UNIVERSITÀ DEGLI STUDI DI SIENA

FACOLTÁ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dottorato di Ricerca in Ingegneria dell'Informazione - XX Ciclo

Ph.D Thesis

FAST CLUSTERING FOR WEB INFORMATION RETRIEVAL

Candidate
FILIPPO GERACI

Advisor:
Prof. MARCO MAGGINI

Co-Advisor:
Dr. MARCO PELLEGRINI

ANNO ACCADEMICO 2007-2008

To my parents
who hid something special in me
and
to Loredana
who was able to find it

Contents

Chapter 1

Introduction

Abstract

Clustering is a widely used technique to partition a set of heterogeneous data to homogeneous and well separated groups. Its main characteristic is that it does not require a-priori knowledge about the nature and the hidden structure of the data domain. Even if this fact makes clustering a hard task, from the other hand it makes clustering s highly flexible data processing tool. In this thesis we investigate clustering techniques and their applications to Web text and video information retrieval. In particular we focus on: web snippets clustering, video summarization and similarity searching. For web snippets, clustering is used to organize the results returned by one or more search engines in response to a user query on the fly. The main difficulties concern: the poor informative strength of snippets, the strict time constraints and the cluster labelling. Video summarization is used to give an idea of the content of a video without downloading it. Also in this case processing time is important, moreover the amount of involved data is much higher. For this task we designed an approximate clustering procedure much faster than the state of the art algorithms and comparable in quality. In document similarity searching, clustering is used off-line in the preprocessing phase. Here the problem of scalability is raised by the huge size of document corpora. A further complication is that users can weight each field of the query. Allowing this feature has negative effects on preprocessing. For this task we designed: a novel clustering strategy to improve results quality and a weighting scheme that does not influence preprocessing.

1.1 Introduction

Since the early 90's the World Wide Web is growing fast. In the last decade, due to the bandwidth constraints, web sites were in essence interconnected texts collections. These text corpora have nowadays become huge. Thus the organization and retrieval of information on the Web has become one of the hottest topic in the information retrieval research community.

In the recent years Internet and, in particular, the Web, has changed. An increasing number of users has now access to broad band connections, thus web sites had the opportunity to enrich their contents with images and also with videos. This situation caused the need to design information retrieval systems able to manage also multimedia data.

The last in the time line but not least important way in which the Web has modified its interface is the so called Web 2.0. Thanks to technologies like AJAX (Asincronous Javascript And XML), a modern web site is no longer a "static" collections of texts, images and videos, but it has become much more similar to a classical interactive stand-alone application. This mutation of the web scenario had also deep implications in the information retrieval community. In fact, to be able to deal with huge data corpora and multimedia data in not enough, algorithms should work on-line. Moreover also the users have changed their requirements. Therefore, the success of a web information retrieval system does not depend only on the quality of the returned results, but also the response time must be considered.

For example, everybody loves *Google* and *Yahoo!* search engines. Their precision is widely appreciated, but also the response time is part of their success. If tomorrow one implements a new search engine much more precise than Google but with an expected response time of two seconds, it is probable that it will used only in those few cases in which Google fails.

As the reader can expect, large corpora coming from the Web are highly heterogeneous. Clustering is a widely used technique to partition a set of heterogeneous objects into small groups of objects related among them. Clustering has many properties that make it largely used and appreciated. Its most important feature is that clustering algorithms do not need any a-priori knowledge of the input data and domain. The only requirement is to provide a notion of distance (or similarity) between pairs of input objects. This property of clustering made it suitable for essentially any data domain. For example clustering is widely used for a preliminary exploration of a large amounts of data. The lack of any information about the data domain is at the same time the of success key and the drawback of clustering. In fact, clustering algorithms tend to produce results of worse quality with respect to, for example, classification algorithms where for each possible category to which data can be assigned to, a set of positive and negative examples must be available. Moreover a not enough well pondered choice of the distance function is likely to produce a complete failure.

1.2 Thesis goals and results

The main goal of this thesis is to propose clustering algorithms able to produce high quality results, but fast enough to be suitable for on-line web applications. We applied our results to three main different contexts: web snippets, video summarization and document similarity searching. These three problems are very different among them and require to work under different constraints. To be competitive in all these tasks, a clustering algorithm must be enough flexible. Moreover these problems are interesting also because they raise a series of related issues we faced in this thesis finding novel solutions.

Snippets clustering has become popular since it was widely used by clustering web search engines like *Vivisimo*. When a user issues a query to the system, the query is redirected to some auxiliary search engines, the returned snippets are clustered and each cluster is labelled with a short sentence to allow users to predict the cluster content. In this particular case, the prediction strength of the labelling algorithm is not less important than the clustering quality. Clustering and labelling are strictly related. In fact, a good cluster with no predictive label is likely to be ignored by the user. Instead clusters that are not well formed are likely to have a poor label. What makes clustering of web snippets a hard problem is the fact that snippets are essentially very small pieces of text and thus the amount of information contained in each of them is poor. This causes a phenomenon in which the distance between two related snippets is no far from the distance between two unrelated ones. Despite the fact that the number of snippets to be clustered is relatively small (typically about 200 items), clustering efficiency still remains an issue because a not negligible part of time is spent querying the auxiliary search engines. Studying this problem we designed the M-FPF algorithm described in chapter 2 which exploit the triangular inequality to speed up the FPF heuristic by Gonzalez [Gonzalez, 1985]. Moreover we improved also the clustering quality providing a novel definition of the concept of *medoid*. Therefore we modified the FPF clustering schema to handle medoids (M-FPF-MD). We also designed a novel labelling algorithm that works following three main steps. Firstly it selects a certain number of representative keywords for each cluster (signatures); then it uses a modified version of the information gain measure to univocally retain keywords that appear more than once only in the most appropriate signature and removes them from the others. Finally, for each cluster, it extracts a short sentence based on its signature. The goal of the first step is to “locally” detect informative keywords for each cluster. The second step has a “global” overview of the clusters and it is necessary to avoid that clusters about different aspects of the same topic have a too similar label. The last step is only used to produce a more charming output for users. With the aim of validating our results, we set up *Armil* a full featured clustering meta search engine.

Video summarization has become an important application since the popularity of video portals in the web like *You tube* began to grow. At the present time an user who wants to decide whether a video is of interest or not, is required to download it entirely and to inspect some random snippets. This situation hinders the video browsing experience of users causing a waste of network resources and time. Video streaming systems can drastically reduce the use of network resources, but the problem to watch completely the video in order to establish if it is of interest still remains. The task of video summarization is completely different from the previous one. First of all because the input objects are video frames (or scenes) instead of texts. Secondly there exist an evident relationship among consecutive frames. Moreover the amount of data involved in the clustering procedure is much higher in consideration of the fact that a typical video has a rate of at least 25 frames per second. This means that a video of just five minutes contains at least 7500 frames. For this particular problem only cluster centers are of interest, the rest is discarded. In fact a summary is obtained concatenating the frame (or scene) associated to each cluster center. The intuition beyond this approach is that, if a cluster is homogeneous, its center should be enough representative of the content the whole cluster. This also means that the error due to the insertion of a point in the wrong cluster has completely no effect on the final results if it does not affect the cluster center. As a direct consequence a faster approximate clustering can be considered as a possible alternative. In this sense we designed some approximation techniques devoted to noticeably speed up the execution of the clustering algorithm attempting to avoid a loss in the result quality. We based our approximated clustering on M-FPF-MD. We observed that the two most time consuming steps in this algorithm are: the update of medoids and the procedure of searching, for each new inserted point, of its closest center. Once the initial medoid is computed, our approximated procedure requires only two distance computations to perform the update. We also approximated the procedure of insertion of a new point into the clusters taking explicitly advantage of the distribution of distances between pairs of consecutive points. We also used this distribution of distances for a procedure that suggests an appropriate size for the storyboard that corresponds to the number of clusters to make. As a final result we designed *ViSto* a completely working Web 2.0 application for static video summary.

Similarity searching is a feature often provided by search engines. Well known examples are: the link “related pages” present in many snippets returned by Google or the “active bibliography” in Citeseer[1]. In these examples the document for which is requested to find the related documents is one of the documents present in the collection, but this requirement is not necessarily a constraint of the problem. The supplied query can be also an external document or a sequence of keywords. The naive solution to similarity searching consists in a linear scan of all the dataset, comparing each document with the query and returning the few closest ones. Even for relatively small document corpora this solution is impractical for on-line computations, therefore efficient algorithms were developed for this purpose. The solution provided by similarity searching algorithms can be either exact or approximate. In the specific case of the retrieval of related pages or articles, an approximate

[1]`http://citeseer.ist.psu.edu`

solution can be preferable especially if it is fast. We concentrate on approximate similarity searching with in mind the goal of providing a high quality solution returned in an a-priori bounded time. We followed the cluster pruning approach, that performs a preprocessing step in which the document corpus is clustered. Then a query consists in the scan of all the documents contained in a certain predefined number of clusters whose centers are the nearest with respect to the query point. Following the cluster pruning approach we observed that the final result quality is strictly related to the distance between the query point and its closest center. This means that, when the query document is also part of the collection (like in the above examples), more compact clusters are likely to produce better quality. In other words, from this observation, we derived that a clustering algorithm for the k-center problem, which minimizes the maximum cluster diameter, is likely to produce high quality results. We also observed that, after just the visit of three or four clusters, the probability to find more related documents examining a new cluster decays drastically. This suggests that it is probably better to examine fewer clusters in some independent clusterings of the collection than more clusters of the same clustering. We tested our approach over two datasets of semi-structured bibliographic records (title, authors, abstract) from *CiteSeer*. In this case, each data object is much bigger and more representative than in the case of web snippets. Moreover, the number of involved data objects is of three order of magnitude bigger than in the case of snippets. Clearly, for the similarity searching problem clustering does not require to be made on-line, but it is part of a preprocessing phase. Despite the fact that the main goal is to attain the highest possible result quality, if one considers the size of the involved datasets and the growth trend of data available in the Internet scalability still remains an important issue. In fact a clustering algorithm able to preprocesses the entire corpus in one day is much better than an algorithm that requires a month. Finally we investigated the more complicated setting of *dynamic vector score aggregation* in which again data objects are semi-structured texts and for each field (title, authors, abstract) there is an independent vector space. Each data object is then represented by the linear combination of its vectors and the user is allowed to assign a weight to each field. Weights are known only at query time. We propose a novel method to embed weights such that it is no longer needed to know/manage them during the preprocessing of data.

1.3 Thesis outline

This thesis is organized as follows:

- **Chapter 2** introduces the problem of clustering in general. In the first part of this chapter we survey the most important clustering approaches, algorithms and distance functions, then we briefly introduce and describe the most used techniques and strategies for the clustering validation problem. In the second part of this chapter we provide details of the FPF heuristic for the k-center problem, our clustering strategies and optimizations and our definition of medoids.

- **Chapter 3** surveys the most common strategies for text clustering. In the first part we introduce the well known vector space model for text retrieval and some variants. Then we discuss how to remedy to some limits of this model due to the intrinsic characteristics of the human language. Then we provide a comparison among distance functions for text and a comparison between our clustering algorithm against the classical k-means. We conclude the chapter with the description of our algorithm for labelling text clusters.

- **Chapter 4** introduces *Armil* a clustering meta search engine for web snippets. Armil is a completely working system whose design was aimed at experimentally testing our findings and at showing that from results of our research it is possible to produce a software prototype not worse in quality and performance with respect to a successful commercial software.

- **Chapter 5** copes with the problem of video summarization. In this chapter we provide a solution to produce both static storyboards and dynamic video abstracts. In the first part of the chapter we discuss our approximation and optimization strategies for the static case. Together with the clustering algorithm, a novel method to suggest a possible appropriate storyboard size is introduced. We compare our results with other state of the art methods. Also in this case we decided to implement a fully featured system to demonstrate the validity of our approach. In the second part we focus on in dynamic video abstracts. Here the main contribution is the definition of the scene's boundaries and our investigation about the use of frames or scenes as input for clustering.

- **Chapter 6** copes with the similarity searching problem. In the first part of the chapter we survey the most common problems related to similarity searching and well known solutions for both the exact and the approximate version of this problem. Then, we derive a relationship between similarity searching and the k-center target function for clustering. Thus we exploit this relationship to improve the cluster pruning approach for approximate similarity searching. Moreover, we introduce a novel method to manage user defined weights in the more complex problem of vector score aggregation. Then we show how to use this scheme to speed up noticeably the clustering phase.

- **Chapter 7** draws some conclusions and summarizes the results described in this thesis. Finally we describe briefly the future directions for our research.

Chapter 2

Clustering

Abstract

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

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

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

2.1 Introduction to clustering

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

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

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

2.1.1 Metric space for clustering

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

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

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

- **Cosine similarity:** it is defined as the cosine of the angle between o_a and o_b. More formally

$$s(o_a, o_b) = \frac{o_a \cdot o_b}{\|o_a\| \cdot \|o_b\|}$$

  A distance can be easily derived from cosine similarity by:

$$d(o_a, o_b) = \sqrt{1 - s^2(o_a, o_b)}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2 Clustering strategy

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

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

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

2.2.1 Partitional clustering

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

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

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

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

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

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

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

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

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

2.2.1.1 FPF algorithm for the k-center problem

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

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

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

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

Basic algorithm

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

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

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

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

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

2.2.1.2 K-means

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

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

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

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

is minimized.

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

The main building blocks of k-means are:

- **the generation of the initial k candidate centroids**: In this phase an initial choice of candidate centroids must be done. This choice in critic because both the final clustering quality and the number of iterations needed to converge are strongly related to this choice. In the next section we will survey the most important initialization strategies. A more complete survey and comparison can be found in [Bradley and Fayyad, 1998; Peña *et al.*, 1999].

- **the main iteration loop**: In the main iteration loop, given a set of k centroids, each input point is associated to its closest centroid, and the collection of points associated to a centroid is considered as a cluster. For each cluster, a new centroid that is a (weighted) linear combination of the points belonging to the cluster is recomputed, and a new iteration starts[1].

- **the termination condition**: Several termination conditions are possible; e.g. the loop can be terminated after a predetermined number of iterations, or when the variation that the centroids have undergone in the last iteration is below a predetermined threshold.

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

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

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

Initialize k-means

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

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

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

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

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

2.2.1.3 PAM: partition around medoids

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

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

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

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

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

2.2.1.4 SOM: self organizing maps

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

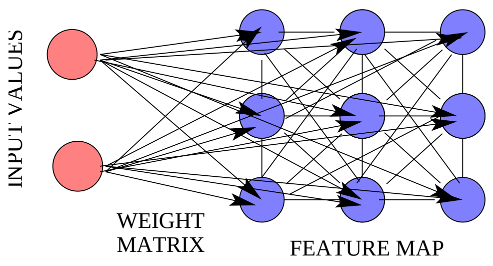


Figure 2.1. A simple 3×3 self organizing map.

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

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

Self-organizing maps work in two phases:

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

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

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

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

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

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

Algorithm 2: The self-organizing map algorithm.

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

2.2.2 Hierarchical clustering

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

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

The two main strategies for hierarchical clustering are:

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

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

2.2.2.1 Divisive clustering

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

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

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

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

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

2.2.2.2 Agglomerative clustering

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

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

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

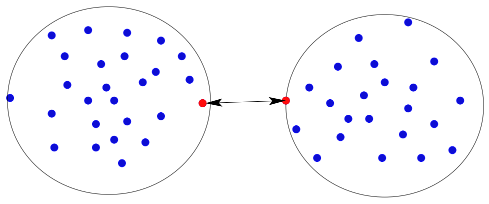


Figure 2.2. Single linkage criterion.

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

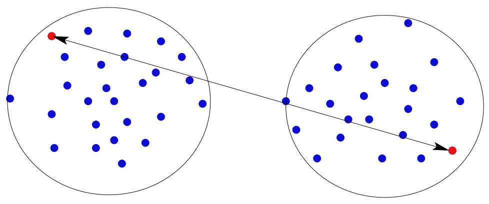


Figure 2.3. Complete linkage criterion.

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

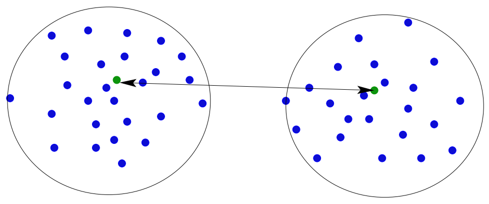


Figure 2.4. Average linkage criterion.

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

$$\begin{aligned} d(C_l, (C_i, C_j)) = \alpha_i d(C_l, C_i) + \alpha_j d(C_l, C_j) + \beta d(C_i, C_j) + \\ + \gamma \mid d(C_l, C_i) - d(C_l, C_j) \mid \end{aligned} \tag{2.1}$$

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

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

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

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

that is the formula of complete linkage.

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

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

2.2.3 The choice of the number k of clusters

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

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

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

2.2.3.1 Stability based techniques

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

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

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

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

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

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

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

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

We then cluster the input set O_{ds}. Suppose at step t we have computed the clusters $C_{t,1}(ds), \dots, C_{t,t}(ds)$ and suppose, for each $o \in O_{ds}$, we keep the index $i(o,t)$ of its closest center among $c_1, \dots, c_t$. Such index can be updated in constant time by comparing $d(o, c_{i(o,t-1)})$ with $d(o, c_t)$, i.e., the distance of o from the "current" center and that to the new center c_t. Now, for each $C_{t,l}(ds)$, $l \in [1, \dots, t]$ we can easily count in time $O(|C_{t,l}(ds)|)$ the number of elements that are closest to the same center c_j, $j \in [1, \dots, t]$, and finally compute the summations in formula 2.2 in time $O(|O_{ds}|)$.

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

2.3 Clustering validation

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

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

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

2.3.1 Internal measures

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

2.3.1.1 Homogeneity and separation

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.1.2 Average silhouette

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

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

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

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

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

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

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

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

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

1. A singleton $\{o_i\}$ has silhouette equal to one because $a_i = 0$ and $b_i > 0$ (each element fits well in a cluster by its own).
2. If there is only one big cluster then for each $o_i \in n$ we have $sil(o_i) = -1$, because $b_i = 0$ and $a_i > 0$ (no element fits well in a cluster with all other elements).

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

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

2.3.2 External measures

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

2.3.2.1 F-measure

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

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

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

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

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

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

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

2.3.2.2 Entropy

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

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

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

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

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

2.3.2.3 Accuracy

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

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

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

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

2.3.2.4 Normalized mutual information

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

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

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

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

2.3.2.5 Normalized complementary entropy

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

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

The normalized complementary entropy of c_j is

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

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

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

2.4 Improving the FPF algorithm for the k-center problem

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

2.4.1 Exploiting the triangular inequality to improve the FPF speed

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

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

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

2.4.2 Using a random sample

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

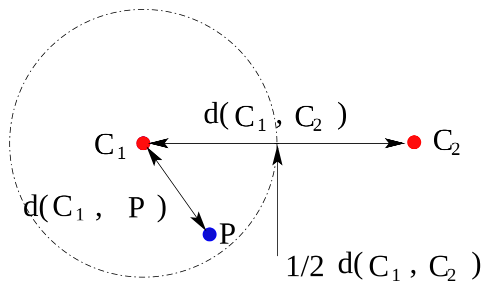


Figure 2.5. Exploiting the triangular inequality.

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

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

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

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

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

Algorithm 3: M-FPF.

2.4.3 Using medoids as centers

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

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

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

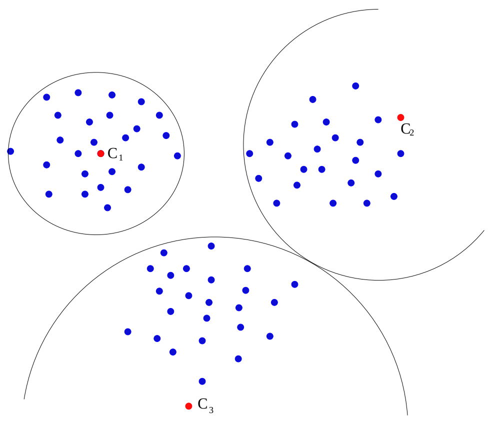


Figure 2.6. An example of clustering made using FPF.

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

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

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

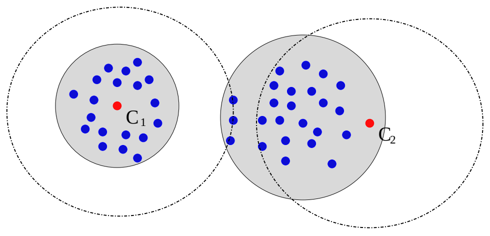


Figure 2.7. An example with two clusters made by FPF, in gray the ground truth.

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

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

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

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

is the medoid of o.

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

The medoid formulae can be generalized via weighting the two components

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

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

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

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

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

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

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

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

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

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

Algorithm 4: M-FPF-MD.

Chapter 3

Text document clustering

Abstract

Dealing with text documents is one of the foremost issues in information retrieval. In this context, clustering plays a strategic role. Large text document corpora have become popular with the growth of the Internet and the decrease of price of disk storage space and connection band-width.

Dealing with text documents is a hard task. This is due to some intrinsic characteristics of human languages. For example, the same word can have different meanings according with the context in which it is referred. Moreover the prefix or suffix of a word can vary in different contexts. All the peculiarities of human languages motivate the effort of researchers in the field of text information retrieval.

In this chapter we survey the most important problems and techniques related to text information retrieval: document pre-processing and filtering, word sense disambiguation, vector space modeling, term weighing and distance functions. In the second part of the chapter we present two text clustering problems (on-line clustering of small semi-structured text corpora and off-line clustering of a large corpus) and report a comparison of our clustering algorithms against k-means, which is the most used algorithm in the text clustering context. Later in the chapter we introduce the problem of cluster labelling: in a nutshell once a certain corpus was clustered into groups of homogeneous documents, in some cases, one would want to synthesize a short label to deduce the cluster topic. In the final part of the chapter we show our solution to this problem.

3.1 Introduction

The term *Information retrieval* (IR) was firstly introduced by Calvin Mooers at the end of 40's. Then information retrieval has become a very broad field of computer science. It can be intuitively defined as the task of finding interesting and typically unstructured "objects" for the user information needs in a wide collection.

Conceptually, IR systems can be designed to manage almost everything (texts, images, videos), but the data type which has concentrated most of the studies is text. The growth of the Internet has stressed even more the interest in this sense. A lot of efforts have been spent to design general models for text information retrieval. Of course, the question about how similar are two documents has still not found a univocally accepted answer. There are many reasons that make it difficult deal with texts. First of all, the same concept can be expressed in many different ways by different writers. Moreover, the use of synonyms can drastically reduce the number of words shared by two related documents. On the contrary, the same word can assume very different meanings according with the context in which it is used. Nevertheless texts have the intrinsic characteristic that a not negligible part of words are due to grammar rules and do not provide additional information.

Despite text documents do not have a clear structure in general, this can not be considered always true. In fact, text documents are usually divided in sections, begin with a title and, in many standard document file formats, their structure is precisely marked using tags (HTML, XML and RTF just to give some examples). Another example is constituted by semi-structured texts that are documents with a poor structure (i.e. web snippets). Semi-structured texts will be the most used documents in the applications described in this thesis. In chapter 4 we will deal with snippets coming from web search engines with the goal of arranging them in an automatically generated hierarchy. In chapter 6 we will test approximate similarity searching techniques dealing with bibliographic records from Citeseer[1].

In this chapter we survey the major techniques designed to manage text documents and some considerations for applying clustering algorithms to these data.

3.2 Text representation

A text document is, in its most simplistic representation, a sequence of words. With the purpose of indexing it or computing its similarity with other documents (or equivalently with a text query), it must be preprocessed to remove the noise due to "syntactic sugar" and make it more treatable by computers. Preprocessing typically consists of many steps:

- **Text normalization**: in a document the same word can appear in different forms. For example the beginning of a sentence begins with capital letter. Naturally, these little variations do not affect the semantics of the term. To

[1]http://citeseer.ist.psu.edu/

make it easier for IR systems to manege texts, terms must be normalized by converting them to lower case, remove dashes in multi-words terms, etc. Numbers, dates and punctuation are removed from texts. In the *bag of words* model, where the context of the words is not used, terms are often sorted lexicographically.

- **Stemming**: one of the major differences between human languages and programming languages is that, in the first case, words have a fixed root and a suffix (often also a prefix) that can vary depending on the context, while, in the last case, keywords are invariable. However, the large part of the words semantics is contained in their root, thus terms with the same root should be considered as the same term. The goal of stemming algorithms is to reduce a word to its root. To complicate this task there is the fact that the rules for extracting the root of a word depend from two aspects: the type of word (i. e. verbs, conjugations) and the language (i. e. English, Italian). Moreover human languages admit a lot of exceptions (i. e. the plural form of the word child is children instead of childs as suggested by the standard rule). Martin Porter [Porter, 1980] in 1979 introduced a rule based algorithm that has become the most famous stemmer algorithm for English still in use. Similar word stemmer algorithms are now available in almost all languages. The major perplexity in using stemming is that it can cause misunderstanding and change the meaning of the words. For example a too aggressive stemming can reduce the word "organization" to "organ" or "policy" to "police". On the other hand, also a mild stemming can modify the original sense of a word.

- **Stop words removal**: to the contrary of computer languages, human languages are rich of words. Most of them have not a meaning by themselves but are used as grammar bricks to build complex sentences (for example articles or prepositions). All these words can be safely removed from the text without any loss of information. There are also words which have a very general meaning or are too popular to be really helpful in the understanding of the semantics of a text or its topic (i. e. above or below). Their removal cause only a marginal loss of information, but has the benefit of reducing the size of the representation of the text (in sections 3.2.1 we will explain the reasons that make this reduction a desirable property). The set of all the terms, removable with a negligible semantics loss, is called stop words list.

- **Vocabulary building**: the growth of the computational power, memory size and bandwidth was followed by an increase of the available textual information. Low hardware costs had the effect that, even small enterprises want to be able to process their internal knowledge base. Vocabularies do not change the models for storing and querying texts, they are only used to produce a more compact representation of texts with the goal of maintain much more documents in main memory. To each distinct term of the corpus an univocal identifier is assigned and it is stored in a table V called vocabulary. Thus, each document can be represented in a more compact form as a list of term identifiers with a consequent reduction of the required storage.

Before to discuss a model for dealing with texts, it is important to understand how words are distributed in documents and what are the implications of this distribution. Words in a corpus are not evenly distributed. A quite small set of words appear very frequently. Some of them like articles and prepositions are connected to grammar and can be removed as stop words, some others are instead topic dependent. A medium size set of words appear with intermediate frequency and a broad set of words appear rarely. This last set is made wider for example by words containing typos or by very specific words. This distribution, known as *power law*, was observed in all the human languages and is widely accepted as an intrinsic human characteristic. Clearly, words that appear with high frequency are useless because they do not exploit differences among documents (their presence in two documents does not means that those document are similar). Also rare words can be useless (the reason will be more clear in section 3.2.1).

