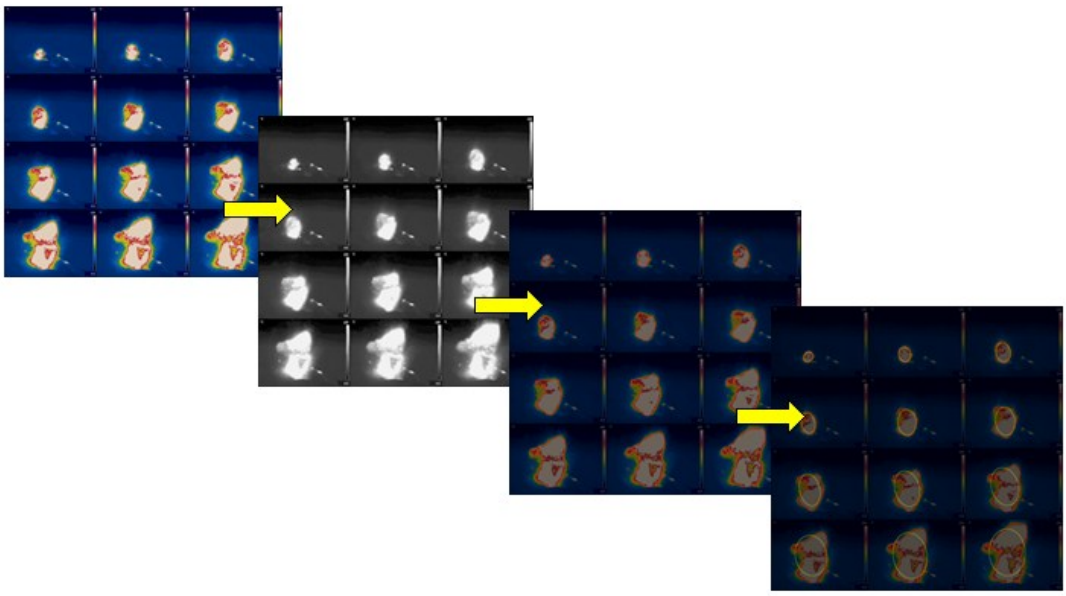


Figure 42 - *Image Processing output for each step*

Once features space is defined, a classification task can be applied, with a metric distance function. This approach suffers from an intrinsic problem related to the number of distance to be computed on a huge amount of data. To overcome this drawback, we built a K-Pole tree, introduced in chapter 3, using a reasonable threshold splitting, in order to obtain a "single-layer" indexing. Looking at the leaf level of the tree (where clusters are located), we find many clusters with an huge amount of frames, and a very small collection of clusters containing a few frames. Smallest clusters will contain the most significant data, since their elements are the most dissimilar from the other objects of the metric space, that reference "common" objects. The small groups of object represents "special" scenarios (major explosions, video noise or software rebooting). Such an indexing was able to "group" the different types of explosion related activities with reference information. For instance, if we have a known image sequence showing an explosion event, we are able to easily and quickly find all sequences which contain an explosion similar to it. Using the same technique, image disturbance similar to those previously known can be recognized, generating an automatic alert to the maintenance personnel of thermal cameras.

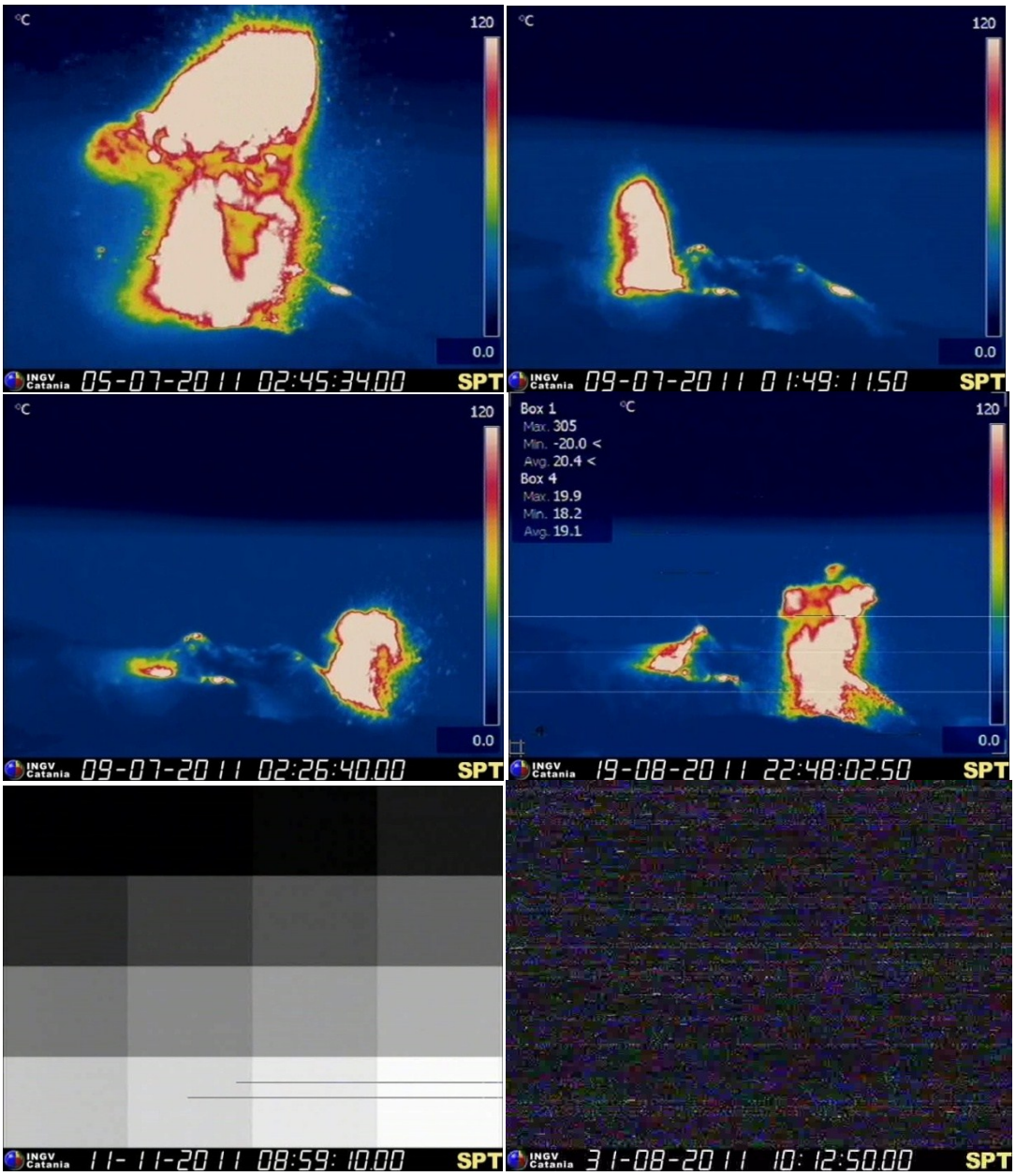


Figure 43 - *Different kinds of scenario that can be automatically detected using our framework*

Conclusions

This thesis is focused on the applications of data mining algorithms on geophysical data provided by the Istituto Nazionale di Geofisica e Vulcanologia (INGV), Osservatorio Etneo.

The concept of indexing and searching in metric space was introduced and the usefulness of appropriate data structure for similarity search with an analysis of different kinds of similarity/distance measures was explained. For practical purposes, a novel indexing structure, called K-*Pole*, was introduced. Such a structure, as showed in experimental analysis, is more efficient than other indexing algorithms because it calculates only a fraction of the distances needed to absolving proximity queries.

Since at active volcanoes the detection and location of explosive activity is generally obtained by video cameras and thermal sensors [114], a framework based on *k-pole* algorithm was developed to obtain a fast thermal image database indexing. Such a system is able to provide several kinds of information such as number of clusters and centroids, data structure of intercluster distance, object distances and so on. Using this algorithm, an entire database of thermal images recorded by a Stromboli monitoring video camera was grouped into clusters with the same characteristics. In the light of it, a basic image query such as the nearest elements closets to a given element, can be performed with a speed time much lower than standard database. In particular, the performed analysis showed that clusters with less elements (frames) will contain the most significant data, since those elements are the "farthest" objects from the center of the metric space, that contains "common" objects. Farthest objects represent unusual scenarios (major explosions, video noise or software rebooting). Furthermore, using the same approach, video noise similar to those previously known can be recognized, hence alerting, in a totally automatic way, the maintenance personnel of thermal cameras.

However, the efficiency of video cameras and thermal sensors is severely reduced or inhibited in case of poor visibility caused by clouds or gas plumes. In these cases, the detection and characterization of explosive activity by infrasound is very useful [117] and some techniques, based on infrasound signals recorded by arrays or networks, were developed to locate the source of this signal and therefore the active vent [82]. All these techniques require that most of the stations work properly and that the noise level is

low. Unfortunately, sometimes during winter season, because of bad weather conditions, the possible lack of signals from some summit stations prevents applying the aforementioned standard location algorithms. For this reason, an automatic system, using the features of infrasound waveforms recorded by just one station, was developed. Such a system, whose learning phase is based on clustering (DBSCAN) and classification techniques (SVM), together with geophysical information, was applied on infrasound signals recorded by the Etna monitoring network. After the training phase this system automatically allows recognizing the active vent, with no location algorithm and by using only a single station with a success of ~95 per cent.