3.2.1 The vector space model

With the term *model* in information retrieval we refer to a representation for documents and queries equipped with a notion of distance/similarity among them. The *vector space model* is a well known model, widely accepted and used, for organizing texts. After preprocessing, described in the previous section, a text document is reduced to a flat list of terms. Moreover, after preprocessing a vocabulary of all the terms in the document becomes available. Thus a document can be stored in a vector that has as many components as vocabulary words. Each component of the vector represents a score for the corresponding word (depending from the chosen weighting schema) or it is 0 if the word is not present in the document. All the documents of the corpus can be arranged in a matrix called *document matrix* such that rows correspond to documents and columns refer to terms.

Let $D = \{d_1, \dots, d_n\}$ be a corpus of n documents such that V is the vocabulary of all the words in D. Thus D can be arranged in a matrix M such that $m_{i,j}$ corresponds to the term $v_i \in V$ in document $d_j \in D$. There are many possible different weighting schemes proposed in the literature. The most advanced IR systems weight terms according to their importance and characteristics (i.e. frequency in the document and in the corpus). The most used weighting schemas are:

- **Boolean (binary) model**: if v_i is present in d_j, then $m_{i,j} = 1$ otherwise $m_{i,j} = 0$.
- **Term frequency (TF)**: let $tf_{i,j}$ be the number of occurrences of term v_i in document d_j. In the term frequency model we have $m_{i,j} = tf_{i,j}$. It could be more convenient to normalize the weights to be independent from the

document length. Let $WC_j = \sum_{i=1}^{|V|} tf_{i,j}$ be the total number of words in d_j, then:

$$m_{i,j} = \frac{tf_{i,j}}{WC_j}$$

According to this normalization, $m_{i,j}$ is in the range $[0, 1]$ and can be interpreted as the probability that word v_i appears in document d_j.

- **Term frequency - Inverse document frequency (TF-IDF)**: let w_i the number of documents in which v_i appears. We define $idf_i = \log n/w_i$. According to the TF-IDF schema $m_{i,j} = idf_i * tf_{i,j}$. To normalize the TF-IDF score in the range $[0, 1]$ the following formula is often preferred:

$$m_{i,j} = \frac{tf_{i,j} * idf_i}{\sqrt{\sum_{k=1}^{|V|} (tf_{k,j}^2 * idf_i^2}}.$$

This scheme assigns high score to those words that appear frequently in a document, but are rare in the corpus. Instead, words that appear in a large portion of the document corpus are not too helpful to exploit differences among documents and thus are considered not important.

The vector space model has two main drawbacks. Since documents are vectors with $|V|$ components, even in the case of small corpora the dimensionality of the resulting vector space is usually high. Moreover, documents are very sparse vectors. A side effect of these two phenomena is that distances among documents tend to become high. In addition, the distance between a pair of similar documents is not so far from the distance between two unrelated ones. The standard technique to reduce vector space dimensionality and make document vectors more dense is the feature selection that will be discussed later in this section.

Another important issue is that the bag of words model does not considerate the context of words. Context is clearly important to extract the sense of a term because the same word could change its meaning in different contexts. This fact became more evident for multi word terms. For example phrasal verbs in English or people names. The problem of assigning the correct meaning to a word, called *word sense disambiguation*, is well studied and many techniques have been proposed in the literature, but they are typically more complex or computationally expensive or their performance depends from a knowledge base and thus are topic dependent. Moreover, none of them has at the moment exploited sufficiently higher performances with respect to the bag of words model to be considered as a valuable alternative.

3.2.1.1 Metric spaces and distance

A natural way to see documents (and queries) in the previously described model is thinking to them as vectors (or points) in a high dimensional Euclidean space.

When a normalized weighting schema is applied, documents lay in the positive part of the surface of a iper-sphere of radius 1.

The most natural way to compute distances among documents is using the classical Euclidean distance. This distance is a special case of the *Minkowski distance* in which the parameter p is set to 2. Given two documents d_1 and d_2 and the related rows of the matrix M, The Minkowski distance is defined as

$$L_p(d_1, d_2) = (\sum_{i=1}^{|V|} |m_{i,1} - m_{i,2}|^p)^{1/p}$$

.

Given two vectors the measure of the angle formed by them can be used as distance between the two represented documents. If the two documents contain exactly the same words, they give raise to the same vector and thus their angle is 0. In case the two documents do not have words in common they produce two orthogonal vectors and thus their distance is maximum. Based on this idea the *cosine similarity* is defined as the cosine of the angle formed by two vector documents. More formally let d_1, d_2 be two documents:

$$s(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| \cdot \|d_2\|}$$

Note that denominator is used only to normalize the cosine similarity to be in the range $[0, 1]$ independently from the length of the involved vectors. This measure is a similarity score. Similarity is the dual concept with respect to distance. The more two objects are similar (similarity value is high) the more their distance tends to 0 and vice versa. Cosine similarity and all algorithms designed to employ similarity measures can be converted to use distances and vice versa. As noted in [Clarkson, 2006] the inner product of two vectors d_1 and d_2 of length 1 (in norm 2), that is the standard cosine similarity of two normalized vectors, is turned into a distance by $D(d_1, d_2) = 1 - s(d_1, d_2)$. This distance function is not a metric in a strict sense since the triangular inequality does not hold, however the following derivation $\|d_1 - d_2\|_2^2 = d_1 \cdot d_1 + d_2 \cdot d_2 - 2d_1 \cdot d_2 = 2(1 - d_1 \cdot d_2) = 2D(d_1, d_2)$ shows that the square root of the distance is indeed a metric. Equivalently one can say that it satisfies the extended triangular inequality $D(d_1, d_2)^\alpha + D(d_2, d_3)^\alpha \geq D(d_1, d_3)^\alpha$ with parameter $\alpha = 1/2$. Moreover a linear combination of distance functions with positive weights defined on the same space is still a metric space $D(d_1, d_2) = \sum_i w_i D_i(d_1, d_2)$ for $w_i \geq 0$. Thus cosine similarity although not giving rise to a metric in a strict sense is nonetheless closely related to a metric space.

Another commonly used coefficient to measure distance between pairs of documents is the *Jaccard coefficient*. In its original form this measure does not take into account weights and reduces a weighted scheme to a binary one. Let $d_1 \cap d_2$ be the set of terms that d_1 and d_2 have in common and $d_1 \cup d_2$ the set of terms present in at least one of the two documents. The Jaccard coefficient is defined as:

$$J(d_1, d_2) = \frac{\#(d_1 \cap d_2)}{\#(d_1 \cup d_2)}$$

Many variants of Jaccard coefficient were proposed in the literature. The most interesting is the *Generalized Jaccard Coefficient* that takes into account also the weight of each term. It is defined as

$$GJC(d_1, d_2) = \frac{\sum_{i=1}^{|V|} \min(m_{i,1}, m_{i,2})}{\sum_{i=1}^{|V|} \max(m_{i,1}, m_{i,2})}$$

GJC is proven to be a metric [Charikar, 2002].

3.2.1.2 Word sense disambiguation

All these distance measures have the drawback that, in different ways, the more documents share terms, the more they are considered related. This is not always true for many reasons. Firstly, the same word can have different meanings in different contexts (*lexical ambiguity*), thus having a word in common does not necessarily imply similarity. Secondly, all human languages allow the use of synonyms to express the same concept with different words, therefore two documents can deal with the same topic sharing only few words. Moreover similar concepts can involve the use of complex semantic relationships among the words. For example, after removing stop words, the two sentences: “the apple is on the table” and “there is an orange on my desktop” have no words in common, but both say something about a “fruit on a board”, thus they are not completely unrelated. The above example shows two important notions of similarity:

- **paradigmatic, or substitutional, similarity** when two words may be mutually replaced in a particular context without change the semantics of the text (i. e. the words table and desktop in the previous example),
- **syntagmatic similarity** when two words significantly co-occur in the same context. (i. e apple, orange and fruit in the previous example).

To take into account these similarities among words many techniques have been proposed. The most common approaches are based on the attempt to generate (manually or automatically) an ontology of words. The advantage of ontologies is that they can be used to define a degree of similarity between couples of words, and thus to find relationships among them. In the previous example both the words “orange” and “apple” have as ancestor the term “fruit” and thus they are related. A lot of effort was done to design indexes to measure the degree of similarity between two words in the ontology graph. Many of them take into account the length of the path between two words. In [Agirre and Rigau, 1996] Agirre and Rigau propose the *conceptual density* that also takes into account the depth of the nodes in the hierarchy (deeper are closer) and the density of nodes in the subhierarchies involved (denser subhierarchies are closer)

The most important project for ontologies of words is *WordNet* [Miller, 1990]. Originally proposed by the Cognitive Science Laboratory at Princeton University only for the English language, WordNet has become a reference for all the information retrieval community and similar projects are now available in many other languages. WordNet is a handmade semantic lexicon that groups words into sets of synonyms called *synsets*. Intuitively one can replace a word in a text with another from the same synset without changing its semantics. A word can appear in more than one synset if it has more than one meaning. Moreover synsets are arranged as nodes in a graph such that there is an edge to connect two nodes if there is a relation between the two synsets. There are different types of possible relations, an exhaustive list of them can be found in the WordNet web site [Miller *et al.*, 2006]. Given two synsets X and Y, the most common types of relations in WordNet are: *hypernym* if every X is a "kind of" Y, *hyponym* if Y is a "kind of" X, *holonym* if X is a part of Y and *meronym* if Y is a part of X. Thus, in our example WordNet has a link between orange and fruit and also between apple and fruit hence it is possible to infer a relation between orange and apple.

Clustering is often used also for grouping words into semantically homogeneous sets. This technique is known as *word clustering* [Dhillon *et al.*, 2002; Li and Abe, 1998]. In this case the set of objects to be clustered are not documents like in the previous case, but only words. Thus, the main issues in this context are the features associated to each word and the definition of the distance among words. In fact the distance should be designed in a manner to take into account all the considerations we made before. There are two main philosophies available in the literature. One is to define a distance over an ontology like [Agirre and Rigau, 1996] and thus the issue of how to create the ontology still remains open. Another opportunity is to use a *distributionally-based semantic similarity* approach. In this last case the key idea is that the semantic content of a word can be predicted studying how the word occurs with other words in a corpus. Two words are considered semantically related if they co-occur in a certain number of documents. In [Brown *et al.*, 1991], a vector containing all the immediately succeeding words in the document, is assigned to each term. For each of these words it is reported the number of times they occur after the considered term in the whole corpus. Then a notion of distance between two terms is defined as the average mutual information among all the pairs of words in the context of the two terms.

The other problem we addressed in this section is that the same word can drastically change its meaning in different contexts, thus it should be disambiguated to avoid misapprehensions. There are two main approaches to solve this problem. In theory, one should attempt exploit a sort of *world knowledge* that makes it possible to determine in which sense a word is used. Moreover, this method must be endowed with an inference mechanism that would make use of the base of knowledge to infer the intended sense of words. Clearly, this approach is not suitable in practice because a computer readable general purpose knowledge base for this task does not exist. Some effort was spent to design knowledge-bases and inference systems, but they are usually very limited since they are essentially handmade. What disambiguation systems do in practice is not to try to understand the meaning of text at all, but simply trying to guess the correct meaning of a word looking at the context in which it appears. The key idea of this approach is that, after observing several contexts in which a word is used, it can be possible to disambiguate a word only considering its own context. Many different methods were proposed in the literature; a good survey of the most important ones and related problems can be found in [Ide and Veronis, 1998], [Sanderson, 2000], [Stokoe *et al.*, 2003] and [Linden, 2005]. Here we limit to mention two of the most important results.

Despite the fact that many important ideas and algorithms for word sense disambiguation have been proposed in the literature since the 50's, the first working disambiguator was written by Lesk [Lesk, 1986] in 1986. It was the first software that, for its characteristics, could be used for large text corpora. It was based on the use of an on-line dictionary. In order to disambiguate a term in a certain context, all its possible definitions were looked up in the dictionary. Then all the definitions were treated as a document corpus. The context of the term to disambiguate was, instead used as a query. Thus the problem of word sense disambiguation reduced to a ranking problem (or alternatively to a similarity searching problem). Dictionary clues are too small pieces of text and this negatively affects the disambiguator precision. Since large text corpora became available to researcher, they were employed to overcome this problem.

In [Gale *et al.*, 2004], a hybrid approach that uses both a dictionary and a document corpus is proposed. The only requirement is that each document in the corpus must be available in at least two different languages. To this purpose they use for example the Canadian Hansards which are available in English and French. The dictionary is used to translate a tern from a language to the other. Note that an ambiguous word has typically at least one different translation for each meaning. For example the word *duty* is often translated as *droit* when used with the sense of tax and as *devoir* when mean obligation. In this way it is possible to automatically extract a certain number of instances for each meaning of the word. Moreover, all the meanings of a word can be ranked by collecting statistics of their frequencies in the corpus. These data are then arranged in a training set and a test set. Thus, statistical models can be used for word sense disambiguation.

3.2.1.3 Feature selection

Documents in the vector space model are represented by very large and sparse vectors. Using the standard TF-IDF weighting scheme one can see them as points in the surface of the positive region of a iper-sphere. As explained before, the high dimensionality of documents in this representation has the side effect that the distances among documents become high and distances between pairs of similar documents tend to be close to the distance between pairs of unrelated ones. This is due to the summation of the contributes in the distance computation given by uninformative words. The goal of feature selection is to remove (or at least drastically reduce) the dimensionality of the vectors by removing the not informative words. There are two main strategies for feature selection: one dependent from the content of the corpus, the other independent. Stemming and stop word removal go in the latter direction. Despite the fact that these strategies achieve a good filtering, they are unable to remove those words that are uninformative in a particular context. For this reason a lot of effort was spent to design algorithms that are able to take into account also the corpus content. Some of these methods like Latent Semantic Indexing (LSI) [Deerwester *et al.*, 1990] and Random Projection [Bingham and Mannila, 2001] employ structural properties of the document corpus. Other methods use information theoretic indexes to measure the informativeness of a word and filter those terms out of a certain range.

information theoretic indexes for feature selection Different indexes were proposed in the literature to measure the informative strength of a word, the great majority of them are suitable only for classification. The key idea in this case is to measure the degree of informativeness of the word for each class and then discard those terms with a poor informative power for all classes. Few indexes are suitable for unsupervised learning, in essence they are based on the document frequency. The most common of these measures used for feature selection are:

- **Document Frequency (DF)**: is the number of distinct documents in which a certain word appears in. This number is often normalized to be in the range $[0, 1]$ by dividing it by the total number of documents in the corpus. According to section 3.2, words that appear with high frequency are useless because they do not exploit differences among documents, rare words are also useless because their contribution in the distance computations is negligible. Thus words with document frequency above or below certain thresholds can be discarded.

- **Term Strength**: was originally proposed in [Wilbur and Sirotkin, 1992] for vocabulary reduction in text retrieval. As the document frequency, this index is not task-dependent, thus can be applied also to clustering. Moreover some variants were proposed in the literature specific for the text categorization problem [Yang and Wilbur, 1996]. We describe here its original definition. Term strength collects statistics about the presence of a word in pairs of related documents in a training set, then it uses these statistics to assign a score to the word. A pair of documents is considered to be related if their distance (usually the cosine distance) is under a certain threshold. Let d_1 and d_2 be two related documents and w a word, term strength is defined as:

  $$TS(w, d_1, d_2) = P(w \in d_1 | w \in d_2)$$

  In other words, given two related documents, term strength is the conditional probability that a word occurs in a document given that it occurs in the other. Let (d_i, d_j) be a pair of related documents in the training set, the term strength is:

$$TS(w) = \frac{\#(d_i, d_j) : w \in d_i \wedge w \in d_j}{\#(d_i, d_j) : w \in d_i \vee w \in d_j}$$

- **Gain**: let n the number of documents in the corpus and df_w the number of documents in which the word w appears in, the *Gain* [Papineni, 2001] function is defined as:

$$Gain(w) = \frac{df_w}{n} * \left(\frac{df_w}{n} - 1 - \log \frac{df_w}{n} \right)$$

In this case the gain function assigns a low score both to rare and to common words. On the contrary of DF which requires a range of admissible values, in this case all the words with gain under a certain threshold are discarded. An important difference between Gain and DF is that in the former case the connection between rare and common filtered words is explicit. Figure 3.1 shows the Gain function.

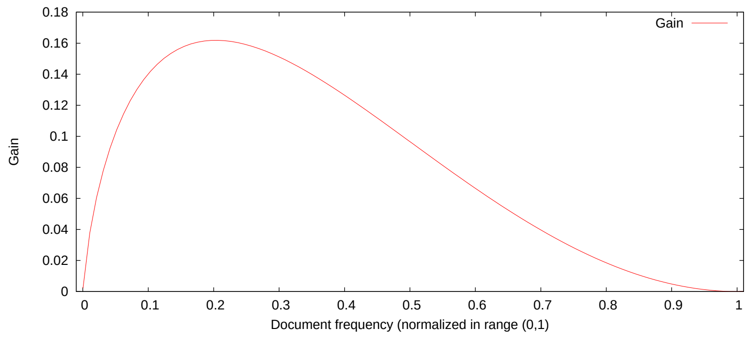


Figure 3.1. The gain function.

- **Information Gain**: is only suitable in the context of classification. The information gain function [Cover and Thomas, 1991; Yang and Pedersen, 1997] measures the contribution in terms of informativeness that the presence or absence of a word gives to a certain class. Let d be a document taken uniformly at random in the set of documents D. $P(v_i)$ is the probability that d contains term v_i, $P(c_j)$ is the probability that d is in category c_j. The complementary events are denoted $P(\bar{v}_i) = 1 - P(v_i)$ and $P(\bar{c}_j) = 1 - P(c_j)$. $P(v_i, c_j)$ is the probability that d is in category c_j and contains term v_i, $P(\bar{v}_i, \bar{c}_j)$ is the probability d does not contain v_i and is not in category c_j. $P(v_i, \bar{c}_j)$ is the probability that d contains v_i but is not in category c_j $P(\bar{v}_i, c_j)$ is the probability that d does not contain v_i and is in category c_j. Clearly being these mutually disjoint events it holds: $P(v_i, c_j) + P(\bar{v}_i, \bar{c}_j) + P(v_i, \bar{c}_j) + P(\bar{v}_i, c_j) = 1$. The information gain is the contribution of the four terms:

$$IG(v_i, c_j) = \sum_{v \in \{v_i, \bar{v_i}\}} \sum_{c \in \{c_j, \bar{c_j}\}} P(v, c) \log \frac{P(v, c)}{P(v)P(c)}$$

Note that information gain assigns a score to v_i only for a category. In order to have a global score for the term v_i different choices are possible. The most common is:

$$IG_{max}(v_i, c_j) = \max_{c_j \in C} IG(v_i, c_j)$$

In this case the key idea is that, if v_i is highly informative for at least one class, its presence helps to classify documents of that class.

- **Gain Ratio**: attempts to overcome some drawbacks of information gain. In fact the value of the information gain formula does not only depends on w_i and c_j, but also from the entropy of the class. Thus normalizing this factor we obtain the gain ratio formula:

$$GR(v_i, c_j) = \frac{IG(v_i, c_j)}{-\sum_{c \in \{c_j, \bar{c_j}\}} P(c) \log P(c)}$$

- **Mutual Information**: is a measure of the degree of dependence between a document d and a class c. Like in the case of information gain, this index is only suitable in the context of classification. More formally mutual information is defined as:

$$MI(w, c) = \log P(w|c) - \log P(w)$$

where w is a word and c is a class. The main drawback of mutual information is that it is highly influenced by the term $P(w)$. Thus for an equal value of the conditional probability rare terms are highlighted. Similarly to information gain a global score for a term w can be computed by one of the following formulas:

$$MI_{avrg}(w, c) = \sum_{i=1}^{m} P(c_i) MI(w, c_i)$$

$$MI_{max}(w, c) = \max_{i=1}^{m} MI(w, c_i)$$

A comparison of many of the above described measures can be find in [Yang and Pedersen, 1997].

Random projection One of the most simple and effective method to reduce the dimensionality of the vector space is the *random projection* [Kaski, 1998; Lin and Gunopulos, 2003]. The idea behind random projection is to reduce the dimensionality of the document matrix by multiplying it for a random projection matrix. More precisely: let M be the document matrix with n documents and m features. Suppose we want to reduce the vector space to be k-dimensional, with $k < m$. Let R a matrix formed by m random k-dimensional vectors. We can project the original document vectors onto a lower dimensional space by:

$$A_{[k \times n]} = R_{[k \times m]} \cdot M_{[m \times n]}$$

Random projection is motivated by the Johnson-Lindenstrauss lemma [W.Johnson and j. Lindenstrauss, 1984]:

Theorem 1. *Let n an integer and $0 < \epsilon < 1$ and k such that*

$$k \geq 4 \left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3} \right)^{-k} \ln n$$

Then for any set M of n points in $\mathbb{R}^m$ there exist a map $f : \mathbb{R}^m \to \mathbb{R}^k$ such that

$$\forall u, w \in M \qquad (1-\epsilon)\|u - w\| \leq \|f(u) - f(w)\| \leq (1+\epsilon)\|u - w\|$$

In simple words, according with the above lemma a set of points in a high-dimensional Euclidean space can be mapped in a lower-dimensional space such that distances between pairs of points are approximately preserved.

One of the mayor issues in the random projection method is the choice of the vectors of R. In theory, if the random vectors are orthogonal the distances between the original points are exactly preserved, thus an orthogonal matrix is desired. In practice, orthogonalization is a very costly operation, thus a reasonable approximation is used. In the literature many methods were proposed to initialize the elements of R, in the most common case they are Gaussian distributed. In [Achlioptas, 2003] two simple possible alternative initializations were proposed to reduce the computational time needed for the calculation for $R \times M$:

- $r_{i,j} = 1$ with probability $1/2$ otherwise $r_{i,j} = -1$
- $r_{i,j} = \sqrt{3} \cdot \begin{cases} -1 & \text{with prob. } 1/6 \\ 0 & \text{with prob. } 2/3 \\ 1 & \text{with prob. } 1/6 \end{cases}$

Latent semantic indexing The main idea behind *latent semantic indexing* (LSI) is to project documents into a low-dimensional space with “latent” semantic dimensions. Even in the case in which two documents do not share terms in the original vector space they can still have a high similarity score in the target space as long as they share “semantically” similar words. Latent semantic indexing is based on the *Singular Value Decomposition* (SVD) applied to the document matrix M.

Latent semantic indexing takes the document matrix M and represents it as a matrix $\hat{M}$ in a k dimensional space ($k << m$) such that it minimizes the following:

$$\Delta = \|M - \hat{M}\|_2 \tag{3.1}$$

Let M be the document matrix with n documents and m features. Using SVD, latent semantic indexing decomposes M into three matrices such that:

$$M_{[m\times n]} = T_{[m\times r]}\Sigma_{[r\times r]}(D_{[n\times r]})^T$$

where T and D are orthogonal matrices that contain respectively the left and the right singular vectors of M and represent the terms and documents in the target space. Σ is a diagonal matrix that contains the singular the values of M and r is the rank of M. If values on Σ are sorted in decreasing order, SVD can be seen as a method to rotate the axes of the target space such that to the i-th axis is associated to the direction with the i-th largest variation. Thus singular values in Σ can be used to rank the "importance" of each dimension in the target space. As a direct consequence latent semantic indexing attempts to reduce the dimensionality in a way such that the dimensions of the target space correspond to the axes of greatest variation.

By restricting the matrices T, Σ and D to their first $k < r$ columns we obtain:

$$\hat{M}_{[m\times k]} = T_{[m\times k]}\Sigma_{[k\times k]}(D_{[n\times k]})^T$$

which is the best approximation for equation 3.1. At this point, to move from the m-dimensional space of words to the k-dimensional space of concepts, documents can be represented as the rows of the following matrix:

$$Z_{[k\times n]} = \Sigma_{[k\times k]}(D_{[n\times k]})^T$$

Despite the high computational cost, latent semantic indexing is one of the most powerful techniques for dimensionality reduction. The choice of the value of k is arbitrary and it is still one of the mayor issues for LSI. A too aggressive dimensionality reduction can negatively affect the quality of results while a too mild reduction can leave noise in the vector space. Typical choices for k are in the range of 100 - 150 features.

3.3 Text clustering

In section 2.2.1.1 and 2.4 we described our family of clustering algorithms based on the Furthest Point First heuristic by Gonzalez [Gonzalez, 1985]. In this section we compare all these algorithms (FPF, M-FPF and M-FPF-MD) in the setting of text clustering. We compared them using two different metric spaces: the well known Cosine Similarity (CS) and the Generalized Jaccard Coefficient (GJC). Moreover we are interested also in comparing our family of algorithms against k-means (described in section 2.2.1.2) which is the most used algorithm for text clustering. As known, k-means performances are strictly dependent from the initialization criterion. Thus to have a better evaluation of the k-means behavior we tested it using three popular initializations described in section 2.2.1.2 (MQ, RC and RP). In order to compare clustering algorithms we used two different data sets:

- **Reuters**: is one of the most used data sets in information retrieval [NIST, 2005]. After an initial cleaning to remove multi-labelled documents, we obtain a corpus of 8538 documents of various lengths organized in 65 mutually exclusive classes. We further remove 10 classes each containing only a single document.

- **Snippets**: is a collection of 30 small corpora of 200 web snippets returned by the Open Directory Project (ODP) search engine in correspondence of the 30 most popular queries according with Google statistics. A more complete description of this dataset and how we established the ground truth can be found in section 4.4.1.1.

The two considered datasets present many differences: Reuters contains much more documents that are much longer than those in Snippets. Documents in Reuters can have a large size variability and are completely flat. Instead, Snippets contains semi-structured documents (a field with the snippet title and one with the snippet body).

Since both the datasets we used are endowed with a manual classification, it was possible to establish a ground truth. We used four measures to validate the clustering correspondence with the ground truth: the *Normalized Mutual Information* (NMI), the *Normalized Complementary Entropy* (NCE), the F-measure and the Accuracy. All these measures are in the range $[0, 1]$ where higher values mean better quality and are described in section 2.3.2.

3.3.1 FPF evaluation

In table 3.1 we show a comparison of the three clustering algorithms for the k-center problem. Each clustering was run using two different distance functions with the goal of comparing also the metrics for text clustering.

As shown in table 3.1, when clustering Reuters data, all the algorithms have a very different behavior depending on the used distance. In all the cases cosine similarity performs better than GJC. Instead, when clustering Snippets, the final clustering quality is always comparable.