In conclusion, this thesis demonstrates the importance of the data mining techniques applied for volcano monitoring purpose.

References

[1] P. Apers, H. Blanken, and M. Houtsma. Multimedia Databases in Perspective. Springer, 1997.

[2] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. In Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94), pages 573-583, 1994.

[3] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. ACM Computing Surveys, 23(3), 1991.

[4] R. Baeza-Yates. Searching: an algorithmic tour. In A. Kent and J. Williams, editors, Encyclopedia of Computer Science and Technology, volume 37, pages 331-359. Marcel Dekker Inc., 1997.

[5] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In Proc. 5th Combinatorial Pattern Matching (CPM'94), LNCS 807, pages 198-212, 1994.

[6] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In Proc. 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98), pages 14-22. IEEE CS Press, 1998.

[7] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, 1999.

[8] J. Bentley. Multidimensional binary search trees used for associative searching. Comm. of the ACM, 18(9):509-517, 1975. 119

[9] J. Bentley. Multidimensional binary search trees in database applications. IEEE Trans. on Software Engineering, 5(4):333-340, 1979.

[10] J. Bentley, B. Weide, and A. Yao. Optimal expected-time algorithms for closest point problems. ACM Trans. on Mathematical Software, 6(4):563-580, 1980.

[11] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: an index structure for high-dimensional data. In Proc. 22nd Conference on Very Large Databases (VLDB'96), pages 28-39, 1996.

[12] B. Bhanu, J. Peng, and S. Qing. Learning feature relevance and similarity metrics in image databases. In Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries, pages 14-18, Santa Barbara, California, 1998. IEEE Computer Society.

[13] A. Del Bimbo and E. Vicario. Using weighted spatial relationships in retrieval by visual contents. In Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries, pages 35-39, Santa Barbara, California, 1998. IEEE Computer Society.

[14] Cantone D, Ferro A, Pulvirenti A, Recupero Recupero D, and Shasha D, Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces, IEEE Trans. Knowl. Data Eng., vol. 17, no. 4, pp. 535–550, 2005.

[15] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In Proc. ACM SIGMOD International Conference on Management of Data, pages 357-368, 1997. Sigmod Record 26(2).

[16] S. Brin. Near neighbor search in large metric spaces. Proceedings of the 21th International Conference on Very Large Data Bases, pages 574-584, 1995.

[17] Johnson, J.B. & Lees, J.M., 2000. Plugs and chugs: seismic and acoustic observations of degassing explosions at Karymsky, Russia and Sangay, Ecuador, J. Volc. Geotherm. Res., 101, 67–82.

[18] Johnson, J.B., Lees, J. & Varley, N., 2010. Characterizing complex eruptive activity at Santiaguito, Guatemala using infrasound semblance in networked arrays, J. Volc. Geotherm. Res., 199, doi:10.1016/j.jvolgeores.2010.08.005.

[19] W. Burkhard and R. Keller. Some approaches to best-match file searching. Comm. of the ACM, 16(4):230-236, 1973.

[20] M. La Cascia, S. Sethi, and S. Sclaroff. Combining textual and visual cues for content-based image retrieval on the world wide web. In Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries, pages 24-28, Santa Barbara, California, 1998. IEEE Computer Society.

[21] Jones, K.R., Johnson, J., Aster, R., Kyle, P.R. & McIntosh, W.C., 2008. Infrasonic tracking of large bubble bursts and ash venting at Erebus volcano, Antarctica, J. Volc. Geotherm. Res., 177, doi:10.1016/j.jvolgeores.2008.02.001.

[22] E. Chavez and J. Marroquìn. Proximity queries in metric spaces. In R. Baeza-Yates, editor, Proc. 4th South American Workshop on String Processing (WSP'97), pages 21-36. Carleton University Press, 1997.

[23] E. Chàvez, J. Marroquìn, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In Proc. String Processing and Information Retrieval (SPIRE'99), pages 38-46. IEEE CS Press, 1999.

[24] E. Chàvez, J. Marroquìn, and G. Navarro. Overcoming the curse of dimensionality. In European Workshop on Content-Based Multimedia Indexing (CBMI'99), pages 57-64 , 1999. To appear. `ftp://-garota.fismat.umich.mx/pub/users/elchavez/fqa.ps.gz.`

[25] L. Childs. A concrete Introduction to Higher Algebra. Springer-Verlag, 1995.

[26] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In Proc. of the 23rd Conference on Very Large Databases (VLDB'97), pages 426-435, 1997.

[27] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98), 1998.

[28] Joswig, M., 1990. Pattern recognition for earthquake detection, Bull. seism. Soc. Am., 80(1), 170–186.

[29] Kay, S.M., 1988. Modern Spectral Estimation, Theory and Application, Prentice-Hall, Englewood Cliffs, NJ.

[30] L. Devroye. A Course in Density Estimation. Birkhauser, 1987.

[31] R. Duda and P. Hart. Pattern Classification and Scene Analysis. Wiley, 1973.

[32] C. Faloutsos and K. Lin. Fastmap: a fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. ACM SIGMOD Record, 24(2):163-174, 1995.

[33] A. Faragò, T. Linder, and G. Lugosi. Fast nearest-neighbor search in dissimilarity spaces. IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(9):957-962, 1993.

[34] W. Frakes and R. Baeza-Yates, editors. Information Retrieval: Data Structures and Algorithms. Prentice-Hall, 1992.

[35] V. Gaede and O. Gunther. Multidimensional access methods. ACM Computing Surveys, 30(2):170-231, 1998.

[36] A. Guttman. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD International Conference on Management of Data, pages 47-57, 1984.

[37] Kecman, V., 2001. Learning and Soft Computing. Support Vector Machines, Neural Networks, and Fuzzy Logic Models, The MIT Press, Cambridge.

[38] Kohler, A., Ohrnberger, M. & Scherbaum, M., 2009. Unsupervised feature selection and general pattern discovery using Self-Organizing Maps for gaining insights into the nature of seismic wavefields, Comput. Geosci., 35, 1757–1767, doi:10.1016/j.cageo.2009.02.004.

[39] A. Jain and R. Dubes. Algorithms for Clustering Data. Prentice-Hall, 1988.

[40] Kumazawa, M., Imanishi, Y., Fukao, Y., Furumoto, M. & Yamamoto, A., 1990. A theory of spectral analysis based on the characteristic property of a linear dynamic system, Geophys. J. Int., 101, 613–630.

[41] L. Micò, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. Pattern Recognition Letters, 17:731-739, 1996.

[42] L. Micò, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. Pattern Recognition Letters, 15:9-17, 1994.

[43] J. Munkres. Topology, A First Course. Prentice-Hall, 1975.

[44] G. Navarro. Searching in metric spaces by spatial approximation. In Proc. String Processing and Information Retrieval (SPIRE'99), pages 141-148. IEEE CS Press, 1999.

[45] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. IEEE Trans. on Pattern Analysis and Machine Intelligence, 19(9):989-1003, 1997.

[46] J. Nievergelt and H. Hinterberger. The grid file: an adaptable, symmetric multikey file structure. ACM Trans. on Database Systems, 9(1):38-71, 1984.

[47] Langer, H., Falsaperla, S., Masotti, M., Campanini, R., Spampinato, S. & Messina, A., 2009. Synopsis of supervised and unsupervised pattern classification techniques applied to volcanic tremor data at Mt Etna, Italy, Geophys. J. Int., 178, doi:10.1111/j.1365–246X.2009.04179.x.

[48] G. Salton and M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[49] D. Sankoff and J. Kruskal, editors. Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison. Addison-Wesley, 1983.

[50] D. Shasha and T. Wang. New techniques for best-match retrieval. ACM Trans. on Information Systems, 8(2):140-158, 1990.

[51] M. Shapiro. The choice of reference points in best-match file searching. Comm. of the ACM, 20(5):339-343, 1977.

[52] R. Sutton and A. Barto. Reinforcement Learning : an Introduction. MIT Press, 1998.

[53] Lesage, P., 2008. Automatic estimation of optimal autoregressive filters for the analysis of volcanic seismic activity, Nat. Hazards Earth Syst. Sci., 8, 369–376.

[54] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. Information Processing Letters, 40:175-179, 1991.

[55] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters, 4:145-157, 1986.

[56] M. Waterman. Introduction to Computational Biology. Chapman and Hall, 1995.

[57] D. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, University of California, La Jolla, California, July 1996.

[58] A. Yao. Computational Geometry, chapter 7, pages 345-380. Elsevier Science, 1990. J. Van Leeuwen, editor.

[59] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93), pages 311-321, 1993.

[60] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In DIMACS Implementation Challenge, ALENEX'99, Baltimore, MD, 1999.

[61] Marchetti, E., Ripepe, M., Ulivieri, G., Caffo, S. & Privitera, E., 2009. Infrasonic evidences for branched conduit dynamics at Mt. Etna volcano, Italy, Geophys. Res. Lett., 36, L19308, doi:10.1029/2009GL040070.

[62] Marple, S.L. 1987. Digital Spectral Analysis with Applications, Prentice Hall, Englewood Cliffs.