Routers data put in evidence also some differences among the clustering algorithms. In fact FPF seems to achieve a worse quality with respect to the others. When clustering Snippets, these differences become much less evident. In fact, the F-measure and accuracy are always comparable; NMI, in the case FPF, is quite

Dataset	Measure	FPF		M-FPF		M-FPF-MD	
		CS	GJC	CS	GJC	CS	GJC
Reuters	NMI	0.180	0.071	0.219	0.077	0.231	0.078
	NCE	0.650	0.542	0.690	0.548	0.701	0.548
	F-mea	0.340	0.286	0.441	0.241	0.389	0.168
	Acc.	0.594	0.488	0.650	0.488	0.646	0.466
Snippets	NMI	0.697	0.683	0.414	0.411	0.422	0.408
	NCE	0.428	0.407	0.678	0.687	0.688	0.684
	F-mea	0.360	0.395	0.338	0.350	0.336	0.356
	Acc.	0.568	0.565	0.542	0.560	0.552	0.563

Table 3.1. Comparison of FPF, M-FPF and M-FPF-MD on the Reuters and Snippets datasets with Generalized Jaccard Coefficient and Cosine Similarity.

higher than that of M-FPF and M-FPF-MD. In contrast NCE of FPF is lower than that of the other algorithms.

3.3.2 k-means evaluation

Table 3.2 reports a comparison of k-means algorithm initialized with three common methods: RC in which the initial centroids are Randomly Chosen, RP [Peña *et al.*, 1999] in which initial centroids are obtained through Random Perturbations and MacQueen's [MacQueen, 1967] in which the initial centroids are randomly chosen, the remaining points are assigned one at time to the nearest centroid and each such assignment causes the immediate recomputation of the involved centroid.

	Algorithm	NMI	NCE	F-mea	Acc.
Reuters	k-means MQ	0.287	0.757	0.332	0.683
	k-means RC	0.305	0.775	0.299	0.743
	k-means RP	0.304	0.775	0.299	0.728
Snippets	k-means MQ	0.721	0.446	0.346	0.585
	k-means RC	0.697	0.422	0.311	0.554
	k-means RP	0.656	0.389	0.239	0.472

Table 3.2. Comparison of k-means with three different initializations on the Reuters and Snippets datasets.

In table 3.2 we observe that when clustering the Reuters data, either RC and RP converge to the same solution. MQ, instead finds a slightly worse solution. In the clustering of Snippets the situation is opposite. In fact, in this case k-means initialized with MQ consistently obtains a better result than using the other initializations.

3.3.3 Clustering running time

In table 3.3 we show running time for all the considered algorithms and variants using both datasets. Note that the k-means running time is strictly dependent from the number of iterations the algorithm performs. There is not a general rule to decide the “right” number of iterations. For the Reuters dataset we constrained k-means to stop when one of the following conditions became true: clustering is stable (i. e. centroids are close to those of the previous iteration), 10 iterations were made. In the case of Snippets we also constrained k-means to stop after a certain time threshold is exceeded. We set this threshold to 5 seconds. This last requirement was introduced to study the k-means behavior in on-line applications.

Table 3.3 has two columns for each dataset. In the first column we report the overall clustering time and in the last column we report the initialization time for k-means (our algorithms do not require an initialization).

	Reuters		Snippets	
Algorithm	Clustering	Initialization	Clustering	Initialization
FPF - CS	70.76	-	0.291	-
FPF - GJC	4.90	-	0.116	-
M-FPF - CS	70.54	-	0.331	-
M-FPF - GJC	5.41	-	0.127	-
M-FPF-MD - CS	88.20	-	0.342	-
M-FPF-MD - GJC	14.55	-	0.151	-
k-means MQ	8663.08	768.30	5.565	0.462
k-means RC	8081.20	74.07	5.584	0.261
k-means RP	25413.29	17301.92	12.416	12.390

Table 3.3. Running time in seconds of all Algorithms, variants and metrics.

A first important observation we can make looking at table 3.3 is that GJC is much faster than cosine similarity. The larger are the documents, the more this difference is evident.

We observed also that k-means initialization running time is a not negligible part of the overall running time. The random perturbation is the slowest initialization, its running time dominates that of the rest of the computation. Not surprisingly the selection of random centers is the fastest initialization.

In the comparison among the algorithms it is evident that the family of FPF based methods is much faster than k-means. Moreover, we observed that the choice of the k-center set (that is the set of points in the solution of the k-center problem 2.2.1.1) runs in comparable time with respect to the choice of random points.

3.3.4 Conclusions

To give a global overview, in table 3.4 and 3.5 we report all performances and running time results for respectively the Reuters and Snippets datasets.

Algorithm	NMI	NCE	F-mea	Acc.	Run Time	Init. Time
FPF - CS	0.180	0.650	0.340	0.594	70.76	-
FPF - GJC	0.071	0.542	0.286	0.488	4.90	-
M-FPF - CS	0.219	0.690	0.441	0.650	70.54	-
M-FPF - GJC	0.077	0.548	0.241	0.488	5.41	-
M-FPF-MD - CS	0.231	0.701	0.389	0.646	88.20	-
M-FPF-MD - GJC	0.078	0.548	0.168	0.466	14.55	-
k-means MQ	0.287	0.757	0.332	0.683	8663.08	768.30
k-means RC	0.305	0.775	0.299	0.743	8081.20	74.07
k-means RP	0.304	0.775	0.299	0.728	25413.29	17301.92

Table 3.4. Performance and running time (in seconds) for the Reuters dataset.

After analyzing the data, it is clear that the perfect clustering algorithm still does not exist and this motivates the effort of researchers. Hence some observations should be done. First of all, the on Reuters data, k-means outperforms algorithms for k-center by a factor of roughly 10% but requires at least two orders of magnitude more time. Using GJC, time and performance differences become more evident. Clearly the best clustering/metric scheme is strictly dependent on the problem requirements and data size. In off-line applications probably computational efficency should be sacrificed in favor of quality, but this is not always true. For example, as we will see in chapter 6, if we must manage a large corpus of data (about 100k documents), the preprocessing time can vary from the order of hours to weeks. When the corpus becomes huge the use of k-means can become impractical.

Algorithm	NMI	NCE	F-mea	Acc.	Run Time	Init. Time
FPF - CS	0.697	0.428	0.360	0.568	0.291	-
FPF - GJC	0.683	0.407	0.395	0.565	0.116	-
M-FPF - CS	0.414	0.678	0.338	0.542	0.331	-
M-FPF - GJC	0.411	0.687	0.350	0.560	0.127	-
M-FPF-MD - CS	0.422	0.688	0.336	0.552	0.342	-
M-FPF-MD - GJC	0.408	0.684	0.356	0.563	0.151	-
k-means MQ	0.721	0.446	0.346	0.585	5.565	0.462
k-means RC	0.697	0.422	0.311	0.554	5.584	0.261
k-means RP	0.656	0.389	0.239	0.472	12.416	12.390

Table 3.5. Performance and running time (in seconds) for the Snippets dataset.

The analysis of Snippets, in proportion, shows essentially no difference in terms of running time with respect to Reuters. Clearly in an on-line setting the use of k-means can result infeasible and FPF must be preferred. Looking at quality performances, there is not a dominating algorithm. NMI and MCE are always in contrast: a high value of NMI corresponds to a poor result for NCE. F-measure and Accuracy are in all the cases stable. We observe also that, in terms of quality performance, the choice of the metric is not influent. Cosine similarity slightly gains in terms of NMI and NCE, while GJC is a bit better in terms of F-measure and accuracy. In terms of running time, instead GJC saves up to the 60% of the time required by cosine similarity.

3.4 Cluster labelling

Once a certain corpus of documents has been clustered, in some applications it is important to have a short (textual) description of its content. For example, in the web snippets clustering we will discuss in chapter 4 a perfect cluster without a good description of its content is not useful. The case of similarity searching, discussed in chapter 6, is instead different since clustering is only a “hidden” tool for the similarity search engine. A cluster label should have at least three desired properties: shortness, syntactically correctness and predictiveness of the cluster content.

In this section we describe in detail our novel approach to cluster labelling. This algorithm, designed for web snippets, is used in *Armil* a web meta search engine described in chapter 4. Our proposed method requires as input the vector space of documents and the clustering; thus it is independent from: the chosen weighting scheme, the feature selection method and the clustering algorithm. The algorithm works in three phases: in the first step it extracts a list of topic representative keywords for each cluster; then it removes some duplicate keywords according with a global criterion; to finish it extracts the best possible sentence from the cluster such that it matches the cluster keywords.

3.4.1 Extracting cluster keywords

In the first phase of the cluster labelling algorithm, we select a certain number of descriptive keywords for the considered cluster. We assume here that documents are represented as bag of words such that each term has a score depending on previous chosen weighting scheme. Moreover, it is also reasonable to assume that feature selection was already done before clustering, thus all not discarded terms should be semantically relevant. Under the above assumptions the more a word is relevant for a document the higher is its score. Thus, the cluster content can be synthesized by the set of words with highest score. We will refer to the words in this set as the *candidate* words. Note that the selection algorithm has scope limited to the considered cluster and does not take into account the content of the other clusters. We will refer to this as *local candidate selection*. While from a certain point of view this means that candidates can be extracted in parallel for each cluster, on the other hand it means that many clusters can have some keywords in common and therefore similar labels. For example this is the case of web snippets in which the query terms are likely to be present in all clusters with high score.

3.4.2 Candidate words selection

The second step overcomes the drawback of the previous step which has only a local scope. The goal here is to decide for which cluster a shared keyword is more appropriate. We can see this as a problem of classification in which each set of candidate words is a category and the keywords that appear in more than set of candidate words must be classified. At the end for each keyword we have an associated set, thus we can remove the word from the other sets.

To classify candidate words we used a modified version of the *information gain* function [Cover and Thomas, 1991; Yang and Pedersen, 1997].

Let x be a document taken uniformly at random in the corpus. $P(t)$ is the probability that x contains term t, $P(c)$ is the probability that x is in category c. The complementary events are denoted $P(\bar{t}) = 1 - P(t)$ and $P(\bar{c}) = 1 - P(c)$. $P(t, c)$ is the probability that x is in category c and contains term t, $P(\bar{t}, \bar{c})$ is the probability x does not contain t and is not in category c. $P(t, \bar{c})$ is the probability that x contains t but is not in category c, $P(\bar{t}, c)$ is the probability that x does not contain t and is in category c. Clearly being these mutually disjoint events it holds: $P(t, c) + P(\bar{t}, \bar{c}) + P(t, \bar{c}) + P(\bar{t}, c) = 1$. The information gain is:

$$IG(t,c) = \sum_{a \in \{t,\bar{t}\}} \sum_{b \in \{c,\bar{c}\}} P(a,b) \log \frac{P(a,b)}{P(a)P(b)}$$

Intuitively, IG measures the amount of information that each argument contains about the other; when t and c are independent, $IG(t, c) = 0$. As explained in section 3.2.1.3 this function is often used for feature selection in text classification, where, if $IG(t, c)$ is high, the presence or absence of a term t is deemed to be highly indicative of the membership or non-membership in a category c of the document containing it.

Examining the information gain formula it is easy to note that it is the summation of four contributions: the presence of the term in the class ($P(t, c) \log \frac{P(t,c)}{P(t)P(c)}$), the absence of the term in the other classes ($P(\bar{t}, \bar{c}) \log \frac{P(\bar{t},\bar{c})}{P(\bar{t})P(\bar{c})}$), the presence of the term in the other classes ($P(t, \bar{c}) \log \frac{P(t,\bar{c})}{P(t)P(\bar{c})}$) and the absence of the term in the class ($P(\bar{t}, c) \log \frac{P(\bar{t},c)}{P(\bar{t})P(c)}$). The first two contributions represent the “positive correlation” between the arguments while the last two factors represent their “negative correlation”.

In the text classification context, the rationale of including both positive and negative correlations is that, the contribution due to the presence or absence of a term is equally indicative of the membership (resp. non-membership) of the document in the category. That is, the term is useful anyway, although in a “negative” sense.

However, in our context we are not interested in the fact that the absence of a keyword in the label increases its information power. We want to measure the contribution of the presence of a keyword in a candidate set to the detriment of the others and hence we are interested to those terms in information gain formula that *positively describe* the contents of a cluster. We modified information gain according with the above consideration:

$$IG_m(t,c) = P(t,c)\log\frac{P(t,c)}{P(t)P(c)} + P(\bar{t},\bar{c})\log\frac{P(\bar{t},\bar{c})}{P(\bar{t})P(\bar{c})}$$

3.4.3 Label generation

Once completed second step, for each cluster we have associated a set of candidate words. The goal of this step is to extract from each cluster the most possible descriptive short sentence based on the candidate words. Thus we can consider all the sentences contained in the documents of the considered cluster as a text corpus, arrange them in a inverted index and use candidate words as query to retrieve related sentences. Clearly in this case we are interested in just a phrase, thus we rank results and extract the label from the sentence on top. Since we want to extract a short sentence of few words, it makes sense to remove from the set of candidate words (query) the terms with low score. In our experiments we reduced these sets to contain only three items and refer to them as the cluster *signature*. Note that we need to make just a query for each cluster, thus in practice we make a linear scan of all the sentences and do not set up the inverted index.

The ranking function should assign a score that takes into account three parameters: the weight of the candidate words present in the phrase, the number of different candidate words retrieved and their inter-distance. Moreover the ranking function should penalize repeated words because they do not add new information to the final user, but make the label longer. In the case of the web snippets we were interested in labels no longer than five words, thus we evaluated the score of all the five word windows in the sentence and return as sentence score the highest value. Let $q = \{q_1, \dots q_h\}$ be the words of the query that produces the snippets corpus, $C = \{c_1, \dots, c_k\}$ the candidate keywords for the cluster $W = \{w_1, \dots, w_n\}$ the words in the considered window (in our case $n = 5$ and $k = 3$) such that w_i has score $s(w_i)$, the ranking of W is:

$$R(W) = \sum_{w_i \in W} R(w_i)$$

where $R(w_i)$ is defined as:

$$R(w_i) = \begin{cases} s(w_i) & \text{if } w_i \in C \wedge \forall j < i \quad w_j \neq w_i \\ 0 & \text{if } w_i \in Q \wedge \forall j < i \quad w_j \neq w_i \\ -1/2 & \text{if } \exists j < i \quad w_j = w_i \\ -1 & \text{otherwise} \end{cases}$$

The window with highest value of R is the candidate label. Clearly the choice of extracting the cluster label from a phrase of the document corpus assures us that the selected sentence is syntactically well written. On the other hand the constraint on the length of the label forced us to use a window with the collateral result of truncating the sentence. To smooth this effect we applied to the candidate label a filter based on the following rules:

- in case of consecutive copies of a word or a pair of words an instance is removed;
- similarly to what is made for stop words, a list of inadmissible "last word" is used to filter out, among the other: prepositions, conjunctions and articles;
- in the case in which the query is formed by a pair of words $Q = \{q_1, q_2\}$, if the label has q_2 as first word, q_1 is inserted before, instead, if q_1 is the last word of the label q_2 is appended;
- in case the same word is the initial and final word of the label an instance is removed (more precisely we remove the last instance if it is not preceded by an inadmissible last word, otherwise we remove the first instance),
- web URLs and e-mail addresses are removed.

All the filter rules are recursively applied until the label reaches a stable form. The output of the filter is the final label. The quality of the resulting labels depends strictly from the quality of the sentences in the corpus and the clustering algorithm. Moreover a mathematical evaluation of the label quality is probably infeasible. For this reasons we do not provide here an evaluation of the labelling algorithm but we delay it to chapter 4 where the clustering and labeling are combined together and a complete system is presented.

Chapter
4

Showcase Armil a clustering based meta-search engine

Abstract

This chapter describes Armil a completely working meta-search engine that groups the Web snippets returned by auxiliary search engines into disjoint labelled clusters. The cluster labels generated by Armil provide the user with a compact guide to assessing the relevance of each cluster to his/her information need. Striking the right balance between running time and cluster well-formedness was a key point in the design of this system. Both the clustering and the labelling tasks are performed on the fly by processing only the snippets provided by the auxiliary search engines, and use no external sources of knowledge. Clustering is performed by means of a fast version of the furthest-point-first algorithm for metric k-center clustering. Cluster labelling is achieved by combining intra-cluster and inter-cluster term extraction based on a variant of the information gain measure. We have tested the clustering effectiveness of Armil against Vivisimo, the de facto industrial standard in Web snippets clustering, using as benchmark a comprehensive set of snippets obtained from the Open Directory Project hierarchy. According to two widely accepted "external" metrics of clustering quality, Armil achieves better performance levels by 10%. We also report the results of a thorough user evaluation of both the clustering and the cluster labelling algorithms.

4.1 Introduction

An effective search interface is a fundamental component in a Web search engine. In particular, the quality of presentation of the search results often represents one of the main keys to the success of such systems. Most search engines present the results of an user query as a ranked list of Web snippets. Ranking algorithms play a crucial role in this approach, since users usually browse at most the 10 top-ranked items. Snippet quality is also an important issue, since good-quality snippets allow the user to determine whether the referred pages match or not his/her information need. In order to provide an useful hint about the real content of the page, a Web snippet includes both the page title and a short text fragment, that often displays the query terms in that context. *Meta-search engines* (MSEs) integrate the items obtained from multiple "auxiliary" search engines, with the purpose of increasing the coverage of the results. However, without an accurate design, MSEs might in principle even worsen the quality of the information access experience, since the user is typically confronted with an even larger set of results (see [Meng *et al.*, 2002] for a recent survey of challenges and techniques related to building meta-search engines). Thus, key issues to be faced by MSEs concern the exploitation of effective algorithms for merging the ranked lists of results retrieved by the different auxiliary search engines (while at the same time removing the duplicates), and the design of advanced user interfaces based on a structured organization of the results, so as to help the user to focus on the most relevant subset of results. This latter aspect is usually implemented by grouping the results into homogeneous groups by means of clustering or categorization algorithms.

In the context of the the World Wide Web, clustering is not only useful for meta-searching. For example, regular search engines use clustering to some extent to avoid showing the user too many semantically-equivalent documents in the first page of results. This activity is close to the classical *duplicate* or *near-duplicate* detection in information retrieval, and one can take advantage of the fact that duplicates and near duplicates are easy to detect via multiple hashing or shingling techniques, and these tasks can be carried out, at least in part, in an off-line setting (see e.g. [Haveliwala *et al.*, 2000]). Moreover, in the duplicate detection activity labelling is not an issue.

This chapter describes the Armil system[1], a meta-search engine that organizes the Web snippets retrieved from auxiliary search engines into disjoint clusters and automatically constructs a title label for each cluster by using only the text excerpts available in the snippets. Our design efforts were directed towards devising a fast clustering algorithm able to yield good-quality homogeneous groups, and a dis-

[1]An armillary sphere (also known as a spherical astrolabe, armilla, or *armil*) is a navigation tool in the form of a model of the celestial sphere, invented by Eratosthenes in 255 BC. Renaissance scientists and public figures were often portrayed with one hand on an armil, since it represented the height of wisdom and knowledge (see `http://en.wikipedia.org/wiki/Armillary_sphere`). The Armil query interface can be freely accessed and used at the url `http://armil.iit.cnr.it/`.

tillation technique for selecting appropriate and useful labels for the clusters. The speed of the two algorithms was a key issue in our design, since the system must organize the results on the fly, thus minimizing the latency between the issuing of the query and the presentation of the results. In Armil, an equally important role is played by the clustering component and by the labelling component. Clustering is accomplished by means of M-FPF-MD, the improved version of the FPF algorithm described in section 2.4.3. The generation of the cluster labels is instead accomplished by means of a combination of intra-cluster and inter-cluster term extraction, based on a modified version of the information gain measure. This approach tries to capture the most significant and discriminative words for each cluster.

One key design feature of Armil is that it relies almost only on the snippets returned by the auxiliary search engines. This means that no substantial external source of information, such as ontologies or morphological and syntactic linguistic resources, is used. We use just stop word lists, stemmers and a very simple language recognition tools. We thus demonstrate that such a lightweight approach, together with carefully crafted algorithms, is sufficient to provide a useful and successful clustering-plus-labelling service. Obviously, this assumption relies on the hypothesis that the quality of the snippets returned by the auxiliary search engines is satisfactory.

We have tested the clustering effectiveness of Armil against Vivisimo the *de facto* industrial standard in Web snippet clustering , using as benchmark a comprehensive set of snippets obtained from the Open Directory Project hierarchy. According to two metrics of clustering quality that are normalized variants of the Entropy and the Mutual Information [Cover and Thomas, 1991], Armil achieves better performance than Vivisimo.

We also report the results of a thorough user evaluation of both the clustering and the cluster labelling algorithms.

4.2 Clustering and labelling of web snippets

Clustering and labelling are both essential operations for a Web snippet clustering system. However, each previously proposed such system strikes a different balance between these two aspects. Some systems (e.g. [Ferragina and Gulli, 2005; Lawrie and Croft, 2003]) view label extraction as the primary goal, and clustering as a by-product of the label extraction procedure. Other systems (e.g. [Kummamuru *et al.*, 2004; Zamir *et al.*, 1997]) view instead the formation of clusters as the most important step, and the labelling phase is considered as strictly dependent on the clusters found.

4.2.1 Related work

Tools for clustering Web snippets have received attention in the research community. In the past, this approach has had both critics [Kural *et al.*, 1999; 1993] and supporters [Tombros *et al.*, 2002], but the proliferation of commercial Web services such as Copernic, Dogpile, Groxis, iBoogie, Kartoo, Mooter, and Vivisimo seems to confirm the potential validity of the approach. Academic research prototypes are also available, such as Grouper [Zamir and Etzioni, 1998; Zamir *et al.*, 1997], EigenCluster [Cheng *et al.*, 2003], Shoc [Zhang and Dong, 2004], and SnakeT [Ferragina and Gulli, 2005]. Generally, details of the algorithms underlying commercial Web services are not in the public domain.

Maarek et al. [Maarek *et al.*, 2000] give a precise characterization of the challenges inherent in Web snippet clustering, and propose an algorithm based on complete-link hierarchical agglomerative clustering that is quadratic in the number n of snippets. They introduce a technique called “lexical affinity” whereby the co-occurrence of words influences the similarity metric.

Zeng et al. [Zeng *et al.*, 2004] tackle the problem of detecting good cluster names as preliminary to the formation of the clusters, using a supervised learning approach. Note that the methods considered in our approach are instead all unsupervised, thus requiring no labelled data.

The EigenCluster [Cheng *et al.*, 2003], Lingo [Osinski and Weiss, 2004], and Shoc [Zhang and Dong, 2004] systems all tackle Web snippet clustering by performing a singular value decomposition of the term-document incidence matrix. The problem with this approach is that SVD is extremely time-consuming, hence problematic when applied to a large number of snippets. By testing a number of queries on Eigencluster we have observed that, when operating on many snippets (roughly 400), a reasonable response time (under 1 second) is attained by limiting the number of generated clusters to a number between 5 and 10, and avoiding a clustering decision for over 50% of the data.

Zamir and Etzioni [Zamir and Etzioni, 1998; Zamir *et al.*, 1997] propose a Web snippet clustering mechanism (Suffix Tree Clustering – STC) based on suffix arrays, and experimentally compare STC with algorithms such as k-means, single-pass k-means [MacQueen, 1967], Backshot and Fractionation [Cutting *et al.*, 1992], and Group Average Hierarchical Agglomerative Clustering (GAHAC). They test the systems on a benchmark obtained by issuing a set of 10 queries to the Metacrawler meta-search engine, retaining the top-ranked 200 snippets for each query, and manually tagging the snippets by relevance to the queries. They then compute the quality of the clustering obtained by the tested systems by ordering the generated clusters according to precision, and by equating the effectiveness of the system with the average precision of the highest-precision clusters that collectively contain 10% of the input documents. This methodology had been advocated in [Hearst and Pedersen, 1996], and is based on the assumption that the users will anyway be able to spot the clusters most relevant to their query. Average precision as computed with this method ranges from 0.2 to 0.4 for all the algorithms tested

(STC coming out on top in terms of both effectiveness and efficiency). Interestingly, the authors show that very similar results are attained when full documents are used instead of their snippets, thus validating the snippet-based clustering approach.

Strehl et al. [Strehl *et al.*, 2000] experiment with four similarity measures (Cosine similarity, Euclidean distance, Pearson Correlation, extended Jaccard) in combination with five algorithms (k-means, self-organizing maps, random clustering, min-cut hyper-graph partitioning, min-cut weighted graph partitioning). Two data sets were used, one formed by 2340 documents pre-classified in 20 categories from the news repository of Yahoo!, the second formed by 966 documents pre-classified in 10 categories from the Business Directory of Yahoo!. Overall quality is measured in terms of normalized mutual information. For a specific clustering also the quality of the single clusters in terms of single cluster purity, single cluster entropy are given. The comparisons are made only in terms of output quality, computing time not being considered relevant in this setting. The highest quality results are obtained via cosine similarity and Jaccard distance combined with min-cut graph partitioning. Our experiments have confirmed that for snippets better results are obtained using variants of Jaccard distance, with respect to standard tf-idf and cosine similarity.

Lawrie and Croft [Lawrie and Croft, 2003] view the clustering/labelling problem as that of generating multilevel summaries of the set of documents (in this case the Web snippets returned by a search engine). The technique is based on first building off-line a statistical model of the background language (e.g. the statistical distribution of words in a large corpus of the English language), and on subsequently extracting “topical terms” from the documents, where “topicality” is measured by the contribution of a term to the Kullback-Leibler divergence score of the document collection relative to the background language. Intuitively, this formula measures how important this term is in measuring the distance of the collection of documents from the distribution of the background language. Additionally, the “predictiveness” of each term is measured. Intuitively, predictiveness measures how close a term appears (within a given window size) to other terms. In the summaries, terms of high topicality and high predictiveness are preferred. The proposed method is shown to be superior (by using the KL-divergence) to a naive summarizer that just selects the terms with highest $tf * idf$ score in the document set.

Kammamuru et al. [Kummamuru *et al.*, 2004] propose a classification of Web snippet clustering algorithms into *monothetic* (in which the assignment of a snippet to a cluster is based on a single dominant feature) and *polythetic* (in which several features concur in determining the assignment of a snippet to a cluster). The rationale for proposing a monothetic algorithm is that the single discriminating feature is a natural label candidate. The authors propose such an algorithm (a type of greedy cover) in which the snippets are seen as sets of words and the next term is chosen so as to maximize the number of newly covered sets while minimizing the hits with already covered sets. The paper reports empirical evaluations and user studies over two classes of queries, “ambiguous” and “popular”. The users were asked to compare 3 clustering algorithms over the set of queries and, for each query, were asked to answer 6 questions of a rather general nature on the generated hierarchy (not on the single clusters).