[63] Mars, J., Lacoume, J.L., Mari, J.L. & Glangeaud, F., 2004. Traitement du Signal Pour G´eologues et g´eophysiciens, Vol. 3, Techniques avanc´ees, Technip.

[64] Masotti, M., Campanini, R., Mazzacurati, L., Falsaperla, S., Langer, H. & Spampinato, S., 2008. TREMOrEC: a software utility for automatic classification of volcanic tremor, Geochem. Geophys.Geosyst., 9(1), Q04007, doi:10.1029/2007GC001860.

[65] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. An efficient algorithm for the approximate median selection problem. Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC 2000), volume 1767 of Lecture Notes in Computer Science, Springer-Verlag, pages 226-238, 2000.

[66] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. ACM Transaction on Database Systems, 24(3):361-404, 1999.

[67] Mark S. Daskin. Network and Discrete Location: Models Algorithms and Applications. Wiley, New York, 1995.

[68] Mark S. Daskin. A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. Communications of the Operations Research Society of Japan, 45:9:428-436, 2000.

[69] I. Calantari and G. McDonald. A data structure and an algorithm for the nearest point problem. In IEEE Transaction on Software Eng., 9(5), 1983.

[70] D. Cantone, G. Cincotti, A. Ferro, and A. Pulvirenti. An efficient algorithm for the 1-median problem. Preprint University of Catania, 2003.

[71] Montalto, P., Cannata, A., Privitera, E., Gresta, S., Nunnari, G. & Patanè, D., 2010. Towards an automatic monitoring system for infrasonic events at Mt. Etna: strategies for source location and modelling, Pure appl. Geophys., 167, doi:10.1007/s00024–010-0051-y.

[72] E. Chavez and G. Navarro. A probabilistic spell for the curse of dimensionality. Proceedings of Third Workshop on Algorithm Engineering and Experimentation (ALENEX01) volume 2153 of Lecture Notes in Computer Science, pages 147-160, 2001.

[73] Neidell, N. & Taner, M.T., 1971. Semblance and other coherency measures for multichannel data. Geophysics, 36, 482–497, doi:10.1190/1.1440186.

[74] B. Chazelle. Computational geometry: a retrospective. Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pages 75-94, 1994.

[75] A. Wai chee Fu, P. M.S. Chan, Y.-L. Cheung, and Y.S.Moon. Dynamic vptree indexing for n-nearest neighbor search given pair-wiewdistances. VLDB Journal, pages 311-321, 1999.

[76] Noble, W.S., 2004. Support vector machine applications in computational biology, in Kernel Methods in Computational Biology, pp. 71–92, eds Scholkopf, B., Tsuda, K., Vert, J., The MIT Press, Cambridge, MA.

[77] P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. Proceedings of the IEEE 16th International Conference on Data Engineering, pages 244-255, 2000.

[78] Sourour Elloumi, Martine Labbe, and Yves Pochet. New formulation and resolution method for the p-center problem. 2001.

[79] Ohrnberger, M., 2001. Continuous automatic classification of seismic signals of volcanic origin at Mt. Merapi, Java, Indonesia, PhD thesis, University of Potsdam, 158 pp.

[80] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining, 1996.

[81] T. Feder and D.H. Greene. Optimal algorithms for approximate clustering. Proceedings of the 20st Annual ACM Symposium on Theory of Computing, pages 434-444, 1988.

[82] Ripepe, M. & Marchetti, E., 2002. Array tracking of infrasonic sources at Stromboli volcano, Geophys. Res. Lett., 29(22), 2076, doi:10.1029/2002GL015452.

[83] Ripepe, M., Poggi, P., Braun, T. & Gordeev, E., 1996. Infrasonic waves and volcanic tremor at Stromboli, Geophys. Res. Lett., 23, 181–184.

[84] A. Gersho and R. Gray. Vector Quantization and Signal Compression. Kluwer Academic, 1991.

[85] T.F. Gonzalez. Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38:293-306, 1985.

[86] Rowe, C.A., Aster, R.C., Kyle, P.R., Dibble, R.R. & Schlue, J.W., 2000. Seismic and acoustic observations at Mount Erebus Volcano, Ross Island, Antarctica, 1994–1998, J. Volc. Geotherm. Res., 101, 105–128.

[87] Ruiz, M.C., Lees, J.M. & Johnson, J.B., 2006. Source constraints of Tungurahua volcano explosion events, Bull. Volcanol., 68, 480–490.

[88] S. Har-Peled. A practical approach for computing the diameter of a point set. Proceedings of the 17th Symposium on Computational Geometry, pages 177-186, 2001.

[89] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. Journal of the ACM, 32:130-136, 1985.