Ferragina and Gulli [Ferragina and Gulli, 2005] propose a method for hierarchically clustering Web snippets, and produce a hierarchical labelling based on constructing a sequence of labelled and weighted bipartite graphs representing the individual snippets on one side and a set of labels (and corresponding clusters) on the other side. Data from the Open Directory Project (ODP)[2] is used in an off-line and query-independent way to generate predefined weights that are associated on-line to the words of the snippets returned by the queries. Data is collected from 16 search engines as a result of 77 queries chosen for their popularity among Lycos and Google users in 2004. The snippets are then clustered and the labels are manually tagged as relevant or not relevant to the cluster to which they have been associated. The clusters are ordered in terms of their weight, and quality is measured in terms of the number of relevant labels among the first n labels, for $n \in \{3, 5, 7, 10\}$. Note that in this work the emphasis is on the quality of the labels rather than on that of the clusters (although the two concepts are certainly related), and that the ground truth is defined “a posteriori”, after the queries are processed.

4.3 The Armil system

We discuss here in detail the architecture of Armil. Overall the computation flow is a pipeline consisting in:

1. data collection and cleaning,
2. first-level clustering,
3. candidate word extraction for labelling,
4. second-level clustering,
5. cluster labelling.

Let us review these steps in order.

(1) Querying one or more search engines: The user of Armil issues a query string that is re-directed by Armil to the selected search engines (at the moment the user can select Google and/or Yahoo!). As a result Armil obtains a list (or several lists) of snippets describing Web pages that the search engines deem relevant to the query. An important system design issue is deciding the type and number of snippet sources to be used as auxiliary search engines. It is well-known that the probability of relevance of a snippet to the user information need quickly decays

[2] `http://www.dmoz.org/`

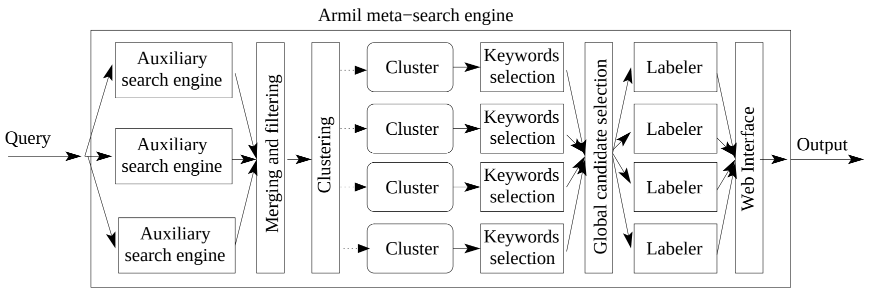


Figure 4.1. The Armil meta-search pipeline.

with the rank of the snippet in the list returned by the search engine. Therefore the need of avoiding low-quality snippets suggests the use of many sources each supplying a low number of high-quality snippets (here high-quality snippet implies both relevance of the snippet to the query and representativeness of the snippet with respect to the associated document). On the other hand, increasing the number of snippet sources raises the pragmatic issue of handling several concurrent threads, the need to detect and handle more duplicates, and the need for a more complex handling of the composite ranking by merging several snippet lists (both globally and within each cluster separately). Since we consider Armil a “proof-of-concept” prototype rather than a full-blown service, we have chosen only two (high-quality) sources, Google and Yahoo!. Since snippet quality decays quickly, we believe that collect 200 snippets is enough reasonable for our purposes. More precisely we query Google asking for 120 results and Yahoo! for 80. This unbalance is only due to some constraints in the number of snippets one can ask to Yahoo!’s interface to be returned in a single query. Thus these numbers optimize the total waiting time. We produce the initial composite ranking of the merged snippet list by a very simple method, i.e. by alternatively picking snippets from each source list.

(2) Cleaning and filtering: Snippets that are too short or with little informative content (i.e. small number of well formed words) or in non-latin alphabets are filtered out. The input is then filtered by removing non-alphabetic symbols, digits, HTML tags, stop words, and the query terms. These latter are removed since they are likely to be present in every snippet, and thus are going to be useless for the purpose of discriminating different contexts. On the other hand query terms are very significant for the user so they are re-introduced in the label generation phase described below. We then identify the prevalent language of each snippet, which allows us to choose the appropriate stop word list and stemming algorithm. Currently we use the ccTLD (Country Code Top Level Domain) of the url to decide on the prevalent language of a snippet. For the purpose of the experiments we only distinguish between English and Italian. For snippets of English Web pages we use Porter’s stemming algorithm, while for Italian ones we use a simple rule-based stemmer we developed in-house. Currently, no other languages are supported. More advanced language discovery techniques are well-known in the literature, however it should be noticed that, given the on-line requirements, methods too expensive in terms of computational effort should be avoided.

(3) First-level clustering: We build a flat k-clustering representing the first level of the cluster hierarchy, using the M-FPF-MD algorithm and the Generalized Jaccard Distance as described in chapter 2. An important issue is deciding the number k of clusters to create. Currently, by default this number is fixed to 30, but it is clear that the number of clusters should depend on the query and on the number of snippets found. A general rule is difficult to find, also because the "optimal" number of clusters to be displayed is a function of the goals and preferences of the user. Moreover, while techniques for automatically determining such optimal number do exist [G. W. Milligan, 1985; Geraci *et al.*, 2007], their computational cost is incompatible with the real-time nature of our application. Therefore, besides providing a default value, we allow the user to increase or decrease the value of k to his/her liking. Clusters that contain one snippet only are probably outliers of some sort, and we thus merge them under a single cluster labelled "Other topics".

(4) Snippets re-ranking: A cluster small enough that the list of its snippets fits in the screen does not require a sophisticated order of presentation. However, in general users are greatly facilitated if the snippets of a cluster are listed in order of their estimated importance for the user. Our strategy is to identify an "inner core" of each cluster and "outliers". In order to achieve this aim we apply the FPF algorithm within each cluster as follows. Since FPF is incremental in the parameter k, we increment k up to a value for which it happens that the largest obtained cluster has less then half of the points of the input cluster. This specific group forms the"inner core", all other points are termed "outliers". The points in the"core" are shown in the listing before the "outliers". Within core and outliers points we use a relative ranking obtained by a linear combination of the native ranking generated by the auxiliary search engines. Note that this computation is done only to decide the order of presentation of the snippets at the first level. It should not be confused with the second-level clustering described below.

(5) Candidate words selection: For each cluster we need to determine a set of candidate words for appearing in its label; these will hereafter be referred to as *candidates*. For this purpose, for each word that occurs in the cluster we sum the weights of all its occurrences in the cluster and pre-select the 10 words with the highest score in each cluster.

We refer to this as *local candidate selection*, since it is done independently for each cluster. For each of the 10 selected terms we compute IG_m, as explained in Section 3.4.2. The three terms in each cluster with the highest score are chosen as candidates. We refer to this as *global candidate selection*, because the computation of IG_m for a term in a cluster is dependent also on the contents of the other clusters. Global selection has the purpose of obtaining different labels for different clusters. At the end of this procedure, if two clusters have the same signature we merge them, since this is an indication that the target number of clusters k may have been too high for this particular query[3].

(6) Second-level clustering: Although the clustering algorithm could in principle be iterated recursively in each cluster up to an arbitrary number of levels, we limit our algorithm to only two levels, since this is likely to be the maximum depth a user is willing to explore in searching for an answer to his/her information needs[4]. Second-level clustering is applied to first-level clusters of size larger than a predetermined threshold (at the moment this is fixed to 10 snippets, excluding duplicates). For second-level clustering we adopt a different approach, since metric-based clustering applied at the second level tends to detect a single large "inner core" cluster and several small "outlier" clusters[5]. The second-level part of the hierarchy is generated based on the candidate words found for each cluster during the first-level candidate words selection. Calling K the set of three candidate words of a generic cluster, we consider all its subsets as possible signatures for second level clusters. A snippet x is assigned to a signature s if and only if all the signature elements are in x and no candidate in $K \setminus s$ is in x. If a signature is assigned too few snippets (i.e. 1) it is considered as an outlier and it is not shown to the user in the second level. Also, if most of the snippets at the first level end up associated to a single signature, then the second-level clusters are not shown to the user since the second-level subdivision would not be any more useful than the first-level subdivision[6].

(7) Labelling: Many early clustering systems would use lists of keywords as the output to show to the user. In order to offer a more syntactically pleasant label we decided to give as output well formed phrases or parts of phrases as it is becoming standard in more recent systems (e.g. [Ferragina and Gulli, 2005]). We use the candidate keywords just as a basis for generating well-formed phrases that will be shown to the user as real cluster labels. For example "meaning life", and "meaning of life" hold the same information but the latter must be grammatically preferred. Given the title of the Web page contained in the snippet, considered as a sequence of words (this time including stop words) we consider all its contiguous subsequences and we give each subsequence a cumulative score as follows: candidates are given a high positive score (its IG_m score), query words a low positive score (set to 0.1), all other words have a negative score (set at -0.2). For labelling a cluster, among all its snippets we select the shortest substring of a snippet title among those having the highest score. This computation can be done efficiently using a dynamic programming approach. The choice of positive and negative weights is to ensure balancing of two conflicting goals: include most of the candidates but few of the non-candidate connecting words in a label. Once a short phrase is selected,

[3]More precisely, we consider the two original clusters with the same signature as second-level clusters, and we produce for each a different second-level label based on the non-signature keywords of those clusters.

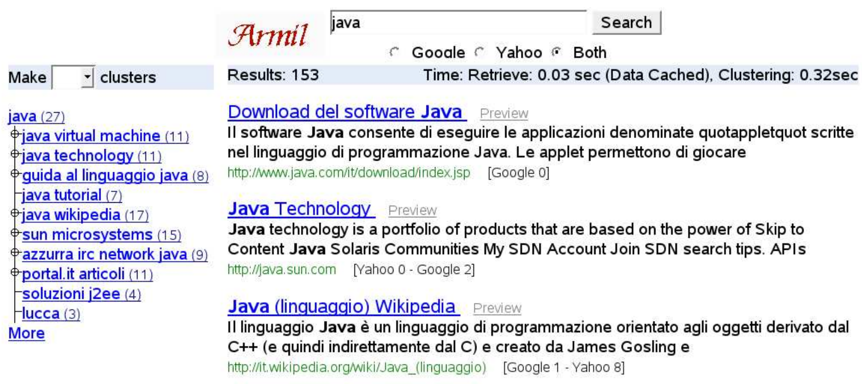
[4]Vivisimo, for example, uses a two-levels hierarchy

[5]We exploited this phenomenon in the snippet re-ranking step (4).

[6]Since second-level clustering is based on first-level candidate words here we depart slightly from the overall idea of separating clustering and labelling.

a rule-based filter is applied to adjust possible grammatical imperfections due to the fact that the selected phrase came from a title whose head and tail were cut and thus the resulting label can appear broken. Filters are better described in section 3.4.3.

(8) Exploiting duplicates: Since Armil collects data from several search engines it is possible that the same URL (maybe with a different snippet fragment of text) is present more than once. We consider this fact as an indication of the importance of the URL. Therefore, duplicates are accounted for in determining weights and distances. Since clustering is based on title and text, it is possible that the same URL ends up in different clusters, for which it is equally relevant. However, if duplicate snippets appear in the same cluster, they are listed only once with an indication of the two sources. Thus duplicate removal is done just before the presentation to the user.



Armil | java | Search

Google Yahoo [✓ Both]

Make clusters

Results: 153   Time: Retrieve: 0.03 sec (Data Cached), Clustering: 0.32sec

java (27)
- java virtual machine (11)
- java technology (11)
- guida al linguaggio java (8)
- java tutorial (7)
- java wikipedia (17)
- sun microsystems (15)
- azzurra irc network java (9)
- portal.it articoli (11)
- soluzioni j2ee (4)
- lucca (3)

More

Download del software Java Preview
Il software **Java** consente di eseguire le applicazioni denominate quotappletquot scritte nel linguaggio di programmazione Java. Le applet permettono di giocare
http://www.java.com/it/download/index.jsp [Google 0]

Java Technology Preview
Java technology is a portfolio of products that are based on the power of Skip to Content **Java** Solaris Communities My SDN Account Join SDN search tips. APIs
http://java.sun.com [Yahoo 0 - Google 2]

Java (linguaggio) Wikipedia Preview
Il linguaggio **Java** è un linguaggio di programmazione orientato agli oggetti derivato dal C++ (e quindi indirettamente dal C) e creato da James Gosling e
http://it.wikipedia.org/wiki/Java_(linguaggio) [Google 1 - Yahoo 8]

Figure 4.2. The Armil meta-search interface.

(9) User Interface: The user interface is important for the success of a Web-based service. We have adopted a scheme common to many search engines and meta-search engines (e.g. Vivisimo), in which the data are shown in ranked list format in the main frame while the list of cluster labels are presented on the left frame as a navigation tree. The interface also allows the user to select the number of clusters, by increasing or decreasing the default value of 30.

4.4 Armil evaluation

We have performed experiments aimed at assessing the performance of Armil. In our tests the behavior of the clustering algorithm, the labeling algorithm and the whole system were compared against the state of the art commercial system Vivisimo.

Vivisimo is considered an industrial standard in terms of clustering quality and user satisfaction, and in 2001 and 2002 it has won the "best meta-search-award" assigned annually by the on-line magazine SearchEngineWatch.com. Vivisimo thus represents a particularly difficult baseline, and it is not known if its clustering quality only depends on an extremely good clustering algorithm, or rather on the use of external knowledge or custom-developed resources.

Vivisimo's advanced searching feature allows a restriction of the considered auxiliary search engines to a subset of a range of possible auxiliary search engines. In particular we restricted it to work over the Open Directory Project (ODP) data. The main advantage of this data is that they came from an handmade classification. By the fact it means that it was possible to establish a ground truth. Thus assessing the quality of a clustering could be done using well-known testing methodologies based on information-theoretic principles.

The problem of measuring the quality of labels and the quality of the whole system is much more complicated. In fact there is not a rigorous method to state how good is a label. To overcome this inconvenient we made an user evaluation that is the standard testing methodology. We performed the study on 22 volunteer master students, doctoral students and post-docs in computer science at our departments (the University of Siena, University of Pisa, and IIT-CNR). The volunteers have all a working knowledge of the English language.

4.4.1 Evaluating clusterings

Following a consolidated practice, we measured the effectiveness of a clustering system by the degree to which it is able to "correctly" re-classify a set of pre-classified snippets into exactly the same categories without knowing the original category assignment. In other words, given a set $C = \{c_1, \ldots, c_k\}$ of categories, and a set Θ of n snippets pre-classified under C, the "ideal" term clustering algorithm is the one that, when asked to cluster Θ into k groups, produces a grouping $C' = \{c'_1, \ldots, c'_k\}$ such that, for each snippet $s_j \in \Theta$, $s_j \in c_i$ if and only if $s_j \in c'_i$. The original labelling is thus viewed as the latent, hidden structure that the clustering system must discover.

The measures we used for evaluating clusterings are: the *normalized mutual information* (NMI) and the *normalized complementary entropy* (NCE) we discussed in 2.3.2.

Higher values of NMI mean better clustering quality. The clustering produced by Vivisimo has partially overlapping clusters (in our experiments Vivisimo assigned roughly 27% of the snippets to more than one cluster), but NMI is designed for non-overlapping clustering. Therefore, in measuring NMI we eliminate the snippets that are present in multiple copies from: the ground truth, the clustering produced by Vivisimo, and that produced by Armil.

However, in order to also consider the ability of the two systems to "correctly" duplicate snippets across overlapping clusters, we have also computed the NCE, in which we have changed the normalization factor so as to take overlapping clusters into account. NCE ranges in the interval $[0, 1]$, and a greater value implies better quality.

4.4.1.1 Establishing the ground truth

The ephemeral nature of the Web is amplified by the fact that search engines have at best a partial view of the available pages relevant to a given query. Moreover search engines must produce a ranking of the retrieved relevant pages and display only the pages of highest relevance. Thus establishing a "ground truth" in a context of the full Web is problematic. Following [Haveliwala *et al.*, 2002], we have made a series of experiments using as input the snippets resulting from queries issued to the Open Directory Project. The ODP is a searchable Web-based directory consisting of a collection of a few million Web pages (as of today, ODP claims to index 5.1M Web pages) pre-classified into more than 590K categories by a group of 75k volunteer human experts. The classification induced by the ODP labelling scheme gives us an objective "ground truth" against which we can compare the clustering quality of Vivisimo and Armil. In ODP, documents are organized according to a hierarchical ontology. For any snippet we obtain a label for its class by considering only the first two levels of the path on the ODP category tree. For example, if a document belongs to class **Games/Puzzles/Anagrams** and another document belongs to class **Games/Puzzles/Crosswords**, we consider both of them to belong to class **Games/Puzzles**. This coarsification is needed in order to balance the number of classes and the number of snippets returned by a query.

Queries are submitted to Vivisimo, asking it to retrieve pages only from ODP. The resulting set of snippets is parsed and given as input to Armil. This is done to ensure that Vivisimo and Armil operate on exactly the same set of snippets, hence to ensure full comparability of the results. Since Vivisimo does not report the ODP category to which a snippet belongs, for each snippet we perform a query to ODP in order to establish its ODP-category.

4.4.1.2 Outcome of the comparative experiment

Similarly to [Ferragina and Gulli, 2005; Kummamuru *et al.*, 2004], we have randomly selected 30 of the most popular queries submitted to Google in 2004 and 2005[7]; from the selection we have removed queries (such as e.g. "Spongebob", "Hilary Duff") that, referring to someone or something of regional interest only, were unlikely to be meaningful to our evaluators. The selected queries are listed in table 4.1.

On average, ODP returned 41.2 categories for each query. In Table 4.2 we report the NMI and NCE values obtained by Vivisimo and Armil on these data. Vivisimo produced by default about 40 clusters; therefore we have run Armil with a target of 40 clusters (thus with a choice close to that of Vivisimo, and to the actual average number of ODP categories per query) and with 30 (this number is the default used in the user evaluation).

The experiments indicate a substantial improvement of about 10% in terms of cluster quality of Armil(40) with respect to Vivisimo. This improvement is an im-

[7]`http://www.google.com/press/zeitgeist.html`

• Airbus	• Harry Potter	• Simpsons
• Chat	• London	• Tiger
• Games	• New Orleans	• Weather
• James Bond	• Star Wars	• Britney Spears
• Mp3	• Wallpaper	• Firefox
• Pink Floyd	• Baseball	• Iraq
• Spiderman	• Ebay	• Matrix
• Tsunami	• Ipod	• Oscars
• Armstrong	• Madonna	• South Park
• Cnn	• Notre Dame	• Tour De France

Table 4.1. Queries.

	Vivisimo	Armil(40)	Armil(30)
NCE	0.667	0.735 (+10.1%)	0.683 (+2.3%)
NMI	0.400	0.442 (+10.5%)	0.406 (+1.5%)

Table 4.2. Results of the comparative evaluation.

portant result since, as noted in 2005 in [Ferragina and Gulli, 2005], “The scientific literature offers several solutions to the web-snippet clustering problem, but unfortunately the attainable performance is far from the one achieved by Vivisimo.” It should be noted moreover that Vivisimo uses a proprietary algorithm, not in the public domain, which might make extensive use of external knowledge. In contrast our algorithm is open and disclosed to the research community.

4.4.2 User evaluation of the cluster labelling algorithm

Assessing “objectively” the quality of a cluster labelling method is a difficult problem, for which no established methodology has gained a wide acceptance. For this reason an user study is the standard testing methodology. We have set up an user evaluation of the cluster labelling component of Armil in order to have an independent and measurable assessment of its performance. We performed the study on 22 volunteer master students, doctoral students and post-docs in computer science at the University of Siena, University of Pisa, and IIT-CNR. The volunteers have all a working knowledge of the English language.

The user interface of Armil has been modified so as to show clusters one-by-one and proceed only when the currently shown cluster has been evaluated. The queries are supplied to the evaluators in a round robin fashion from the 30 predefined queries listed in table 4.1. For each query the user must first say whether the query is meaningful to him/her; an evaluator is allowed to evaluate only queries meaningful to him/her. For each cluster we propose three questions:

(a) Is the label syntactically well-formed?

(b) Can you guess the content of the cluster from the label?

(c) After inspecting the cluster, do you retrospectively consider the cluster as well described by the label?

The evaluator must choose one of three possible answers (Yes; Sort-of; No), and his/her answer is automatically recorded in a database. Question (a) is aimed at assessing the gracefulness of the label produced. Question (b) is aimed at assessing the quality of the label as an instrument predictive of the cluster content. Question (c) is aimed at assessing the correspondence of the label with the content of the cluster. Note that the user cannot inspect the content of the cluster before answering (a) and (b).

Also in this case we used the same set of queries used for evaluating clustering quality, but in this case we do not used the snippets collected from ODP, but we used Google and Yahoo as auxiliary search engines. The two main reasons for this choice are: normally Armil has to deal with those data and not with ODP data, snippets in ODP are typically hand-made and this can introduce a bias in the labeling algorithm.

Each of the 30 queries has been evaluated by two different evaluators, for a total of 60 query evaluations and 1584 cluster evaluations. The results are displayed in the following table:

	Yes	Sort-of	No
(a)	60.5%	25.5%	14.0%
(b)	50.0%	32.0%	18.0%
(c)	47.0%	38.5%	14.5%

Table 4.3. Results of the user evaluation.

Summing the very positive and the mildly positive answers we can conclude that, in this experiment, 86.0% of the labels are syntactically acceptable, 82.0% of the labels are reasonably predictive and 85.5% of the clusters are sufficiently well described by their label. By checking the percentages of No answers, we can notice that sometimes labels considered non-predictive are nonetheless considered well descriptive of the cluster; we interpret this fact as due to the discovery of meanings of the query string previously unknown to the evaluator.

The correlation matrices in Table 4.4 show more precisely the correlation between syntax, predictivity and representativeness of the labels. Entries in the top part give the percentage over all answers, and entries in the bottom part give percentage over rows.

	b-Yes	b-Sort-of	b-No
a-Yes	42.67%	12.81%	5.11%
a-Sort-of	5.74%	15.27%	4.41%
a-No	1.64%	3.78%	8.52%
a-Yes	70.41%	21.14%	8.43%
a-Sort-of	22.58%	60.04%	17.36%
a-No	11.76%	27.14%	61.08%

	c-Yes	c-Sort-of	c-No
b-Yes	33.52%	12.81%	3.72%
b-Sort-of	11.36%	16.85%	3.66%
b-No	2.14%	8.90%	7.00%
b-Yes	66.96%	25.59%	7.44%
b-Sort-of	35.64%	52.87%	11.48%
b-No	11.88%	49.30%	38.81%

	c-Yes	c-Sort-of	c-No
a-Yes	35.98%	18.93%	5.68%
a-Sort-of	8.64%	12.81%	3.97%
a-No	2.39%	6.81%	4.73%
a-Yes	59.37%	31.25%	9.37%
a-Sort-of	33.99%	50.37%	15.63%
a-No	17.19%	48.86%	33.93%

Table 4.4. Correlation tables of questions (a) and (b) (top), (b) and (c) (middle), (a) and (c) (bottom).

The data in Table 4.4 (top) show that there is a strong correlation between syntactic form and predictivity of the labels, as shown by the fact that in a high percentage of cases the same answer was returned to questions (a) and (b).

The middle and bottom parts of Table 4.4 confirm that while for the positive or mildly positive answers (Yes, Sort-of) there is a strong correlation between the answers returned to the different questions, it is often the case that a label considered not predictive of the content of the cluster can still be found, after inspection of the cluster, to be representative of the content of the cluster.

4.4.3 Running time

Since the hardware architecture of Vivisimo is not known, a fair comparison of running time is not possible. Moreover the time needed to enquiry and obtain snippets from the auxiliary search engines depends from many external variables: the available bandwidth or a possible commercial partnership with the queried search engines.

Our system runs on an a Intel(R) Pentium(R) D CPU 3.20GHz, 3Gb RAM and operating system openSUSE 10.2. The code was developed in Python V. 2.5. Excluding the time needed to download the snippets from the auxiliary search engines, the 30 queries have been clustered and labelled in 0.32 seconds on average; the slowest query took 0.37 seconds.

4.5 Conclusions

Why is Armil not “yet another clustering search engine”? The debate on how to improve the performance of search engines is at the core of the current research in the area of Web studies, and we believe that so far only the surface of the vein has been uncovered. The main philosophy of the system/experiments we have proposed follows these lines: (i) principled algorithmic choices are made whenever possible; (ii) clustering is clearly decoupled from labelling; (iii) attention is paid to the trade-off between response time and quality while limiting the response time within limits acceptable by the user; (iv) a comparative study of Armil and Vivisimo has been performed in order to assess the quality of Armil’s clustering phase by means of effectiveness measures commonly used in clustering studies; (v) an user study has been set up in order to obtain an indication of user satisfaction with the produced cluster labelling; (vi) no use of external sources of knowledge is made.

Chapter 5

Clustering for static and dynamic video summaries

Abstract

Video browsing has become one of the most popular activity in the Web. Thus a tool to provide an idea of the content of a given video is becoming a need. Both static and dynamic summarization techniques can be used in this purpose as they set up a video abstract. For static summaries, a set of frames are extracted from the original video to produce a summary, while in the dynamic case short video clips are selected and sequenced to provide the summary. Unfortunately, summarization techniques typically require long processing time and hence all the summaries are produced in advance without any possible user customization. With the large user heterogeneity, this is a burden. In this chapter, we set up an experimental environment where we test clustering performances considering different: categories of video, abstract lengths and low-level video analysis. For summaries we used a fast approximation of M-FPF-MD algorithm. In the static case we worked at frame level, while in the dynamic case we observed that clustering whose input is based on the selection of video scenes performs better than clustering based on video frames. In the end we designed ViSto*, a completely working Web-based customizable service for on-the-fly video summarization to show that, to provide a customized service, fast clustering algorithms should be considered.*

5.1 Introduction

The availability of digital video contents over the web is growing at an exceptional speed due to the advances in networking and multimedia technologies and to the wide use of multimedia applications: videos can be downloaded and played out from almost everywhere using many different devices (e.g., cellphones, palms, laptops) and networking technologies (e.g., EDGE, UMTS, HSDPA, Wi-Fi). The large popularity is highlighted by the enormous success of web sites like Google-Video, YouTube and iTunes Video, where people can upload/download videos. In such a scenario, a tool for performing video browsing would be really appreciated. To handle the enormous quantity of video contents, many proposals have been presented for indexing, retrieving and categorizing digital video contents. In this chapter we focus on *summarization techniques*, which aim at providing a concise representation of a video content. The motivation behind these techniques is to provide a tool able to give an idea of the video content, without watching it entirely, such that an user can decide whether to download/watch the entire video or not. In essence, these techniques are well suited for browsing videos.

In particular, two different approaches are usually followed for producing a concise video representation:

- **static video summary**, which is a collection of video frames extracted from the original video,
- **dynamic video skimming** (or *video abstract*), which is a collection of short video clips.

It is worth mentioning that, in both cases, the output is obtained by analyzing some low-level characteristics of the video stream (e.g., colors, brightness, speech, etc.) in order to find out possible aural/visual clues that would allow a high-level semantics video understanding.