[90] Vapnik, V.N., 1998. Statistical Learning Theory, John Wiley & Sons, New York.

[91] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pages 604-613, 1998.

[92] Vergniolle, S. & Brandeis, G., 1994. Origin of the sound generated by Strombolian explosions, Geophys. Res. Lett., 21, 1959–1962.

[93] Weston, J. & Watkins, C., 1999. Multi-class support vector machines, presented at the Proceedings ESANN99, ed. M.Verleysen, Brussels,Belgium.

[94] R.T. Ng and J. Han. Clarans: a method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering, 14(5):1003- 1016, 2002.

[95] H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous bisector* trees - a tool for efficient partitioning of complex schemems of geometric objects. Data Structure and Efficient Algorithms volume 594 of Lecture Notes in Computer Science, pages 186-203, 1992.

[96] C.M. Procopiuc. Geometric techniques for clustering theory and practice. Ph.D. Dissertation - Duke University, 2001.

[97] Dorit S. Hochbaum, editor. Approximation Algorithms for NP-hard Problems. PWS publishing company, Boston, 1995.

[98] Withers, M., 1997. An automated local/regional seismic event detection and location system using waveform correlation, PhD thesis, New Mexico Tech., Socorro, NM.

[99] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A wavelet based clustering approach for spatial data in very large databases. VLDB Journal, 8(3-4):289-304, 2000.

[100] Taylan Ilhan and Mustafa Pinar. An efficient exact algorithm for the vertex p-center problem. 2001.

[101] Withers, M., Aster, R., Young, C., Beiriger, J., Harris, M., Moore, S. & Trujillo, J., 1998. A comparison of select trigger algorithms for automated global seismic phase and event detection, Bull. seism. Soc. Am., 88, 95–106.

[102] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. Proceedings of the ACM SIGMOD Conference on Management of Data, pages 103-114, 1996.

[103] N. Mladenovic, M. Labbe, and P. Hansen. Solving the p-center problem with tabu search and variable neighborhood search. 2000.
http://smg.ulb.ac.be/Preprints/Labbe00_20.html

[104] David B. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. Technical report, Ithaca, NY 14853, 1995.
http://citeseer.nj.nec.com/shmoys95computing.html

[105] Jurij Mihelic, Borut Robic. Approximation algorithms for the k-center problem: an experimental evaluation. University of Ljubljana.

[106] Alfredo Ferro, Rosalba Giugno, Alfredo Pulvirenti. Efficient boundary values generation in general metric spaces for software component testing. Università di Catania.

[107] F. Dehne and H. Nolteimer. Voronoi trees and clustering problems. Information Systems, 12(2):171-175, 1987.

[108] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering, 9(5), 1983.

[109] R.C. Gonzales and R.E. Woods. Digital Image Processing. Addison-Wesley, 1993.

[110] K. Michielsen, H. De Raedt, and T. Kawakatsu, "Morphological Image Analysis", in Computer Simulation Studies in Condensed-Matter Physics XIII, eds. D.P. Landau et al., Springer Proceedings in Physics, Volume 86 (Springer, Berlin), 87 - 91 (2000).

[111] Bustos B, Pedreira O, Brisaboa N. A dynamic pivot selection technique for similarity search. SISAP '08: proceedings of the first international workshop on similarity search and applications, pp 105–112, 2008.

[112] Behncke, B. & Neri, M., 2003. Cycles and trends in the recent eruptive behaviour of Mount Etna (Italy), Can. J. Earth Sci., 40, 1405–1411.

[113] Berkhin, P., 2002. Survey Of Clustering Data Mining Techniques, Accrue Software, San Jose, CA.

[114] Bertagnini, A., Coltelli, M., Landi, P., Pompilio, M. & Rosi, M., 1999. Violent explosions yield newinsights into dynamics of Stromboli volcano, EOS, Trans. Am. geophys. Un., 80, 633–636.

[115] Burges, C.J.C., 1998. A tutorial on support vector machines for pattern recognition, Data Min. Knowl. Discov., 2, 121–167.

[116] Cannata, A., Catania, A., Alparone, S. & Gresta, S., 2008. Volcanic tremor at Mt. Etna: inferences on magma dynamics during effusive and explosive activity, J. Volc. Geotherm. Res., 178, doi:10.1016/j.jvolgeores.2007.11.027.

[117] Cannata, A., Montalto, P., Privitera, E., Russo, G. & Gresta, S., 2009a. Tracking eruptive phenomena by infrasound: May 13, 2008 eruption at Mt. Etna, Geophys. Res. Lett., 36, doi:10.1029/2008GL036738.