In this chapter we analyze both static and dynamic summarization techniques. For the static case many different techniques have been proposed [Shahraray and Gibbon, 1995; Ueda *et al.*, 1991; Zhuang *et al.*, 1998; Hanjalic and Zhang, 1999b; Gong and Liu, 2003; Hadi *et al.*, 2006; Mundur *et al.*, 2006b], most of them based on clustering techniques. The common key idea is to produce the storyboard by clustering together similar frames and by showing a limited number of frames per cluster (in most cases, only one frame per cluster is selected). With this approach, it is important to select the features upon which frames are considered similar, and different criteria may be employed (e.g., colors distribution, luminance, motion vector, etc.). For the dynamic case, in the literature, different techniques have been proposed (see e.g.[Oh *et al.*, 2004]), but there are relatively few works that address clustering techniques for video skimming. In both the cases the proposed methods usually use computationally expensive and very time-consuming algorithms.

Although existing techniques produce acceptable quality storyboards, they usually use complicated clustering algorithms and thus are computationally expensive and very time consuming. For instance, in [Mundur *et al.*, 2006b] the computation of the storyboard takes around ten times the video length. As a result, this requires video web sites (e.g., The Open Video Project) to pre-compute video abstracts and to present them *as-is*, without offering user customizations. In fact, it is unreasonable to think of an user waiting idle for a latency time comparable to the duration of the original video to get an abstract. This is a burden, as customization is becoming more and more important in the current Web scenario, where users have different resources and/or needs. For instance, a mobile user has less bandwidth than a DSL-connected user, and he/she might want to receive a short storyboard in order to save bandwidth. Conversely, an user who is searching for a specific video scene might want a more detailed storyboard.

The contribution of this chapter is to investigate the benefits of using clustering techniques to produce video abstracts for the Web scenario. More precisely:

- **static storyboards**: we designed an approximated version of M-FPF-MD that takes advantage from the property of HSV vectors to speed up the computation and make the technique suitable for Web video browsing, allowing users to customize the outcome storyboard according to their needs. We also designed a mechanism that suggests a storyboard length based on the video characteristics, but the user can select the length of the storyboard and, thanks to the speed-up of our approach, he/she can re-run the summarization until satisfied with the result.

- **dynamic storyboards**: to this purpose, we set-up an experimental environment considering different clustering algorithms (the well known k-means and the approximated M-FPF-MD), different categories, both in color and in motion terms, of videos (cartoon, TV-show and TV-news), different abstract lengths (2 minutes and 4 minutes), different low-level video analysis (frame-based with HSV color distribution of every frame and scene-based with HSV color distribution of every scene). Note that a video scene is a sequence of consecutive video frames that begins and ends with an abrupt video transition and a silence.

The evaluation of video summaries is done by investigating both the storyboard generation time and the storyboard quality and by comparing the results with other clustering based approaches like k-means [Phillips, 2002], Open Video [Open-Video, 2002], and DT Summary [Mundur *et al.*, 2006b].

Results show that our approximated M-FPF-MD requires one order of magnitude less time than the fastest of the other clustering algorithms. This allowed us to set up a complete web based system (called ViSto) able to make summaries on the fly. Furthermore, the storyboard quality investigation (measured through a Mean Opinion Score) shows that the storyboard quality is comparable to the other approaches.

5.2 Video summary overview

5.2.1 The metric space and distances

Each video frame can be described with a histogram of color distribution. This technique is simple to compute and also robust to small changes of the camera position and to camera partial occlusion. Among the possible color spaces, we consider one supported by the MPEG-7 standard, namely the HSV.

HSV defines the color space in terms of three components: *Hue* (the dominant spectral component, that is the color in its pure form), *Saturation* (the intensity of the color, represented by the quantity of white present) and *Value* (the brightness of the color).

According to the MPEG-7 generic color histogram description [Manjunath *et al.*, 2001], we consider the color histogram as composed of 256 bins. Hence, for each input frame, we extract a 256-dimension vector, which represents the 256 bin colors histogram in the HSV color space of the given video frame. The vectors are then stored in a matrix F whose row f_i represents the i-th HSV vector (or equivalently we use f_i as the i-th frame of the video F).

5.2.2 Related work on static video summarization

Different approaches have been proposed in the literature to address the problem of summarizing a video stream. The most common approach relieves on two phases: firstly a set of video shots is extracted from the video, then for each shot, one or more key-frame are returned. Usually, one (the first) [Shahraray and Gibbon, 1995] or two (the first and the last) [Ueda *et al.*, 1991] key-frames are chosen. A drawback of this approach is that, if the shot is dynamic, the first (or the last) frame may not be the most representative one and hence different approaches, like clustering techniques, have been proposed.

In [Zhuang *et al.*, 1998] the authors propose a clustering algorithm to group video frames using color histogram features. As reported in [Mundur *et al.*, 2006b], the approach does not guarantee an optimal result since the number of clusters is pre-defined by a density threshold value. [Hanjalic and Zhang, 1999b] presents a partitioned clustering algorithm where the key-frames that are selected are the ones closest to each cluster centroid. In [Mundur *et al.*, 2006b] an automatic clustering algorithm based on Delaunay Triangulation (DT) is proposed; here frames are described through HSV color space distribution. Instead of color space distribution, [Hadi *et al.*, 2006] uses local motion estimation to characterize the video frames and then an algorithm based on the k-medoids clustering algorithm is used.

Although the produced storyboards may achieve a reasonable quality, the clustering computational time is the main burden of these approaches. In fact, the extraction of the video features may produce an enormous matrix (depending on the number of frames that compose the video, i.e. the matrix rows and on the number of features that represents each single frame, i.e., the matrix columns). For this reason, mathematical techniques are used in the attempt to reduce the size of the matrix. For instance, [Gong and Liu, 2003] applies the Singular Value Decomposition to the matrix, while [Mundur *et al.*, 2006b] uses the Principal Component Analysis. Needless to say, this requires additional processing time. Another common approach assumes that frames contain a lot of redundant information and hence, instead of considering all the video frames, only a subset is taken (the so-called *pre-sampling* approach) (e.g., [Mundur *et al.*, 2006b]).

Our proposal does not use any mathematical technique to reduce the video feature matrix, and the decision of using the pre-sampling is left to the user. We present the expected storyboard generation time for different pre-sampling rates (or no sampling) and the user will decide either to use sampling or not, eventually selecting the most appropriate rate.

5.2.3 Related work on dynamic video skimming

Different approaches have addressed the problem of video skimming [Oh *et al.*, 2004; Truong and Venkatesh, 2007]. In general one can classify the proposed methods according to several categorical axis:

- the data domain (generic, news, home videos, etc.),
- the features used (visual, audio, motion, etc.),
- the intent (personalization, highlights, information coverage),
- the duration (defined a priori, a posteriori, or user-defined).

Here we focus on techniques for generic videos, using only visual and audio features.

- **Sampling based methods** Authors in [Nam and Tewfik, 1999] propose a frame sampling technique where the sampling rate is proportional to a local notion of ”visual activity”. Such sampling based methods produce quickly shorter videos but suffer from uneven visual quality, and visual discomfort, and are usually not suitable for dealing with the associated audio trace.
- **Frame-based methods** In principle any method for selecting a static storyboard can be turned into a dynamic one by selecting and concatenating the shots/scenes containing the key-frames of the storyboard. For example, the method in [Hanjalic and Zhang, 1999a] works at frame level using a partitional clustering method applied to all the video frames. The optimal number of clusters is determined via a cluster-validity analysis and key frames are selected as centroids of the clusters. Video shots, to which key frames belong, are concatenated to form the abstract sequence. In this approach the dynamic efficiency/quality depends directly from those of the static case.

- **Scene-Based methods** Authors in [Tan and Lu, 2003] formulate the problem of producing a video abstract as a graph partitioning problem over a graph where each node is associated to a shot, and an edge is set up if the similarity of two shots is higher than a pre-defined threshold. The corresponding incidence matrix is clustered using an iterative block ordering method. One can notice that setting up the graph is already quadratic in the number of shots, thus this method is likely unsuitable for on-the-fly processing of long videos. The notion of a *scene transition graph* is also used in [Yeung and Yeo, 1996]: a complete link hierarchical agglomerative clustering is used together with a time-weighted distance metric, introducing an overhead that is unsuitable for on-the-fly computations. Authors in [Ngo *et al.*, 2005] use a *scene transition graph* that is clustered via spectral matrix decomposition. In this case, the mechanism needs 23 minutes to analyze a 69 minutes video. Once again, the approach is unsuitable for on-the-fly operations.

As mentioned, customization and generation of on-the-fly abstracts are very important properties. Hence, an analysis of the benefits introduced by clustering techniques is necessary. In fact, many different methods are surveyed in [Truong and Venkatesh, 2007], but none claims to have the on-the-fly performance and the user-oriented flexibility that is needed in web video browsing applications.

5.3 Clustering to produce static video storyboards

The goal of clustering is to group together similar frames and to select a representative frame per each group to produce the storyboard sequence. In particular, for storyboard generation using clustering, the following main choices must be done:

- **vector space and distance function**: while the vector space is typically chosen among standard video representations, the distance function should reflect the semantic distance between couples of frames,
- **clustering algorithm**: having in mind the goal of producing the video summary on the fly, allowing to set up a service available on the web, clustering must be as faster as possible without sacrifying quality. The algorithm should cluster quickly thousands of frames,
- **suggesting the number of frames**: the choice of the size of the storyboard is not less important. While from a point of view, leaving free the user to set the storyboard size is among the desiderata, from the other side at least an automatic suggestion can be helpful. In fact in a storyboard with too few frames some important details can be lost, while a too long storyboard can cause repetition of similar frames and the introduction of noise,
- **selection of a representative frame for each cluster**: especially in the case of storyboard with few frames, each cluster can contain frames not so similar

among them, hence a criterion to select a frame representative for the cluster represent a strategic choice for the overall quality of a clustering-based summarization system.

In 5.3.1 we describe the result of our investigations for all the previous described choices and show *ViSto* which is a complete web-based system for static video summarization.

5.3.1 Showcase: ViSto Video Storyboard

In this section we describe ViSto a Visual VIsual STOryboard system for Web Video Browsing. ViSto is a completely working system for static video summarization. It is the result of our investigations about a fast enough clustering to allow on-line computation and good enough to be considered helpful for users. To achieve this goals, we used an approximated version of the M-FPF-MD described in chapter 2.

The characteristics of ViSto are very important in the current Web scenario and will be more and more important in future years. Although a possible storyboard length is suggested, ViSto allows users to customize the storyboard by selecting the number of video frames that compose the storyboard.

Also the storyboard generation time can be customized. In fact, since this time depends on the original video length, ViSto estimates the time necessary to produce the storyboard and gives the user the possibility of requiring a video pre-sampling.

Pre-sampling is a technique largely used to reduce the clustering time (for instance, the mechanism proposed in [Mundur *et al.*, 2006b] uses it). It is based on the assumption that all consecutive frames within a certain small time interval (i.e. a second) are likely to be redundant, therefore considering only one of them will suffice. By using a sampling rate, the number of video frames to analyze can be drastically reduced. Needless to say, the sampling rate assumes a fundamental importance, as the larger this sampling rate is, the shorter is the clustering time, but the poorer results might be. For this reason, ViSto simply estimates the time necessary for producing the storyboard using different sampling rates and leaves to the user the decision of using sampling and eventually selecting the desired sampling rate.

As shown in Figure 5.1, ViSto is composed of three phases: first, the video is analyzed in order to extract the HSV color description; second the clustering algorithm is applied to the extracted data and third, a post-processing phase aims at removing possible redundant or meaningless video frames from the produced summary.

Vector space and distance function Using HSV as vector space to represent frames, all the distances defined in 2.1.1 can be used. To better understand which distance performs better, we tested L_1, L_2 and the *Generalized Jaccard Coefficient* (GJC) over a set of 50 videos from the Open Video Project [Open-Video, 2002].

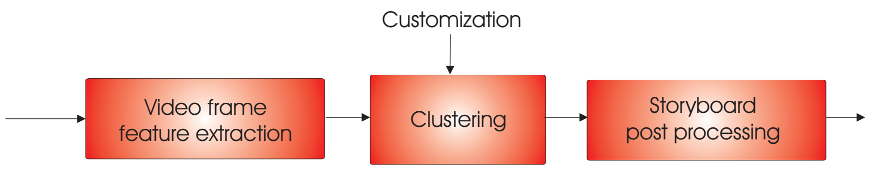


Figure 5.1. The three-steps ViSto Scheme.

Recall that GJC in the HSV space can be defined as follow: given two HSV vector histograms $s = (s_1, ...s_{256})$ and $z = (z_1, ...z_{256})$

$$GJC(s, z) = 1 - \frac{\sum_i \min(s_i, z_i)}{\sum_i \max(s_i, z_i)}.$$

GJC is proven to be a metric [Charikar, 2002].

In our tests, for each video we measured the pairwise distance between pairs of consecutive frames. Figure 5.2 shows an example of how these distances are distributed, along time. For all the metrics we took into account the distance between similar frames is small. Instead, in the case of dissimilar frames, it is possible to observe that GJC tends to return more spread values than L_1 and L_2 with respect to the values returned for pairs of similar frames. Thus GJC highlights better the differences between frames. The figure shows also that the distribution generated via GJC tends to have a peak in correspondence of a fade in the video (It can not be established the type of fade).

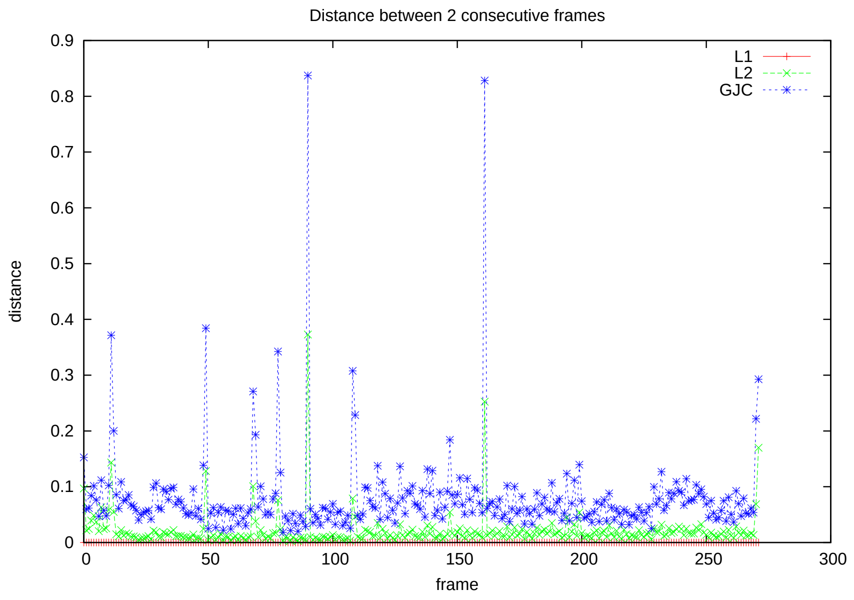


Figure 5.2. Pairwise distances of consecutive frames.

Clustering algorithm For clustering we used an approximated procedure based on the M-FPF-MD algorithm described in chapter 2. We report here the algorithm from scratch: given n frames and a number k of desired clusters, a sample of $\sqrt{nk}$ frames are selected and clustered in k non overlapping clusters by means of the FPF algorithm. The remaining points are added one by one in the cluster corresponding to the closest medoid. Medoids are updated accordingly. We already described some techniques we developed and successfully applied to improve the speed of this schema without modifying its output. For video summarization we want to go a step forward. We observed that the two main costly operations in this algorithm are: the assignment of a new point to its closest cluster that may require up to k distance computations and the recomputation of the medoid whose computational cost, in the worst case, is linear in the number of elements of its cluster.

We also noted that, in the task of video summarization, where only a representative element for cluster is really used and the rest are ignored, the insertion of a point in the wrong cluster does not corrupt the final output quality since it has no effect in the choice of the representative frame. Thus, for this problem, it can be considered acceptable an approximated clustering algorithm in which points are inserted in the cluster that approximatively (and more probably) is the correct one.

Before to describe how we choose the cluster in which to insert a new point, two considerations must be done:

- as shown in figure 5.2 the distance between two consecutive frames tends to be small with some peak when the scene change in some way. While for clustering tasks clearly this is not true in general, in the case of video data this distribution occurs almost always,

- consider two consecutive points p_i and p_{i+1} and let $d(p_i, p_{i+1})$ be their distance. Let c the closest medoid to p_i. Due to the triangular inequality, the distance $d(c, p_{i+1}) \leq d(c, p_i) + d(p_i, p_{i+1})$. Given another medoid c_j, if $1/2d(c, c_j) > d(c, p_i) + d(p_i, p_{i+1})$, then $d(c, p_{i+1}) \leq d(c_j, p_{i+1})$.

We make the, not necessarily true, hypothesis that medoids are far from each other. Therefore, if the distance between two consecutive points p_i and p_{i+1} is small enough, and p_i was assigned to cluster c_j we decide that also p_{i+1} is assigned to c_j.

Also for the recomputation of medoids we use an approximated algorithm. Let $P = p_{i+1}$ the last point inserted in a cluster, a and b the two diametral points of the cluster and m its medoid. After the clustering of the first $\sqrt{nk}$ frames with FPF, for each cluster its medoid is computed using the standard algorithm described in section 2.4.3. Note that, after this computation, the distances $d(m, a)$, $d(m, b)$ and $d(a, b)$ are available. The medoid update can be done using the following approximate procedure:

1. Compute the distances $d(P, a)$ and $d(P, b)$,

2. if $d(P, a) > d(a, b) \wedge d(P, a) > d(P, b)$ discard b, replace it with P and set $d(a, b) = d(P, a)$,

3. if $d(P, b) > d(a, b) \wedge d(P, b) > d(P, a)$ discard a , replace it with P and set $d(a, b) = d(P, b)$,

4. if $d(a, b) > d(P, a) \wedge d(a, b) > d(P, b)$:

   (a) if $|d(P, a) - d(P, b)| + |d(P, a) + d(P, b) - d(a, b)| < |d(m, a) - d(m, b)| + |d(m, a) + d(m, b) - d(a, b)|$ discard m and P becomes the new medoid. Set $d(m, a) = d(P, a)$ and $d(m, b) = d(P, b)$,

   (b) otherwise discard P

Note that since the above procedure keeps updated the distances $d(m, a)$, $d(m, b)$ and $d(a, b)$; only 2 distances must be computed: $d(P, a)$ and $d(P, b)$.

Suggesting the number of frames Although customization allows the user to freely choose the number of frames in the storyboard, we can not exclude the case in which the user has no idea of what such a number might be the “right” one. Hence, we implemented a fast way to make a reasonable estimate of the number of frames that better represents the entire video (denoted with k). This number is always suggested to the user and is used as a default value in case the user does not give any other preference.

We first take a sample $F' \subseteq F$ of the frames of the entire video, taking one out of ten consecutive frames. We then compute the pairwise distance d_i of consecutive frames f'_i, f'_{i+1}, according to GJC, for all such pairs in F'. Figure 5.2 shows an example of how these distances are distributed, along time. We observe that there are instants of time in which the distance between consecutive frames varies considerably (corresponding to peaks), while there are longer periods in which the d_i’s variance is small (corresponding to very dense regions). Usually, peaks correspond to sudden movement in the video or to scene change, while in dense regions frames are more similar one to the other.

To estimate k we count the number of peaks using the following procedure:

1. Order all the d_i’s in increasing order and, for each value v assumed by the d_i’s, count how many pairwise distances are equal to v, i.e. let $t(v) = |\{i \mid d_i = v\}|$;

2. Determine the value Γ for which the function $t(v)$ shows a consistent decreasing step and throw away all the frames that are closer than Γ to their successive; *i.e*, $F'' = \{f'_i \in F' \mid D(f'_i, f'_{i+1}) > \Gamma\}$;

3. Consider the set F'' of remaining frames and count in how many “separated” sets, according to time, they group; *i.e.*, partition F'' into an appropriate number of sets such that if f'_{i_j} and $f'_{i_{j+1}}$ belong to the same set, then

$i_{j+1} - i_j < T$, where T is a small interval of time (meaning the two frames are displayed one shortly after the other).

The number of sets into which F'' is partitioned gives the number of peaks. The number k of frames suggested to the user is set to the number of peaks minus one (videos usually begin and end with a peak).

To test if frame sampling influences the estimate of k, we considered our prediction method using all the frames in a video and using only a sample of frames (one out of ten). We tested all the 50 video in [Mundur *et al.*, 2006a] and we found out that the estimate of k is exactly the same for 47 videos, while it differs by ± 1 for the remaining three. We conclude that the prediction method is not affected by sampling.

The prediction method (with sampling) applied to the 50 videos in [Mundur *et al.*, 2006a] took on the average 0.1 seconds to estimate k (with values spanning from 0.22 to 0.04 seconds).

Selection of a representative frame for each cluster The content of each cluster can be heterogeneous and the distance between a pair of frames inside a cluster can be high. This is due to the fact that the number of clusters is independent from the number of video cuts in the original video, but it depends from the length of the storyboard the user required. Moreover there are frames that are intrinsically not informative (i.e. black frames due to video cuts). In absence of a semantic way to determine which frame of the cluster is the most appropriate as representative frame for each cluster, the most natural choice in our schema was to select the medoid.

5.3.2 ViSto evaluation

ViSto is evaluated through a comparison study with other approaches: an accelerated version of k-means [Phillips, 2002], the Delaunay-based technique (DT) [Mundur *et al.*, 2006b] and the one used by the Open Video Project [Open-Video, 2002].

The study is carried out with two different sets of videos: one is taken from [Mundur *et al.*, 2006a] and is a subset of short videos available within the Open Video Project [Open-Video, 2002] (MPEG-1 encoded with a resolution of 352x240); the second set is composed of long entertainment and informative videos (e.g., cartoon, TV-shows and TV-news), MPEG-1 encoded with a resolution of 352x288.

Note that we consider different types of videos in order to evaluate our approach under different conditions with respect to color and motion. All the experiments have been performed using a Pentium D 3.4 GHz with 3GB RAM, with the aim of investigating two different parameters: the time necessary to produce the storyboard and the quality of the produced summary.

5.3.2.1 Storyboard generation time

The time necessary to generate a video summary of a given video is an important parameter to decide whether a mechanism can be used to produce on-the-fly summaries or not. Therefore, we evaluate the processing time for different videos with different lengths.

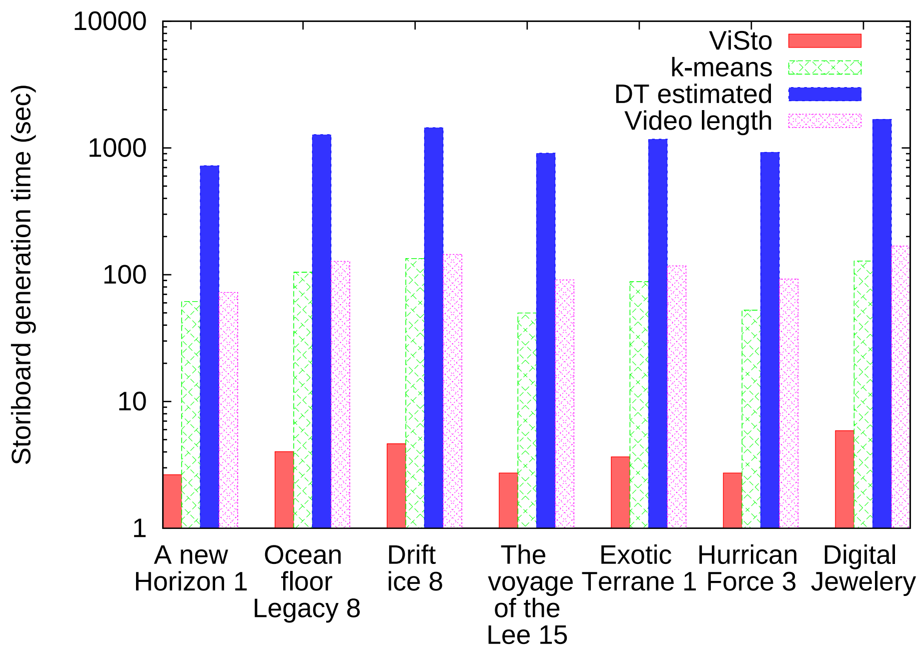


Figure 5.3. Storyboard generation time: DT vs k-means vs ViSto [Logarithmic scale].

Figure 5.3 presents results obtained analyzing a set of seven different videos [Mundur *et al.*, 2006a], whose length spans from the 72 seconds (A New Horizon 1) to 168 seconds (Digital Jewelry). Since no statement is given about the time needed to build the storyboards in the Open Video Project [Open-Video, 2002], as well as nothing is said about the running time of the method on which the project is based [DeMenthon *et al.*, 1998], here we compare our ViSto approach, with k-means and with DT [Mundur *et al.*, 2006b].[1] In the figure we also report the entire video length. Note that results are presented on a logarithmic scale, due to the considerable difference among the compared techniques. It can be observed that the usage of k-means and of DT is not reasonable to produce on-the-fly summaries; in fact, k-means, to produce the summary, needs a time comparable with the video length, while DT needs about 1000 seconds. Conversely, ViSto needs less than 10

[1]Results of DT are simply estimated using the value described in [Mundur *et al.*, 2006b], where it is reported that the mechanism requires between 9 and 10 times the video length to produce the summary.

seconds and hence is well suited to produce on-the-fly summaries. Roughly, in all the tests, ViSto is 25 times faster than k-means and 300 times faster than DT.

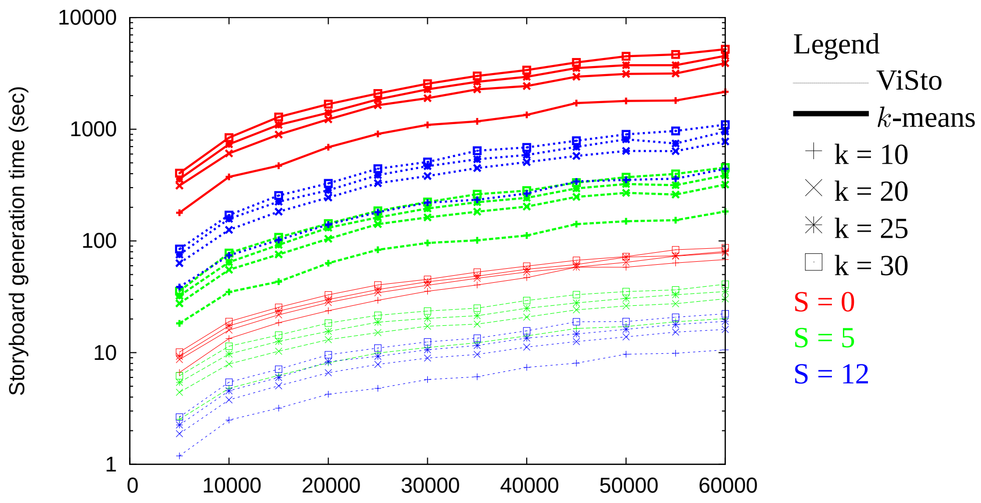


Figure 5.4. Storyboard generation time: A comparison between k-means and ViSto with and without sampling (Sx, with $x = 5, 12$, is the sample rate) [Logarithmic Scale].

A more general investigation on the time necessary to produce a storyboard is presented in Figure 5.4. We vary the length of the given video from 5000 frames (200 seconds) to 60000 frames (40 minutes), the length of the produced storyboard (10, 20, 25 and 30 frames) and the rate of the pre-sampling (none, 1 out of 5 and 1 out of 12) that is applied to the video frame feature matrix. We compare k-means and ViSto (the code of the other approaches is not available).

Once again, due to the large difference between the results of the two approaches, the storyboard generation time is presented on a logarithmic scale. With no surprise, the storyboard generation time depends on its length (the longer the storyboard is, the longer is the computational time) and on the pre-sampling rate (the higher the sampling rate is, the shorter is the computational time). This applies to both approaches. Results confirm that k-means requires a generation time that causes the method to be unsuitable for on-the-fly video summarization: with no doubts, 178 seconds to summarize a 200 seconds video is too much, not to mention the 36 minutes (2165 seconds) required to summarize a 40 minutes video (60000 frames). Only with a pre-sampling of 1 out of 12, k-means can be used for short videos (18 seconds required for a 200 seconds video), but not for longer videos (183 seconds required for a 40 minutes video). To better understand the ViSto behavior, Figure 5.5 presents a detailed close-up of Figure 5.4. The ViSto storyboard generation time with no sampling is reasonable only for videos whose length is up to 15000 frames (10 minutes). In fact, it is not thinkable to let the user wait for more than 20-25 seconds. For longer videos, a sampling of 1 out of 5 frames produces a waiting time no longer than 20/25 seconds for videos up to 35000 frames

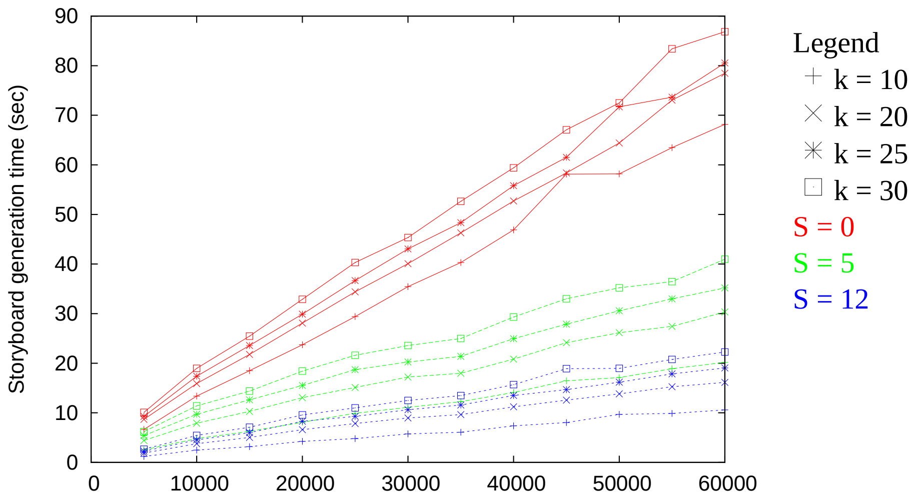


Figure 5.5. ViSto Storyboard generation time with and without sampling (Sx, with $x = 5, 12$, is the sample rate).

(23 minutes). For video larger than 35000 frames, a pre-sampling of 1 out of 12 frame should be considered.

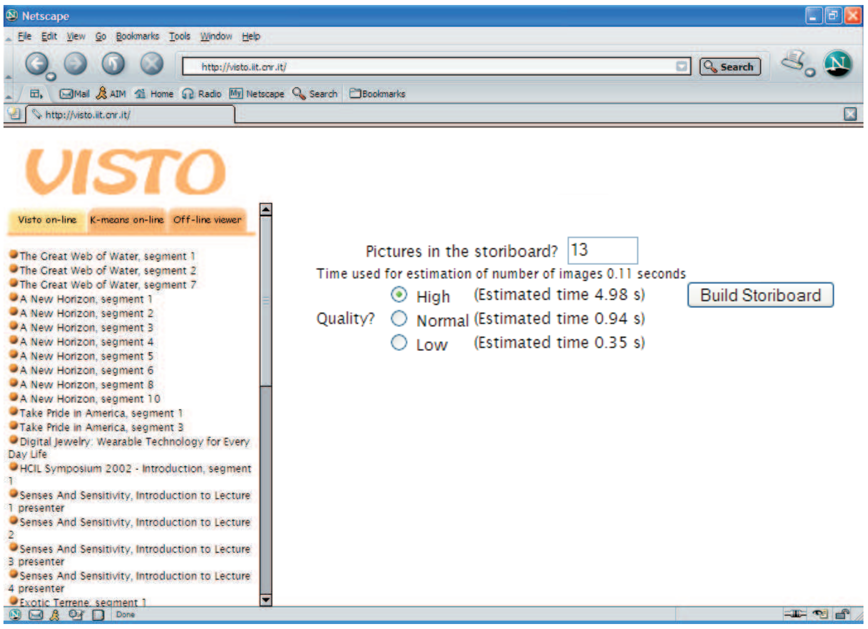
Since the pre-sampling rate might affect the storyboard quality, we let the user select whether to apply a sampling or not. Figure 5.6 shows the ViSto interface, where a user, in addition to the storyboard length, can also select the quality of the storyboard based on the time he/she is willing to wait.

5.3.2.2 Storyboard quality

The time necessary to produce a video storyboard is an important issue, but the quality of the produced storyboard is even more important. In fact, bad quality storyboards (i.e., storyboards that do not well represent the video content) are useless, no matter if they are generated in an instant. For this reason, in the following we investigate the quality of the video summaries produced by ViSto.

The storyboard quality evaluation was carried out by comparing the ViSto results with the one of the Open Video Project, the DT-summary and the k-means. We made two sets of experiments: one with short videos using the data set of [Mundur *et al.*, 2006a] and the other with long videos recorded from TV. Unfortunately, since both Open Video and DT-summary softwares (or details to reimplement them) are not freely available, we could compare with them only using data of [Mundur *et al.*, 2006a] for which the output storyboards of both algorithms are available. The set of experiments using long videos was done comparing ViSto only against k-means.

Quality evaluation is investigated through a Mean Opinion Score (MOS) test; in particular, we asked a group of 20 people with different background (Ph.D. students, graduate students, researchers) to evaluate the produced summaries. The pro-



- The Great Web of Water, segment 1
- The Great Web of Water, segment 2
- The Great Web of Water, segment 7
- A New Horizon, segment 1
- A New Horizon, segment 2
- A New Horizon, segment 3
- A New Horizon, segment 4
- A New Horizon, segment 5
- A New Horizon, segment 6
- A New Horizon, segment 8
- A New Horizon, segment 10
- Take Pride in America, segment 1
- Take Pride in America, segment 3
- Digital Jewelry: Wearable Technology for Every Day Life
- HCIL Symposium 2002 - Introduction, segment 1
- Senses And Sensitivity, Introduction to Lecture 1 presenter
- Senses And Sensitivity, Introduction to Lecture 2
- Senses And Sensitivity, Introduction to Lecture 3 presenter
- Senses And Sensitivity, Introduction to Lecture 4 presenter
- Exotic Terrene, segment 1

Pictures in the storiboard? 13
Time used for estimation of number of images 0.11 seconds
Quality? [✓ High] (Estimated time 4.98 s) Normal (Estimated time 0.94 s) Low (Estimated time 0.35 s)
Build Storiboard

Figure 5.6. ViSto: Length and quality of the storyboard can be easily customized. ViSto is available at `http://visto.iit.cnr.it`.

cedure was the following: we first showed them the video and then the summary, asking whether the summary was a good representation of the original video.

The quality of the video summary was scored on a scale 1 to 5 (1=bad, 2=poor, 3=fair, 4=good, 5=excellent) and people were not aware of the mechanism used to produce the video summary. The length of the produced summary was set in order to match the other approaches (i.e., if the Open Video summary was of 5 frames, the length of the ViSto summary was set to 5 frames, too).

Figure 5.7 reports the results obtained when evaluating short videos obtained from [Mundur *et al.*, 2006a]: *A new Horizon 1* (72 seconds long), *Ocean floor Legacy 8* (127 seconds long), *Drift ice 8* (144 seconds long), *The voyage of the Lee 15* (90 seconds long), *Exotic Terrane 1* (117 seconds long), *Hurricane Force 3* (92 seconds long) and *Digital Jewelery* (168 seconds long). With the exception of *Digital Jewelery* for the DT method and *A new Horizon* for Open Video, these methods achieve poor results. ViSto achieves the best score for *Hurricane Force 3*, *Exotic Terrain 1* and *The voyage of the Lee 15*. With respect to the remaining videos, ViSto and k-means achieve comparable results.

Figure 5.8 presents the summaries of the *A new Horizon 1* video, where Open Video, k-means and ViSto achieve comparable results. As the MOS reported, it is possible to note that the output of the three storyboards achieve a comparable quality.

Figure 5.9 presents the summaries generated by ViSto and k-means for the video *Exotic Terrane 1*. The video is a documentary that shows a mountain land-

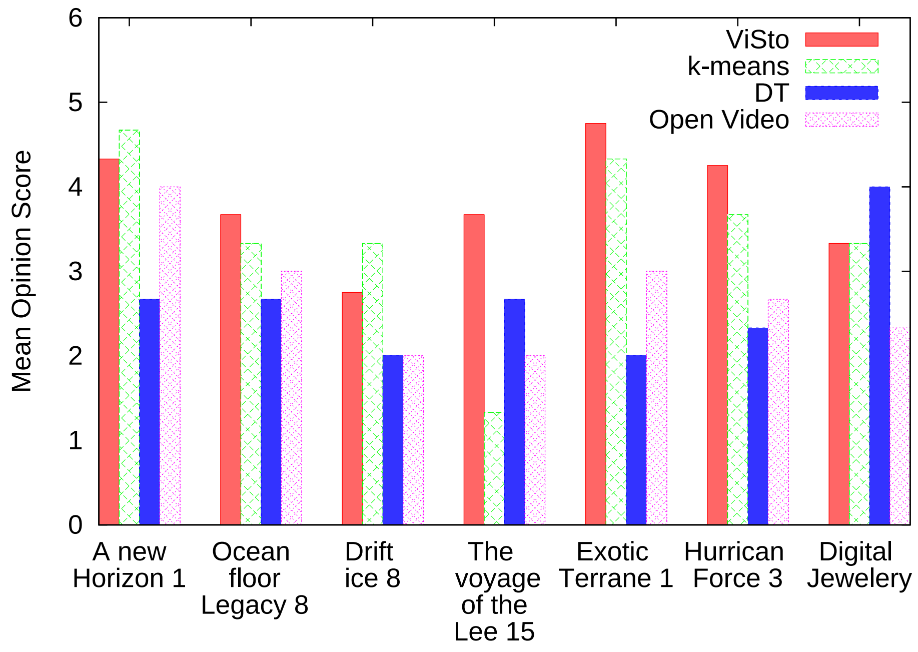


Figure 5.7. Mean Opinion Score of different storyboard of short videos.

scape with some animals. Although some frames are the same in both summaries, ViSto shows a view from the sky and a frame with the video title (last two frames).

Figure 5.10 reports the MOS results obtained when evaluating long videos: *The Simpsons* (20 minutes long), *TV-News* (30 minutes long), *Lost* (40 minutes long) and Talk-Show (15 minutes long). Due to the length of these videos, we produce two different storyboards: one with 15 frames and the other with 30 frames.

Results are comparable for *Lost* and for *TV-News* and different for *The Simpsons* (k-means achieves better results) and for *talk-show* (ViSto achieves better results). These two latter cases are detailed in Figure 5.11 and in Figure 5.12, where the difference is quite clear. In particular, it is interesting to observe that the summaries of Figure 5.11 are completely different, although related to the same video. This can be explained considering the nature of the video taken into consideration: first, just a very small number of frames (15) composes the the storyboard of a video containing a much larger number of frames (30,000); second, in this video, many frames have the same background color and show a yellow character, resulting in high color similarity of frames. Observe that the ViSto summary is composed by frames that show significant color differences. On the other side, the summary of Figure 5.12 shows how some of the key-frames selected by k-means are very similar one to the other, while ViSto gives a more comprehensive overview of the people participating to the talk show.

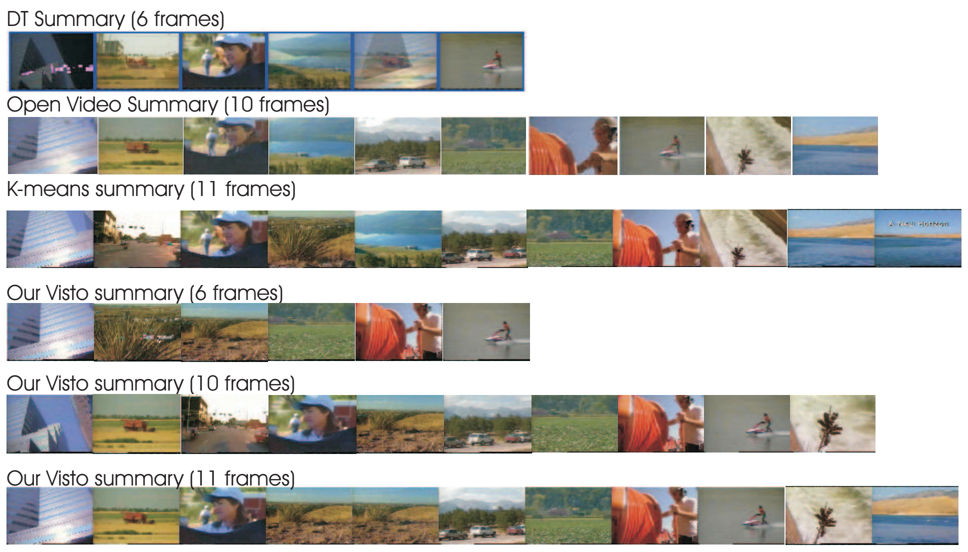

DT Summary (6 frames)
Open Video Summary (10 frames)
K-means summary (11 frames)
Our Visto summary (6 frames)
Our Visto summary (10 frames)
Our Visto summary (11 frames)

Figure 5.8. *A new Horizon*: storyboard comparison.

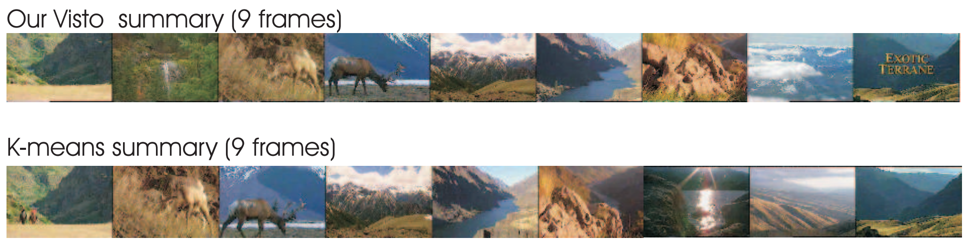


Figure 5.9. *Exotic Terrane*: k-means and ViSto comparison.

5.4 Clustering to produce dynamic video skimming

In the literature, among the few clustering techniques designed to produce video abstracts, some start by clustering frames and then recover the scenes from the selected frames; others, cluster scenes and then select one scene per cluster. Although the definition of a scene might vary from technique to technique, the final output is always produced by sequencing the selected video scenes.

In this section we analyze both approaches (i.e., frame-based and scene-based selection), where a scene is a segment of video that begins and ends with a silence and a video cut. Note that, when talking about the scene to which the frame belongs to, we mean the only scene in which the frame appears.

Frames are represented as explained in section 5.2.1 and stored in a matrix M_{HSV}, while each scene is represented by a 256-dimensional vector whose i-th entry is the mean of the corresponding entries of the frames belonging to the scene. Scenes are stored as rows of the matrix M_{avgHSV}.

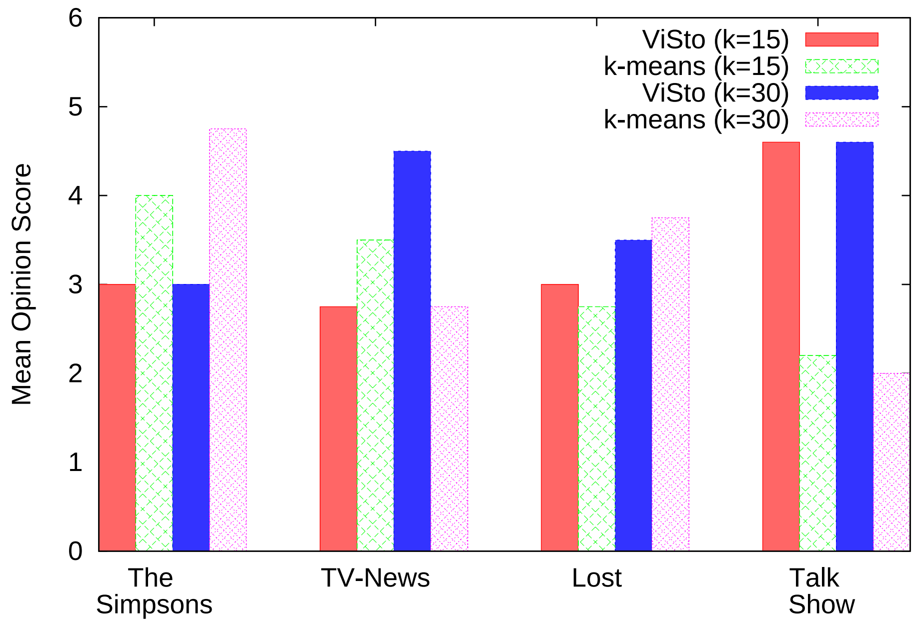


Figure 5.10. MOS evaluation of long videos.

5.4.1 Video segmentation

A video abstract is composed by a sequence of the most important segments of the original video and hence the abstract quality also depends on the video segmentation process. We observed that it is of crucial importance for the process to consider both audio and video features of the video to be summarized. In fact, if a video is divided according only to visual information, for instance by splitting the video where there is a video cut, (which happens when two consecutive video frames have few parts in common), it is likely that a video segment has an incomplete audio.

To split the video into segments we use the technique proposed in [Furini, 2007] that takes into account low-level audio and video features. When a video cut is detected, the audio energy at the video transition is checked: if there is silence, the transition is considered to be the end of a segment, otherwise it is assumed that the segment is not over. When combining segments obtained in this way, we get a fluid, understandable abstract in which the audio is completely intelligible and not interrupted.

5.4.2 Comparison between the use of scenes or frames

Considering the problem of static storyboard generation, in which the final output is a set of frames, it seems reasonable to consider frames as input of the clustering

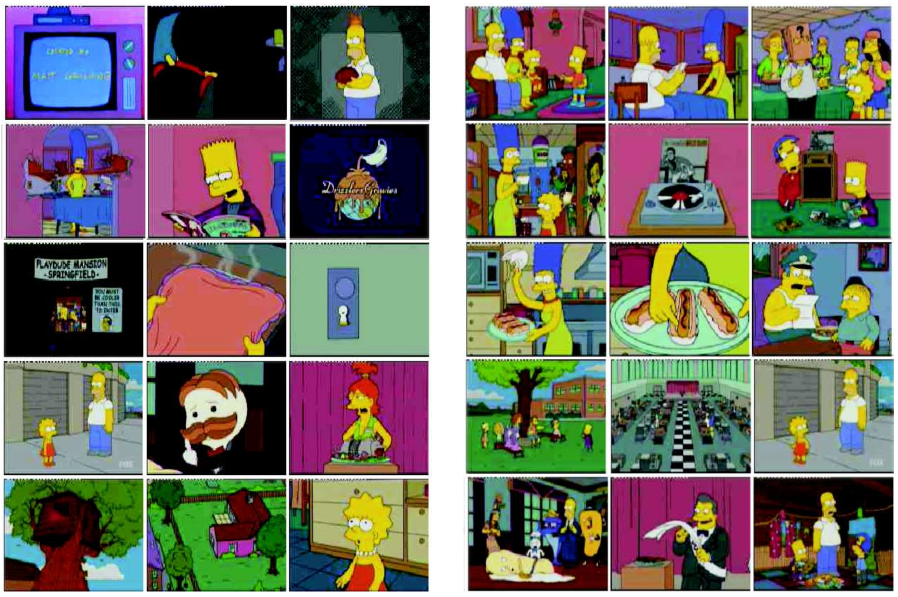


Figure 5.11. *The Simpsons*: ViSto vs k-means.

algorithm. In the video skimming framework, where a sequence of scenes will be returned, one should evaluate whether is more convenient an approach that uses frames for clustering and then from them derives video shots, or the alternative approach in which the video is immediately segmented and scenes are the input of the clustering algorithm.

For the sake of testing which approach performs better, given a video and a desired abstract length T, we produce two abstracts, one obtained by clustering vectors in M_{HSV} (corresponding to frames), and one by clustering vectors in M_{avgHSV} (corresponding to scenes).

We evaluate two clustering algorithms with different characteristics: one is the well-known k-means [Phillips, 2002], widely used and considered in literature, the other is the approximated version of the M-FPF-MD algorithm described in section 5.3.1.

As in the static case, both algorithms require as input the number of clusters k, and, for each cluster they return a representative element. To measure the frame (scene) similarity, we consider the Generalized Jaccard Distance [Charikar, 2002], since, as noted in section 5.3.1 this metric has shown to perform well for HSV vectors (see also [Furini *et al.*, 2007]).

We do not take into account other well known clustering algorithms (e.g., Hierarchical clustering) because these are computationally slower than k-means and, hence, they do not apply to our scenario of on-the-fly customized video abstract generation.

We produce abstracts in the following way:

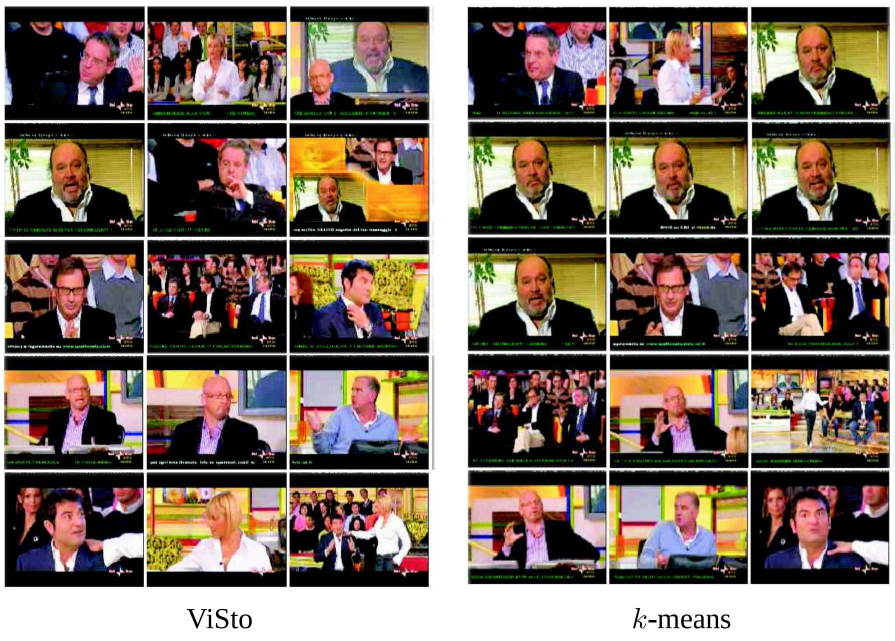


Figure 5.12. *Talk-show*: ViSto vs k-means.

- **Abstract by frames:** given T in seconds and fps, the frame per seconds of the original video, we compute the number of frames that should be in the abstract as $\#SF = T \cdot fps$. We estimate the number of scenes $\#SS$ in the abstract with a value such that the ratio between the number of frames and the number of scenes in the original video and in the abstract is the same. Chosen an arbitrary small integer constant c, we cluster vectors in M_{HSV} in $k = \#SS \cdot c$ clusters, obtaining k representative frames. For each frame we determine the scene to which it belongs to and we increment by one a counter associated to the scene (initially all counters are set to zero). Starting from the scene with higher counter, and considering scenes in decreasing order, we select the scenes to be in the abstract until the total length of the selected scenes reaches the time T. Observe that c is used to produce a number of clusters higher than the number of scenes that will compose the abstract, generating a significant ranking of the scenes by means of the counters.

- **Abstract by scenes:** we cluster vectors of the matrix M_{avgHSV}. The main problem, in this case, is to find the appropriate value for k such that the resulting output storyboard has the desired length. We were forced to use different approaches for ViSto and k-means. In the case of ViSto we exploited the fact that FPF generates a new permanent center of a new cluster at each iteration, giving a way to rank centers, *i.e.*, a selection order. Note that items that are clustered represents scenes and that, the order in which they are considered in the clustering process is completely independent from the order in which the scenes appear in the original video. Thus we simply overestimated k to be higher than T/SL where SL is the average length of a scene and then, in the same order in which clusters were created, scenes are selected and inserted in the abstract. The process continues until the total abstract length reaches the duration T. For the k-means algorithm we proceed with a brute force approach to determine the k clusters necessary to produce an abstract of length T (note that at the moment we are not discussing the best way to choose k). In both cases, the selected scenes are ordered according to the time in which they appear in the original video and the sequence that has been obtained is presented as the abstract.

Random abstract To evaluate if clustering might help in producing video abstracts, we also compute abstracts by choosing frames and scenes at random. If there is no significant difference between these abstracts and those produced using clustering, the only natural conclusion is that there is no reason in spending time and resources with clustering. To produce the randomized abstracts we proceed as follows:

- **Abstract by frames:** choose frames at random and select their corresponding scenes until the total length of the selected scenes reaches T.
- **Abstract by scenes:** randomly choose scenes until the total length of the selected scenes reaches T.

In both cases, we reorder the scenes according to the time in which they appear in the original video and we output the resulting abstract.

5.4.3 Evaluation

To evaluate the benefits of using clustering algorithms to produce video abstracts in the web scenario, we set up an experimental environment investigating the performance of clustering algorithms against a random approach. In order to have a wide test bed, we consider three different categories of videos: cartoons, TV-Shows and TV-News. Movies have not been considered since video abstracts reveal too much contents (e.g, the end of the movie), and hence ad-hoc techniques to produce *highlights* are more suited for this category. Table 5.1 reports a detailed description of the dataset.

Afterwards, a HSV color analysis produces, for each video, two different tables (representing M_{HSV} and M_{avgHSV}), which are the input for the clustering algorithms.