[118] Cannata, A., Montalto, P., Privitera, E. & Russo, G., 2009b. Characterization and location of infrasonic sources in active volcanoes: Mt. Etna,

September–November 2007, J. geophys. Res., 114, doi:10.1029/2008JB006007.

[119] Cheng, J., Randall, A. & Baldi, P., 2006. Prediction of protein stability changes for single-site mutations using support vectormachines, Proteins, 62, 1125–1132, doi:10.1002/prot.20810.

[120] Davies, D.L. & Bouldin, D.W., 1979. A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell., 1, 224–227.

[121] Devos, O., Ruckebusch, C., Durand, A., Duponchel, L. & Huvenne, J.P., 2009. Support vector machines (SVM) in near infrared (NIR) spectroscopy: focus on parameters optimization and model interpretation, Phys. Chem. B, 113, 6031–6040.

[122] Di Grazia, G., Cannata, A., Montalto, P., Patanè,D., Privitera, E., Zuccarello, L. & Boschi, E., 2009. A new approach to volcano monitoring based on 4D analyses of seismo-volcanic and acoustic signals: the 2008 Mt. Etna eruption, Geophys. Res. Lett., 36, doi:10.1029/2009GL039567.

[123] Ester, M., Kriegel, H.P., Sander, J. & Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise, Proc. KDD, 96, 226–231.

[124] Firstov, P.P. & Kravchenko, N.M., 1996. Estimation of the amount of explosive gas released in volcanic eruptions using airwaves, Volcanol. Seismol., 17, 547–560.

[125] Fukao, Y. & Suda, N., 1989. Core modes of the Earth's free oscillations and structure of the inner core, Geophys. Res. Lett., 16, 401–404.

[126] Garces,M., Harris, A., Hetzer, C., Johnson, J., Rowland, S., Marchetti, E. & Okubo, P., 2003. Infrasonic tremor observed at Kilauea Volcano, Hawaii, Geophys. Res. Lett., 30, 2023, doi:10.1029/2003GL018038.

[127] Gresta, S., Ripepe, M., Marchetti, E., D'Amico, S., Coltelli, M., Harris, A.J.L. & Privitera, E., 2004. Seismoacoustic measurements during the July–August 2001 eruption at Mt. Etna volcano, Italy, J. Volc. Geotherm. Res., 137, 219–230.

[128] Hagerty, M.T., Schwartz, S.Y., Garces, M.A. & Protti, M., 2000. Analysis of seismic and acoustic observations at Arenal Volcano, Costa Rica, 1995–1997, J. Volc.Geotherm. Res., 101, 27–65.

[129] Harris, A.J.L., Blake, S., Rothery, D.A.&Stevens, N.F., 1997. A chronology of the 1991 to 1993 Mount Etna eruption using advanced very high resolution radiometer data: implications for real-time thermal volcano monitoring, J. geophys. Res., 102, 7985–8003.

[130] Hastie, T., Tibshirani, R. & Friedman, J., 2002. The Elements of Statistical Learning, p. 533, Springer, New York.

[131] Hori, S., Fukao, Y., Kumazawa,M., FurumotomM. & Yamamoto, A., 1989. A new method of spectral analysis and its application to the Earth's free oscillations: the "Sompi" method, J. geophys. Res., 94(B6), 7535–7553.

[132] Hsu, C.W., Chang C.C. & Lin, C.J., 2007. A practical guide to support vector classification, http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf (last accessed 2011 January 31).

[133] Hsu, C.W. & Lin, C.J., 2002. A comparison of methods for multi-class support vector machines, IEEE Trans. Neural Netw., 13, 415–425.

[134] Hwanjo, Y., Yang, J.&Han, J., 2003. Classifying large data sets using SVMs with hierarchical clusters, in Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC.

[135] Johnson, J.B., 2005. Source location variability and volcanic vent mapping with a small-aperture infrasound array at Stromboli Volcano, Italy, Bull. Volcanol., 67, 1–14.

[136] Matoza, R.S., M.A.H. Hedlin, and M.A. Garces (2007), An Infrasound Array Study of Mount St. Helens, J. Volcanol. Geotherm. Res., 160, 249-262, doi:10.1016/j.jvolgeores.2006.10.006.

[137] Jain, Murty, Flynn (1999). “Data clustering: A survey”. ACM Comput. Surv., 31:264–323.

[138] Fukunaga K., Introduction to statistical pattern recognition (2nd ed), Academic Press, Boston 1990.

[139] Oehler K.L. and Gray R.M. (1995), Combining Image Compression and Classification Using Vector Quantization, IEEE Trans. Patter Analysis and Machine Intelligence, vol. 17, no. 5, pp. 461-473.