We produce two sets of video abstracts: one contains 2 minutes long abstracts and the other contains 4 minutes long abstracts. These lengths have been chosen to be reasonable for a video abstract. For each set, we compute two different video abstracts for each video: one is frame-based and the other is scene-based. This

Category	duration	avrg. scenes	title	# videos
TV-Shows	40 min	534	Charmed	1
			Roswell	1
			Dark Angel	1
			Lost	1
Cartoons	20 min	322	The Simpsons	2
			Futurama	2
TV-News	15 min	67	Sky TV	2
			BBC	2

Table 5.1. Dataset description.

means that, considering also randomly produced abstracts, for each given video we have 6 different video abstracts.

The goal of this experimental environment is to test the quality of the produced video abstracts and also the generation time in order to potentially offer a customized service.

5.4.3.1 Quality evaluation using a ground-truth

The set up of the evaluation of a video abstract is a difficult task: objective metrics like PSNR cannot be applied to videos of different lengths, hence, user evaluation has to be considered [Truong and Venkatesh, 2007]. However, since the presence of long videos may discourage a truthful evaluation, a more effective method is to compute a *ground-truth* abstract of each video (a manual built abstract containing the most important video scenes), and compare the produced abstracts with it. We proceed as follows:

(a) Given the original video, we manually split it into *Super-Scenes* (s-scene) each having a self contained meaning (*e.g.*, dialog between two characters in the kitchen; trip from here to there by car and so on). A s-scene might contain more than one scene (as defined in this chapter) or can be a fraction of a scene (e.g. two different actions taking place during one single background piece of music).

(b) We ask a set of 10 users (undergraduate and Ph.D. students, young researchers and non-academic) to score each s-scene with a value from zero to five (0 = un-influent, 5 = fundamental). Then, to each s-scene we associate a score, that is computed as the average of the scores given by the users.

(c) Given an abstract, each scene is scored with the score given to the s-scene it belongs to or with the sum of the scores given to the scenes it is composed of. The abstract receives a score that is equal to the sum of the scores of the scenes it is composed of.

It might be observed that also this evaluation approach needs the intervention of several users, as it was in the user study approach. On the other hand, in the ground-truth approach, once the process of scoring s-scene is done, experiments can be carried out and automatically evaluated.

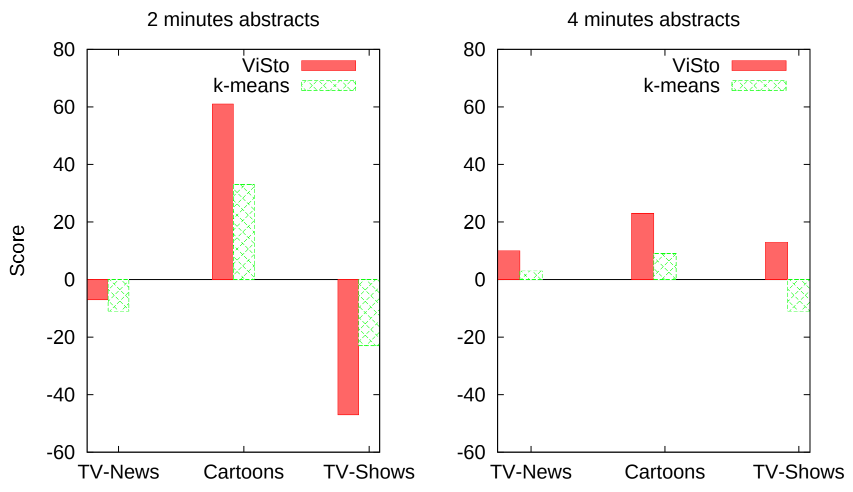


Figure 5.13. Ground Truth Evaluation: Comparison of abstracts produced using a frame analysis. Results are normalized with respect to the random abstract scores (positive values mean better results than random, negative worse).

Before presenting the details of the ground-truth evaluation, it is worth pointing out that the data produced by the set of users presented a large statistical difference in the scores related to TV-News, whereas more homogeneous scores have been given for cartoon and TV-Shows videos. This shows the importance of a story-line in the video: TV-News has multiple story-lines, each one presented in a different video clip and some users might prefer some story-lines to others (e.g., the same soccer video may be evaluated as very important by a soccer fan, whereas it can be meaningless for his wife). Conversely, when there is a single (or few) story-line, as in Cartoon/TV-Shows, evaluation of the video clips tends to be more oriented to the video story-line and less to the users interests.

Figure 5.13 reports the results of the ground-truth evaluation of abstracts produced with frame-based analysis. Results are normalized with respect to the quality achieved by the random approach (i.e., positive values mean better results than random, negative worse). Clustering techniques are worth using only for Cartoon videos; for TV-News there is no significant difference with the random approach; for TV-Shows videos, clustering are not worth using for 2 minute abstracts, whereas some benefits are present for 4 minute abstracts. The TV-News behavior is not surprising considering the large statistical difference of the ground-truth evaluation. Instead the bad results of 2 minutes long TV-Show abstracts are a little bit surprising. An explanation might be that the abstracts are too short compared to the

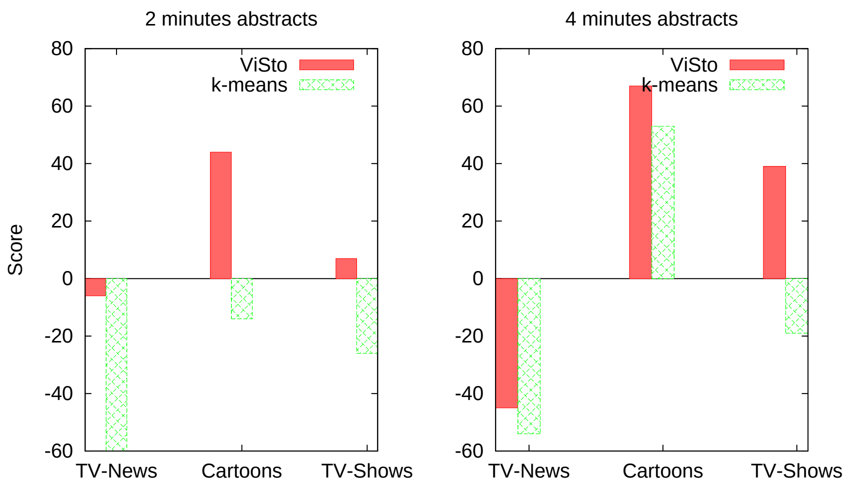


Figure 5.14. Ground Truth Evaluation: Comparison of abstracts produced using the scene analysis. Results are normalized with respect to the random abstract scores (positive values mean better results than random, negative worse).

original videos (2 minute abstracts against a 40 minute video) and hence it is difficult to pick up interesting scenes to fill in such a limited time abstract (for instance, *Roswell* abstracts perform very similar to the one randomly produced, whereas *Charmed* abstracts are much worse than the random ones). In fact, looking at the longer 4 minute abstracts, we can find better results for clustering.

Figure 5.14 reports the results of the ground-truth evaluation of abstracts produced with scene-based analysis. Also in this case, results are normalized with respect to the achieved random quality. Clustering techniques are not worth using for TV-News videos, whereas there are benefits for cartoon videos. For TV-Show videos there are no significant benefits for 2 minute abstracts, whereas clear benefits are present for 4 minutes long abstracts produced with the M-FPF-MD technique.

5.4.3.2 Generation time evaluation

In this section we analyze the abstract generation time, which is very important for video abstract length customization, as abstracts have to be produced on-the-fly to meet the user request. The following results are obtained using a simple Pentium D 3.4 GHz with 3GB RAM. Although more powerful hardware can be employed to lower the generation time, the ratio is likely to be the same.

Figure 5.15 reports results related to the abstract generation time (in seconds) with frame analysis, given on a logarithmic scale. Generation time of random abstract is not reported as it is less than one second, regardless of the type of video. Needless to say, the lower the generation time, the better for a customized service.

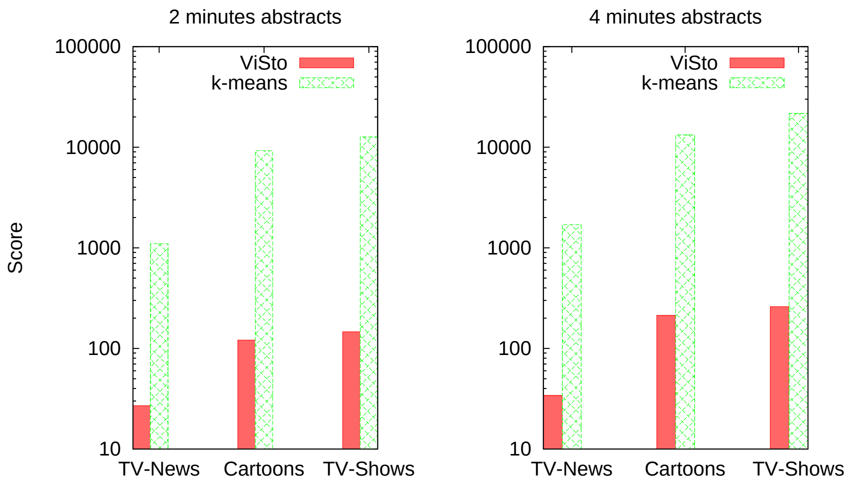


Figure 5.15. Generation Time: Comparison of abstracts produced using frame analysis. Results are presented on a logarithmic scale.

Observe that k-means is out of the game (note that we don't consider the time spent looking for a good value of k), as it takes too much time to produce a video abstract. M-FPF-MD has reasonable performances only for TV-News (27 seconds to produce an abstract for a 15 minutes video).

Figure 5.16 reports results related to the abstract generation time (in seconds) with scene analysis, given on a logarithmic scale. Again, the generation time of random abstracts is not reported as it is around 0.1 seconds, regardless of the type of the video. M-FPF-MD has always good performances (19 seconds to generate a 4 minute abstract of a 40 minute video and less than one second for TV-News videos), whereas, k-means has reasonable generation time only for TV-News videos.

5.4.3.3 Summary of results

Experimental results lead to the following conclusions:

- Clustering techniques seems not to be useful for multiple story-line videos like TV-News. If videos are based on a story-line, as cartoons and TV-Shows, the benefits of clustering are significant, especially for 4 minute abstracts based on video scene analysis.

- Random selection has to be preferred to clustering for 2 minutes long abstracts. In such videos, the limited number of scenes that can be selected to compose the abstract, compared to the total number of video scenes, does not leave much space for interesting choices.

- Generation of abstracts by frame analysis takes much longer than those by scene analysis (e.g., scene-based clustering is one order of magnitude more efficient than the frame-based one) and hence abstracting by frame analysis is not a winning strategy.

- k-means is too time consuming to be considered as a technique to produce on-the-fly abstracts.
- If user customization is enabled only a very fast clustering algorithm as the approximate M-FPF-MD can be used.

Another important issue is the estimation of the number of clusters k to get an abstract of the desired length. Since, it is not easy to associate k with scene lengths, clustering algorithms should be not forced to start over again if the choice of k results incorrect, thus incremental clustering represents a better strategy.

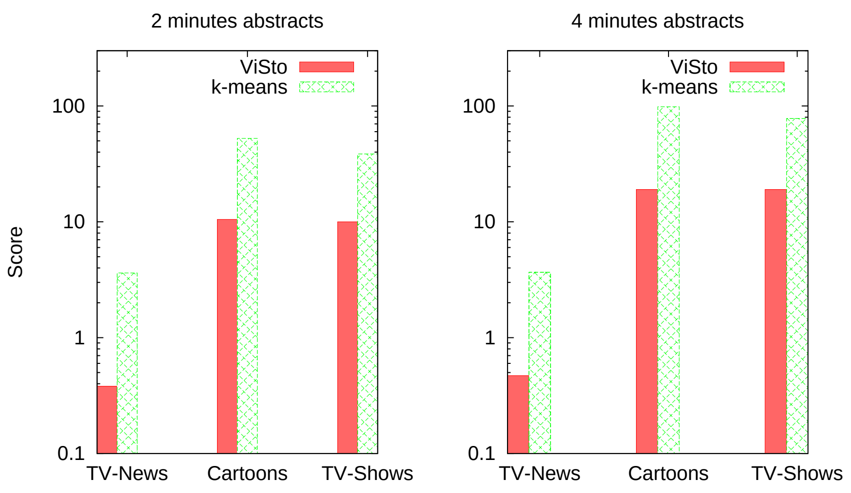


Figure 5.16. Generation Time: Comparison of abstracts produced using scene analysis. Results are presented on a logarithmic scale.

5.5 Conclusions

In this chapter we studied the problem of on-the-fly generation of video abstracts of long videos. In the case of static storyboard generation many clustering based approaches are proposed in the literature, but they are too slow to be applied to on-line contexts like the Web. For dynamic video skimming, clustering is typically not considered as a possible approach due to its computational cost. We proposed ViSto, a mechanism designed to produce customized on-the-fly video storyboards. ViSto is based on an approximated version of the M-FPF-MD algorithm. We tested ViSto against an accelerated version of the standard k-means and, only for static case, in which other methods were available, against DT summary and the approach used in Open Video. The results of the comparison show that ViSto is about 25 times faster than k-means and 300 times faster than DT.

A rigorous evaluation of the quality of storyboard is not present in the literature. Moreover it is a hard task also for humans. The same output can be considered differently among a group of people. For this reason we evaluated both static and dynamic storyboards by means of an extended user study. In the case of dynamic summarization we were also interested to analyzed the benefits of using clustering techniques on frames or directly on scenes to produce video abstracts. Results show that although in most cases ViSto achieved the best score, it can be said that the quality of the storyboard produced by all the considered algorithms is comparable and, in the dynamic case, the use of scenes is helpful in obtaining a better results.

Chapter
6

Clustering for similarity search

Abstract

Modern text retrieval systems often provide a similarity search utility*, that allows the user to find efficiently a fixed number k of documents in the data set that are the* most similar *to a given query (here a query is either a simple sequence of keywords or a full document). We consider the case of a textual database made up of semi-structured documents. For example, in a corpus of bibliographic records any record may be structured into three fields: title, authors and abstract, where each field is an unstructured free text. Each field, in turns, is modeled with a specific vector space. The problem is more complex when we also allow users to associate a score to each such vector space at query time, influencing its contribution to the overall dynamic aggregated and weighted similarity. We investigated a way of using clustering to prune the search space at query time. We observed that the more the query is close to a cluster center, the more the k-center objective function assure that the elements in the cluster are the nearest neighbors for the query. Also the embedding of the weights in the data structure was investigated with the purpose of allowing user query customizations without data replication like in [Singitham* et al.*, 2004]. The validity of our approach is demonstrated experimentally by showing significant performance improvements over the scheme proposed in [Singitham* et al.*, 2004] and also with respect to the method [Chierichetti* et al.*, 2007]. We also speed up the pre-processing time by a factor at least thirty.*

6.1 Introduction to similarity searching

Similarity Searching in an ubiquitous operation much studied in a variety of research communities including: database, spatial databases, data structure, computational geometry, information retrieval [Chávez *et al.*, 2001]. An alternative name is *nearest-neighbor searching*, while another old name is the *post office problem*.

The problem can be studied in a variety of settings: static or dynamic, offline/on-line, main memory/secondary memory, streaming data vs fully stored data, exact solution/approximate solution, 1-nearest-neighbor vs k-nearest-neighbor, worst case asymptotic guarantees vs heuristics.

As in all data structure problems if one assumes that the data is available for pre-processing the most relevant performance measurements are pre-processing time, storage and query time. In a dynamic setting also the time for dynamic operations is of interest. In an approximate case one is also interested in the quality of the approximation.

Another important discrimination is based on the nature of the data being searched (free textual data, structured records, images, sounds, etc..), on the nature of the representation of the items being searched (eg. dense vectors versus sparse vectors), on the intuitive notion of “similarity/distance” being investigated, and on the formal notion of “distance” being adopted.

Another very closely related problem is that of fixed-radius searching (a type of range-searching) in which one searches for all the elements in a data set at distance less than a certain parameter r from a query object.

Similarity searching is strongly connected to many other important problems:

- **Set-Membership**: one can see the set-membership problem as a limiting case. Given a finite subset S of a universe U, determine for an $x \in U$ if $x \in S$. Naturally one could see this problem as an exact similarity searching problem where the distance of two elements is given by $d(x, y) = 0$ for any $x \in S$, and $d(x, y) = 1$ for any $x \in U \setminus S$. Sometimes this is called the “exact match” problem.

- **Inverse Indexing**: Web Search Engines have made popular Inverse Indices to solve the following problem: Given a family $\mathcal{F} = \{F_1, ...F_k\}$ of subsets of U, for any $x \in U$ return all $F_i \in \mathcal{F}$ such that $x \in F_i$. This can be seen as a similarity search problem in which $d(x, F_i) = 0$ if $x \in F_i$, and 1 otherwise. Note that here we have two classes of objects, points and sets, rather than one. Hence a notion of distance between a set and a point must be defined.

- **Voronoi Diagrams**: given a universe U and a distance function $d : U \times U \rightarrow \mathbb{R}$ and a finite set $S \subset U$, let $\mathcal{D}(x, S) = \arg\min_{y \in 2^S} d(x, y)$, a function $U \rightarrow S$. Partition U into subsets with the same pre-image (that is, define the equivalence classes under the inverse mapping $\mathcal{D}^{-1}$). Note that a Voronoi diagram, in a sense, stores compactly the answers to all possible 1-nearest neighbor queries. It can be used by transforming the nearest-neighbor query into a point-location query in this special subdivision of U. Unfortunately in many high dimensional situations the Voronoi diagram itself is a too large object to be of any use when stored explicitly.

- **Dynamic Vector Score Aggregation**: given a set S of s sources of evidence and a set E of n records, let $\sigma_i(e_j)$ be a source score for each record $e_j \in E$ and each source $s_i \in S$. Moreover for each source s_i we have a scalar positive weight w_i that is user-defined and changes dynamically for each query. The dynamic aggregated score of e_j is $\sum_{i=1}^{s} w_i \sigma_i(e_j)$. The *Dynamic Vector Score Aggregation* problem [Singitham *et al.*, 2004] is to find the k elements in E with the highest dynamic aggregate score. In [Singitham *et al.*, 2004], the authors note that in absence of any further structure the only solution is an exhaustive computation of the aggregate score for all the elements in E and the determination of the k highest elements in the ranking induced by the aggregation score. Therefore they consider the special case when each feature of the records e_j is actually a vector, and the source score function $\sigma_i(e_j)$ is a geometric distance function measuring the distance of e_j to a query point q (equivalently one can define a dual similarity function to the same purpose). Observe that if $s = 1$ and the source score is a geometric proximity function (e.g. a metric) to a query point then this problem reduces to the classical k-nearest neighbor problem. The difficulty in handling the k-nearest-neighbor problem in the general case of a linear combination of $s \geq 2$ geometric proximity functions stems from the need of combining the scores from generally unrelated sources compounded with the presence of arbitrary positive weights. In [Singitham *et al.*, 2004] the Vector Score Aggregation problem is solved by extending the cluster pruning technique for the geometric k-nearest-neighbor.

In this chapter we observe a relationship between the approximate similarity searching problem and the k-center problem. Thus we show how the cluster pruning strategy for approximate similarity searching can be improved using a clustering algorithm which has k-center as objective function. Moreover, since we observed that the most of the nearest neighbors are found visiting the first few clusters we modified the cluster pruning approach to work with a multi-clustering obtained by merging few independent clusterings.

To finish, using cosine similarity to measure the distance among pairs of documents, we derived a different scheme for the dynamic vector score aggregation problem respect with the one proposed in [Singitham *et al.*, 2004]. Our method is able to deal with weights just at query time, thus we do not need to know/manage them during the preprocessing of data. This reduced our preprocessing time and, contrarily to the method proposed in [Singitham *et al.*, 2004], made it independent from the number of sources of evidence present in the dataset.

6.2 Related work on similarity searching

A lot of effort has been spent studying similarity searching and literature about this topic is wide. There are two main categories of similarity searching problems: *exact similarity searching* in which the i-th retrieved element is exactly the i-th nearest neighbor; and *approximate similarity searching* in which the following relationship holds:

$$\frac{d(q, Retrived_i)}{d(q, NN_i)} < 1 + \epsilon$$

where q is the query point, $Retrived_i$ is the i-th retrieved element, NN_i is the i-th nearest neighbor and ϵ is a small positive constant.

6.2.1 Exact similarity searching

The most of exact similarity searching data structures are based on partitioning the search space and organizing the obtained partitions in a tree-like data structure. At query time the tree structure is used to decide in which partitions to search for the nearest neighbors. Among the most commonly appreciated partitioning methods, there are: the *ball partitioning* and *generalized hyperplane partitioning* criteria introduced in [Uhlmann, 1991] and the *excluded middle partitioning* described in [Yianilos, 1999]. In figures 6.1, 6.2 and 6.3 the three methods are summarized.

In a nutshell given a metric space $M = (U, d)$ and a set of objects $S \in U$:

- **ball partitioning**: given a point $p \in U$ and a constant d_m, S is divided in two groups S_1 and S_2 such that the inner points of the ball of center p and radius d_m are in S_1 while the remaining are in S_2,

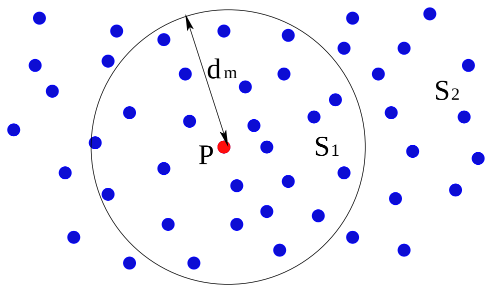


Figure 6.1. Ball partitioning.

- **hyperplane partitioning**: given two points $p_1 \in U$ and $p_2 \in U$, S is splitted in two sets S_1 and S_2 such that points in S_1 are closer to p_1 than to p_2 and vice versa,

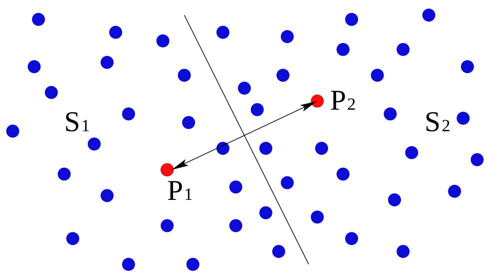


Figure 6.2. Hyperplane partitioning.

- **excluded middle partitioning**: given a point $p \in U$, a threshold distance d_m, and a constant δ, the set S is divided in three subspaces S_1, S_2 and S_3. The points at distance at most $d_m - \delta$ from p are assigned to S_1, the points furthest than $d_m + \delta$ from p are assigned to S_2 and the unexpended points are inputed to S_3. The main advantage of this partitioning scheme with respect to the others is that, also in the case in which the requested query point q is too close to such a boundary, if the searched points stand in the a ball with center q and radius δ it is always possible to exclude at least one partition in the visiting procedure.

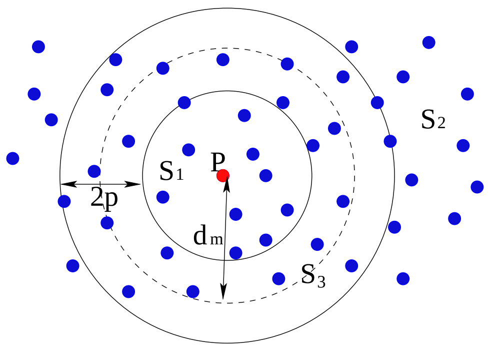


Figure 6.3. Excluded middle partitioning.

Let now describe some common methods for exact similarity searching that employ the above described partitioning techniques for organizing data.

The most simple method consists in building a binary tree structure using ball partitioning for assigning points to one of the halves of the tree. At the beginning all the points are assigned to the root, a pivot point p is randomly chosen and the space is partitioned. The root node is labeled with p. Each partition is again splitted using the same criterion until a stop condition is reached. Queries consist in a top-down visit of the tree; visiting node t, the distance between the query and the pivot p_t is computed and the algorithm decides which partition (could be both) might contain similar data. The querying process continues until, for each visited path, a leaf node is reached.

The bisector tree (BST) [Kalantari and McDonald, 1983] is probably the first data structure proposed that takes advantage from the hyperplane partitioning. Also this tree structure is built recursively. At each node two pivots points p_1 and p_2 are selected and hyperplane partitioning is applied. Objects are assigned to the two children of the considered node according with their distance to p_1 and p_2. For each pivot its covering radius is computed and stored in the appropriate child node. The covering radius is the distance between the pivot and the furthest point in its partition. Also in this case the query consists in a top-down visit of the tree. Considering a node t having p_t as pivot and r_t as covering radius and a query point q, in a metric space points in t are further than $d(p_t, q) - r_t$ from q. Thus, if searched points are closer, then there are no interesting points in that subtree and it can be safely pruned.

Brin [Brin, 1995] tackles the problem of detecting the points of a data set in a metric space within a radius r of a query point x. Both r and x are specified at query time. The proposed data structure the GNAT (Geometric Near-Neighbor Access Tree) uses only pairwise distance computations and builds a deep search tree by recursive subdivision clustering at each node of the tree. At each node random sampling and a greedy selection of candidates split points using the furthest-point-first idea is performed. The GNAT is used to find exactly all points in a radius r from a query point x and each node of the data structure that potentially intersects the query ball is visited. Potentially one could use GNAT and a binary search scheme on the radius value r so to detect the ball r_l containing exactly l elements closest to the query point x. In practice such a scheme could result in a visit of a much larger fraction of the points in the database than needed. Variants are discussed in [Chávez and Navarro, 2000; Cantone *et al.*, 2005; Figueroa *et al.*, 2006].

6.2.2 Approximate similarity searching

Fagin et al. in [Fagin *et al.*, 2003] show that one can solve approximate Euclidean nearest neighbor by first projecting data points and query points onto a set of 1-dimensional flats (lines), compute the rank of the query in each 1-dimensional space, and then combine (aggregate) these ranks using deep techniques from *voting theory*. Experiments reported in [Fagin *et al.*, 2003] are on dense data sets in dimension 100 (stock market time series) and dimension 784 (vectorialization of digital images).

Cluster pruning is an approach to similarity searching that is rather simple but, just because of its simplicity, is suitable to handling very large data sets in very high dimensional spaces (as arise for example in handling large corpora of free textual information). Consider the data items as points in a high dimensional space endowed with a distance function. Subdivide the data points into many small compact clusters and elect a representative point in each cluster. When a query point q

is given, q is compared with the representatives and based on this comparison one decides either to explore further the cluster or to disregard completely the associated cluster. The heuristic step (no guarantee) is that the selected clusters contain the exact (or approximate) answer we are looking for. There are many algorithmic design choices one has to take (see a recent paper [Chierichetti *et al.*, 2007] exploring many of these choices). The approach itself in general has been used before, see e.g. [Hafner *et al.*, 1998; Sitarama *et al.*, 2003].