[140] Xie Q.B., Laszlo C.A. and Ward R.K., (1993), Vector Quantization Technique for Nonparametric Classifier Design, IEEE Trans. Patter Analysis and Machine Intelligence, vol. 15, pp. 1326-1330.

[141] Jain, A.K., Duin, P.W., and Mao J. (2000), Statistical pattern recognition: a review, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22, pp. 4-37.

[142] Mao, J. And Jain, A. K. (1996), A self-organizing network for hyperellipsoidal clustering (HEC), IEEE Trans. Neural Netw., 7, pp. 16–29.

[143] Mika, S., Ratsch, G., Weston, J., Scholkopf, B. Mullers, K.R. (1999), Fisher discriminant analysis with kernels, IEEE International Workshop on Neural Networks for Signal Processing IX, Madison, USA, August, pp. 41–48.

[144] Dubes, R. C. (1993), Cluster analysis and related issues. In Handbook of Pattern Recognition & Computer Vision, C. H. Chen, L. F. Pau, and P. S. P. Wang, Eds., World Scientific Publishing Co., Inc., River Edge, NJ, pp. 3–32.

[145] Aldridge, M., (2006), Clustering an overview, in "Lecture Notes in Data Mining", Berry M.W., and Browne, M. eds., pp. 99-107, World Scientific.

[146] Zhang, K. (1995), Algorithms for the constrained editing distance between ordered labeled trees and related problems, Pattern Recogn, 28, pp. 463–474.

[147] Wilson, D. R. And Martinez, T. R. (1997), Improved heterogeneous distance functions, J. Artif. Intell. Res., 6, pp. 1-34.

[148] Diday, E., and Simon, J. C. (1976), Clustering analysis, In Digital Pattern Recognition, K. S. Fu, Ed. Springer-Verlag, Secaucus, NJ, pp. 47–94.

[149] Ichino, M., and Yaguchi, H. (1994), Generalized Minkowski metrics for mixed feature-type data analysis, IEEE Trans. Syst. Man Cybern., 24, pp. 698–708.

[150] Knuth, D. (1973), The Art of Computer Programming, Addison-Wesley, Reading, MA.

[151] Fu, K. S. and Lu, S. Y. (1977), A clustering procedure for syntactic patterns, IEEE Trans. Syst. Man Cybern., 7, pp. 734–742.

[152] Kaufman L., and Rousseeuw, P.J. (1990), Finding Groups in Data: An Introduction to Cluster Analysis, Wiley, New York.

[153] Dunham, M. (2003), Data Mining: Introductory and Advanced Topics, Prentice Hall, Upper Saddle River, NJ.

[154] McQueen, J. (1967), Some methods for classification and analysis of multivariate observations, In Proceedings of the Fifth Berkeley Symposium on

Mathematical Statistics and Probability, pp. 281–297.

[155] Modha, D. S., and Spangler W. S. (2002), Feature weighting in k-means clustering, Machine Learning, 50.

[156] Anderberg, M. R. (1973), Cluster Analysis for Applications, Academic Press, Inc., New York, NY.

[157] Wu, K., and Yang, M. (2002), Alternative c-means clustering algorithms, Pattern Recognition, 35, pp. 2267-2278.

[158] Wu, W., Xiong, H., and Shekhar, S. (2004), Clustering and Information Retrieval, Kluwer Academic Publishers, Norwel, MA.

[159] Pena, J., Lozano, J., and Larranaga, P. (1999), An empirical comparison of four initialization methods for the k-means algorithm, Pattern Recognition Letters, 20, pp. 1027-1040.

[160] Likas, A., Vlassis, N., and Verbeek, J. (2003), The global k-means algorithm, Pattern Recognition, 36, pp. 451-461.

[161] R. Baeza-Yates. A unified view to pattern-matching problems. Dept. of Computer Science, Univ. of Chile. ftp://sunsite.dcc.uchile.cl/pub/users/rbaeza/unified.ps.gz, 1995.

[162] Gonzalez, R.C., Thomas, M.G. (1978), Syntactic Pattern Recognition: an Introduction, Addison Wesley, Reading, MA.

[163] Aliotta, M., Cannata, A,, Cassisi, C., Montalto, P., Privitera, E., Pulvirenti, A. (2010). “Unsupervised clustering of infrasonic events at Mount Etna using DBSCAN and SVM”. In: Geophysical Research Abstracts- EGU General Assembly 2010. Vienna, 2-7 May 2010, vol. 12.

[164] Cannata, A., Montalto, P., Aliotta, M., Cassisi, C., Pulvirenti, A., Privitera, E., Patanè, D. (2011). “Clustering and classification of infrasonic events at Mount Etna using pattern recognition techniques”. Geophysical Journal International, 185: 253–264.