Indyk and Motwani propose the *locality sensitive hashing* [Indyk and Motwani, 1998], that reduces the approximate similarity searching problem to the *Point location in equal balls* problem (PLEB). Given a set $C = \{c_1, \ldots, c_n\}$ of balls of a fixed radius r in a metric space and a query point q, if there exist a ball such that $q \in B(c_i, r)$, then return c_i. Thus the authors introduce a family $\mathcal{H} : S \to U$ of hashing functions such that if $q \in B(c_i, r)$ then $Pr_{\mathcal{H}}[h(q) = h(c_i)] \geq p_1$ while if $q \notin B(c_i, r)$ then $Pr_{\mathcal{H}}[h(q) = h(c_i)] \leq p_2$ with $p_1 > p_2$. Essentially the main property of LSH is that similar objects hashed with $\mathcal{H}$ are much more likely to collide than dissimilar objects. At query time the query q is hashed and only the points in the ball B_i such that $h(q) = h(c_i)$ are scanned to find the most similar to q.

In [Bawa *et al.*, 2005] a LSH-based index is presented. Given a family $\mathcal{H} = \{h_1, \ldots h_n\}$ of LSH functions, l hash tables are built independently. For each table, k hash functions are selected at random. Then each document is hashed using all these functions and placed in the bucket which has as key the concatenation of the hash functions. For example if we consider $\mathcal{H} : S \to [0, 9]$ in the hash table there will be 10^k buckets. The smaller is k, the greater is the probability of conflicts among dissimilar points. But, a large value for k means a large number of buckets and an increase of probability that similar objects are assigned to small buckets. At query time nearest neighbors are searched separately in all the l hash tables and then recombined. Note that one of the major advantages of this data structure is that it can be used in parallel or distributed frameworks.

A series of papers describe solutions tailored on the scenario in which data are accessed as a stream, that is they are presented to the system in a sequence and assigned to a cluster once and for all. One can work in a scenario in which there is loss or no loss of information, see [Guha *et al.*, 2003; Charikar *et al.*, 1997; Farnstrom *et al.*, 2000]. Clearly this is a more extreme scenario in which, besides the simple scalability issue (due to sheer size of the input), one has to cope also with the restriction of deciding the association of each data item only once. In the applications over large textual corpora coping with this last requirement it is likely to produce lower quality results. The experiments reported in [Farnstrom *et al.*, 2000], that exhibit no loss in quality against standard k-means, are performed on a data set in 481 dimensions, where each record is dense (i.e. most components are non zero).

6.2.2.1 User defined aggregated metrics

In the standard version of the similarity searching problem the user is allowed to choose the queries, but not the underlying distance function that is fixed at preprocessing time. Suppose now that we want to give the user the possibility of choosing a metric of his/her own choice at query time. One special case of this scenario is for example when the objects to be searched have an internal structure and the overall distance function is an *aggregation* of the outcome of several distance functions defined on the components of the structure. Even if the data set to be searched is the same, at different times the user might prefer a different relative weight of the individual factors within the overall aggregated distance function. This choice is taken at query time and the data structure must be flexible in this respect. This scenario has been considered in a recent paper by Singitham et al. [Singitham *et al.*, 2004]. In [Ciaccia and Patella, 2002], Ciaccia and Patella discuss which general relations should hold between two metrics that allow to build a data structure using the first metric, but perform searches using the second (e.g. a user defined) one. They propose a method that can be applied to any generic distance based search tree. The performance analysis is based on probabilistic distance distributions.

6.2.2.2 Cluster based search versus inverted index based search

Voorhees [Voorhees, 1986] discusses the problem of comparing the performance of similarity search algorithms based on inverted indices with those based on clustering. The issue has been re-evaluated by Can et al. [Can *et al.*, 2004]. For sake of simplicity assume that the similarity of two vectors is given by their inner product and that both documents and queries have been re-cast in a vector space. Inverted Index Similarity Searching is based on the principle that taking the non-zero components of the query vector we can access all and only the documents having also a non-zero value for at least one of those components; thus we can compute all inner products in a component by component order, and select at the end the document most similar to the query. This approach is suitable for exact nearest neighbor queries, however it runs into difficulties for approximate queries.

6.2.2.3 Measuring output quality

Given a certain database of objects and a query, in the exact similarity searching context, all the algorithms return exactly the same result. Thus what make the difference among different solutions are: preprocessing time and query time. In the approximate similarity searching setting, instead, it is also important to establish how good is the returned solution. In this last case, beyond preprocessing and query time, many other aspects should be considered. For example: how many (and which ones) of the true similar objects appear among the retrieved objects and how far are the retrieved points from the query. For example, looking for the two most similar objects, it is better to find the closest object rather than the second one. Moreover it could be better to find the second and the third most similar objects instead of the closest object and a far one.

Two popular quality indexes are often used: the *mean competitive recall* and the *mean normalized aggregate goodness*. Since they are employed also in [Singitham *et al.*, 2004] and [Chierichetti *et al.*, 2007], we will use them for our experiments.

- **Mean Competitive Recall.** Let k be the number of similar documents, we want to retrieve (in our experiments $k = 10$) and $A(k, q, E)$ the set of the k retrieved documents for a query q by algorithm A on data set E, and the *Ground Truth* $GT(k, q, E)$, the set of the k closest points in E to the query q which is found through an exhaustive search; the competitive recall is $CR(A, q, k) = |A(k, q, E) \cap GT(k, q, E)|$. Note that competitive recall is an integer number in the range $[0, \ldots, k]$ and a higher value indicates higher quality. The Mean Competitive Recall $\bar{CR}$ is the average of the competitive recall over a set of queries Q:

$$\overline{CR}(A, Q, E) = \frac{1}{|Q|} \sum_{q \in Q} CR(A, q, k)$$

  This measure tells us how many of the true k nearest neighbors an algorithm was able to find.

- **Mean Normalized Aggregate Goodness.** We define as the Farthest Set $FS(k, q, E)$ the set of k points in E farthest from q. Let the sum of distances of the k furthest points from q be $W(k, q, E) = \sum_{p \in FS(k,q,E)} d(q, p)$. The normalized aggregate goodness:

$$NAG(k, q, A) = \frac{W(k, q, E) - \sum_{p \in A(k,q,E)} d(q, p)}{W(k, q, E) - \sum_{p \in GT(k,q,E)} d(q, p)}.$$

  Note that the Normalized Aggregate goodness is a real number in the range $[0, 1]$ and a higher value indicates higher quality. The Mean Normalized Aggregate Goodness $\overline{NAG}$ is the average of the normalized aggregate goodness over a set of queries Q:

$$\overline{NAG}(A, Q, E) = \frac{1}{|Q|} \sum_{q \in Q} NAG(A, q, k).$$

  Among the possible distance functions there is a large variability in behavior (for example some distance functions are bounded, some are not). Moreover for a given E and q there could be very different ranges of possible distance values. To filter out all these distortion effects we normalize the outcome of the algorithm against the ground truth by considering the shift against the k worst possible results. This normalization allows us a finer appreciation of the different algorithms by factoring out distance idiosyncratic or border effects.

6.3 Our contribution

In this chapter we first deal with the problem of fast approximate similarity searching for semi-structured text documents, then we focus on the more general problem in which users can decide to dynamically assign different weights to each field of the searched text (Dynamic Vector Score Aggregation). We observed an analogy between the similarity searching problem and the k-center objective function for clustering. Thus, according with the cluster pruning approach, we used a clustering algorithm for the k-center problem for approximate similarity searching. Moreover we derived a new and much simpler way for managing dynamic weights, avoiding to consider them in the preprocessing phase and thus to replicate clustering with many assignments of weights. To finish, by using a different clustering strategy (in which we introduced a certain data redundancy) we obtain further benefits in terms of precision. In particular we will describe alternatives for the following key aspects:

- **The ground clustering algorithm**. When searching for the nearest neighbors of a query point q, it is natural to consider a cluster good for such a search when its diameter is small. This leads to considering the optimal k-center problem (i.e. finding a decomposition minimizing the maximum diameter of any cluster produced) as a better objective to attain with respect to other conceivable objectives. Thus we are led to consider the Furthest-Point-First heuristic, that is 2-competitive for this problem [Gonzalez, 1985]. We attain two benefits: (1) the quality of the output is increased, as demonstrated by the experiments in Section 6.4, (2) the preprocessing time is reduced by orders of magnitude since we can use fast variants of this algorithm (see e.g. [Geraci *et al.*, 2006b; 2006a]).

- **Multiple clusterings**. In cluster pruning search one decides beforehand to visit a certain number of clusters whose “leaders” are closest to the query point. However, there is a hidden law of diminishing returns: clusters further away from the query are less likely to contain good k-neighbors. We use a different strategy: we form not one but several (three in our experiments) different independent clusterings and we search all three of them but looking into fewer clusters in each clustering.

- **How weights are embedded into the scheme**. In the general Vector Score Aggregation problem the user supplies a query (this can be either a document in the database or a collection of keywords that capture the concept being searched for) and a weight-vector that expresses the user’s perception of the relative importance of the document features in capturing the informal notion of “similarity”. We show in Section 6.3.3 that, surprisingly, one needs not to be concerned with dynamic weights at all during pre-processing; the solution for the unweighted case is good also for the weighted one.

By introducing these three variations we significantly outperform the state of the art algorithms for this problem.

6.3.1 Clustering algorithm

Cluster pruning leaves open the issue of which is the best clustering strategy to adopt. An exhaustive answer to this question is probably impossible since cluster pruning is heuristic. However, some considerations can be done. We found that there is a relationship between similarity searching and the optimal k-center problem. The k-center target is to find an assignment of cluster centers that minimizes the wider cluster radius. Thus the found clusters should be compact. If the query point q is exactly the center of cluster C_i, its elements are the first $|C_i|$ nearest neighbors of q. This also means that the more a query point q is close to the cluster center, the more it is probable that its nearest neighbors are points of the cluster. More specifically, if the cluster radius is D and the distance $d(q, c_i) < D$, all the nearest neighbors at distance at most $D - d(q, c_i)$ are points of c_i. Since k-center imposes that the wider cluster diameter is the smallest possible, in the case queries are points of the clustering, this means that also the maximal distance between the query and the closest center is minimized.

In chapter 2 three clustering algorithms for k-center problem were described: the standard FPF introduced in section 2.2.1.1 and two variants M-FPF that emploies random sample techniques (see section 2.4.2) and M-FPF-MD which introduces medoids (see section 2.4.3).

The choice of which of these clustering algorithms is the best candidate for similarity searching can not be supported by theory. Each of these algorithms has pros and cons. For example the standard FPF has theoretical guarantees, instead the use of medoids has a better distribution of centers. For this reason we simply tested the performance of all the algorithms on two databases of documents and adopted the best one. In section 6.4.3 we report the results of our tests which show that consistently M-FPF has the best performance.

6.3.2 Multiple clusterings

Apparently it seems that it would suffice to run a k-center algorithm on the input data (instead of k-means like in [Singitham *et al.*, 2004]) and the trick is done. The situation is slightly more complex. Generally speaking k-means is much slower than k-center, but typically produces higher quality clusters (the iterative process has a smoothing effect).

In a different setting, when strictly on-the-fly computations are required, such as those described in chapter 4 where interactive web snippets clustering is done or in chapter 5 where a video storyboard must be produced on demand, time constraints can make the difference between a successful tool and a boring useless software. In our setting, however, clustering is a pre-computation done off-line and, while one would appreciate a result in hours rather than days, clearly, quality of the outcome has to be considered in absolute terms.

We have found that, while one application of M-FPF is not able to outperform k-means in quality, a few independent applications of randomized M-FPF (*multi-clustering*) produce a set of overlapping clusters that yield better output quality. This intuition, later shown experimentally, came from two observations: after visiting t clusters the exam of a further one tends to be even more useless, redundancy of points makes the system more robust.

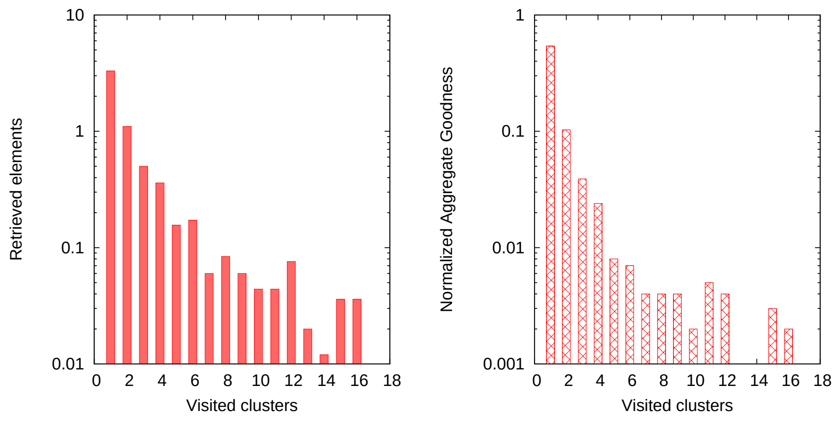


Figure 6.4. The *marginal utility* of examining the cluster $t+1$ after the inspection of t clusters. On the x axis the number of visited clusters, on the y axis the *marginal utility* in terms of recall (left) and normalized aggregate Goodness (right).

We studied how helpful is the visit of one more cluster after examining a certain number of clusters. In figure 6.4 we show how, after the visit of t clusters, the *marginal utility* changes after examining a new cluster. The *marginal utility* is computed for both Normalized Aggregated Goodness and Recall and it is expressed as the difference between the value of the measure visiting t and $t+1$ clusters.

Figure 6.4 shows that the visit of the first cluster is the most important and the *marginal utility* of examining a new cluster decays drastically after just the visit of three or four clusters. This suggests that is probably better to examine less clusters in some independent clusterings, than more clusters of the same clustering.

The second observation that suggested us to use multi-clustering was that the redundancy of points makes the system more stable in terms of query performance. This effect can be better explained with the example of figure 6.5.

In the figure the red circles and the green circles represent two independent clusterings (the center of a cluster is the point of the same color). Let q be the query and the yellow points the nearest neighbors. Suppose we want to visit two clusters searching for the nearest neighbors. If one considers only the red clustering q is close to R_1 and R_2, thus all nearest neighbors are found. Instead considering

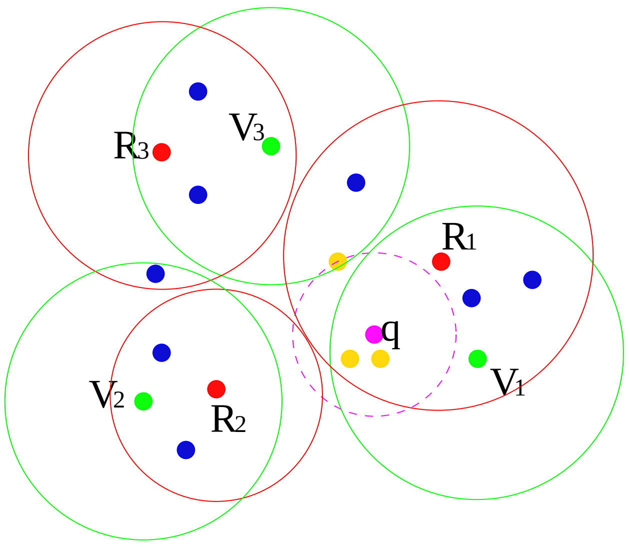


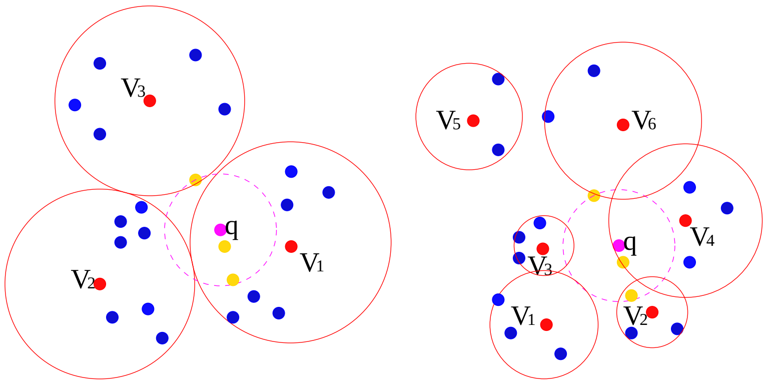
Figure 6.5. Two independent clusterings (one in red and one in green) of the same set of points. In magenta a query point and in yellow its nearest neighbors.

the green clustering, q is close to V_1 and V_2, but one of the searched points is in V_3. If one considers all the red and green clusters as a single clustering, then R_1 and V_1 are the closest centers and all the searched points are found. It is easy to note that in the last case the clusters with highest *marginal utility* are selected increasing the final expected quality.

The figure also shows that the mean distance among the query point and the nearest centers in the multi-clustering scheme must be equal or smaller than in the single clustering scheme.

At this point one can erroneously think that the advantage is in the higher number of centers and not in multi-clustering. Moreover multi-clustering has the disadvantage that there could be points evaluated more then once. Thus it could be better to make more smaller clusters and increase the number of visited clusters balancing the final cost. The example of figure 6.6 shows on the left a set of points divided in three clusters (CL1) and on the right the same set splitted in six clusters (CL2). Given the query q: we examine 2 clusters on CL1 and 4 on CL2. Note that only the cost of finding the nearest centers increases searching the structure on the right. In fact, since the visited clusters in CL2 are twice those visited in CL1, this balances the lower expected number of elements in each visited cluster. In both cases there is a nearest neighbor not found and the higher number of clusters on the right does not help.

In the comparison of k-means and multi-clustering there is an extra cost in terms of the number of distance computations to be paid at query time when searching the latter. However, since each distance computation in k-center involves only



(a) CL1 (b) CL2

Figure 6.6. The same set of points divided in 3 clusters (left) and 6 (right). In magenta the query point q and in yellow the 3 nearest neighbors.

sparse vectors (i.e. there are no dense centroids), the two effects balance out at query time. In our experiments three applications of k-center to different random samples of the input suffice.

There are two main ways to query multi-clustering. Suppose one wants to examine t different clusters at query time, the system can consider the three k-clusterings as a single clustering structure with $3k$ clusters (and centers) and examine t clusters, or it can query independently the three clusterings visiting $t/3$ clusters for each structure. We observed that this choice does not affect the final result neither in terms of quality nor in terms of speed. Thus we decided to use the latter strategy.

6.3.3 How weights are embedded in the scheme

In the vector score model the queries are of the form $q = (q_1, \dots, q_s)$ where each q_i is a vector of unit length; moreover the user supplies a weight vector $w = (w_1, \dots, w_s)$ where each w_i is a positive scalar weight, and the weights sum to 1. The element e_j in the input set E is of the form $((e_j)_1, \dots, (e_j)_s)$ where each $(e_j)_i$ is a vector of unit length. The aggregate similarity is: $s_{AD}(q, e_j) = 1 - d_{AD}(q, e_j) = \sum_i w_i(q_i \cdot (e_j)_i)$ where $q_i \cdot (e_j)_i$ is the cosine similarity between q_i and $(e_j)_i$. We remind that, as shown in section 3.2.1.1, from cosine similarity can be derived a distance D for which the extended triangular inequality holds. Thus, the aggregate distance function is $d_{AD}(q, e_j) = 1 - \sum_i w_i(q_i \cdot (e_j)_i)$.

One should notice that because of the linearity of the summation and the inner product operators, the weights can be associated to the vector space: $\sum_i w_i(q_i \cdot (e_j)_i) = \sum_i q_i \cdot w_i(e_j)_i = q \cdot we_j$. This association has been chosen in [Singitham *et al.*, 2004]. Thus the challenge arises from the fact that one has to do pre-processing without knowing the real weights that are supplied on-line at query time.

A different aggregation Let q be a query point, c a center of cluster $C(c)$, p a point in cluster $C(c)$, and d a distance function that satisfies the extended triangular inequality with parameter α. The effectiveness of clustering search stems from the observation that the distance $d(q,p)$ is bounded by an increasing function of $d(q,c)$ and $d(c,p)$. Moreover when $p \in C(c)$, the distance $d(c,p)$ has the smallest value over all centers in the clustering. Thus using the center c closest to q gives us the best possible upper estimate of the distance $d(q,p)$. We have:

$$d(q,p) \leq (d(q,c)^{\alpha} + d(c,p)^{\alpha})^{1/\alpha}$$

Consider now the *weighted similarity* WS:

$$WS(w,q,p) = \sum_i w_i(p_i \cdot q_i) = \sum_i (w_i q_i) \cdot p_i = Q_w \cdot p.$$

where $Q_w = [w_1 q_1, .., w_s q_s]$ is the weighted query vector of vectors. Since the linear combination of weights and queries might not result in a unit length vector we perform a normalization (depending only in the weights and query point) and obtain a *normalized weighted distance* NWD:

$$NWD(w,q,p) = 1 - \frac{WS(w,q,p)}{|Q_w|} = 1 - Q_w/|Q_w| \cdot p = d(Q'_w, p),$$

where $Q_w/|Q_w| = Q'_w$ is the normalized weighted query vector of vectors. Now we are in the condition of using the above generalized triangular inequality and establish that:

$$NWD(w,q,p) = d(Q'_w, p) \leq (d(Q'_w, c)^{\alpha} + d(c,p)^{\alpha}))^{1/\alpha}.$$

Since $d(c,p)$ is independent of the pair q, w we can do at preprocessing time a clustering based on the input set E and the distance d, regardless of weights and queries. At query time we can compute $d(Q'_w, c)$ and combine this value with $d(c,p)$ to get the upper estimate of $NWD(w,q,p)$ that guides the searching. The conclusion of this discussion is that using cosine similarity the multi-dimensional weighted case can be reduced to a mono-dimensional (i.e. not weighted case) for which we have good data structures.

The discussion above shows that the pre-processing can be done independently of the user provided weights and that any distance based clustering scheme can be used in principle. Weights are used to modify directly the input query point and are relevant only for the query procedure.

6.4 Experiments

In this section we will first show benefits obtained using the multi-clustering strategy with respect to standard clustering, then we compare our solution against two baselines: the algorithm in [Singitham *et al.*, 2004] that uses k-means clustering and the algorithm in [Chierichetti *et al.*, 2007] modified such as to use the simple cluster pruning randomized strategy proposed in [Singitham *et al.*, 2004].

6.4.1 Baseline algorithms

Let now give just some details of the two algorithms we used as baseline to evaluate the quality of our solution.

In [Singitham *et al.*, 2004] several schemes and variants are compared but experiments show that the best performance is consistently attained by the Query Algorithm 3 (*CellDec*) described in [Singitham *et al.*, 2004, Section 5.4]. The preprocessing is as follows. For simplicity we consider the 3-dimensional case, that is a data set where each record has 3 distinct sources of evidence (e.g. in our tests, title, authors and abstract of a paper). We consider the set T of positive weight-vectors summing to one (this is the intersection of the hyperplane $w_1+w_2+w_3=1$ with the positive coordinate octant). We split T into 4 regular triangles T_1, T_2 and T_3 each incident to a vertex of T and the central region T_4.

For each region we build a different vector space. Let $V_{i,j}$ be the vector corresponding to record e_j and source s_i. Since the value of weights in region T_4 (the central one) are comparable, we form a composite vector as follows $V(T_4)_j = V_{1,j} + V_{2,j} + V_{3,j}$. For the other three regions we simply apply a squeeze factor θ in correspondence of the two lowest weights. Thus we have $V(T_1)_j = V_{1_j} + \theta V_{2,j} + \theta V_{3,j}$, $V(T_2)_j = \theta V_{1_j} + V_{2,j} + \theta V_{3,j}$ and $V(T_3)_j = \theta V_{1_j} + \theta V_{2,j} + V_{3,j}$.

Experiments in [Singitham *et al.*, 2004] show that a value of $\theta = 0.5$ attains the best results. At query time, given the query $Q = (q, w)$ one first detects the region of T containing w, then uses q in the associated indexing data structure for cluster-pruning.

In [Chierichetti *et al.*, 2007] Chierichetti et al. propose a very simple but effective scheme for doing approximate k-nearest neighbor search for documents. In a nutshell, after mapping n documents into a vector space they choose randomly $K = \sqrt{n}$ such documents as representatives, and associate each other document to its closest representative. Afterwards, for each group the *centroid* is computed as “leader” of the group to be used during the search. In [Chierichetti *et al.*, 2007] the authors are able to prove probabilistic bounds on the size of each group which is an important parameter that directly influences the time complexity of the cluster prune search. Dynamically weighted queries are not treated in [Chierichetti *et al.*, 2007], therefore we choose as a second base-line to employ [Chierichetti *et al.*, 2007], in place of k-means, within the weighting framework of [Singitham *et al.*, 2004]. We will refer to it as *PODS07* for lack of a better name.

6.4.2 Experimental setup

We implemented all the algorithms in Python. Data were stored in textual *bsd* databases. Tests have been run on a Intel(R) Pentium(R) D CPU 3.2GHz with 3GB of RAM and with operating System Linux.

Following [Singitham *et al.*, 2004] we have downloaded the first one hundred thousands Citeseer bibliographic records[1]. Each record contains three fields: paper title, authors and abstract. We built the two data sets described in table 6.1. In the table, it is also reported the number k of clusters made by all the considered algorithms. After applying standard stemming and stop words removal, three vector spaces were created: one for each field of the documents. Terms in the vector are weighted according to the standard *tf-idf* scheme.

Dataset	TS1	TS2
Input size (MB)	41.80	76.13
# Records	53722	100000
# Clusters	500	1000

Table 6.1. Dataset description.

Without loss of generality, we used documents extracted from the data set as queries. Test queries have been selected by picking a random set of 250 documents. During searches the exact match of the query document is not counted. In our experiments we used the 7 sets of weights reported in table 6.2 adopted also in [Singitham *et al.*, 2004]. For each set of weights, we always used the same query set. This gave us the opportunity of comparing results for different choices of the weight vector.

Weight clue	Author	Title	Abstract
1	0.33	0.33	0.33
2	0.4	0.4	0.2
3	0.4	0.2	0.4
4	0.2	0.4	0.4
5	0.6	0.2	0.2
6	0.2	0.6	0.2
7	0.2	0.2	0.6

Table 6.2. Weights used for queries.

6.4.3 Comparison among different k-center algorithms

As explained in section 6.3.1 our intuition suggested us to use a clustering algorithm that attempts to solve the k-center problem since there is an affinity with the similarity searching problem. We faced the problem to establish which one of the algorithms described in chapter 2 for the k-center problem is the best candidate for similarity searching: the standard FPF (see section 2.2.1.1) or our M-FPF which uses a random sample (see 2.4.2) or the M-FPF-MD which uses a random sample

[1]`http://citeseer.ist.psu.edu/`

and medoids (see section 2.4.3). Each of this algorithms has pros and cons. For example the standard FPF has theoretical guarantees instead the use of medoids has a better distribution of centers.

Table 6.3 shows a comparison of the 3 clustering algorithms for the k-center problem (FPF, M-FPF, M-FPF-MD) using the TS2 dataset. We obtained analogous results using the TS1 dataset. Observe that M-FPF obtains better results in all cases in terms of both Normalized Aggregate Goodness and Recall.

	Normalized Aggregate Goodness			Recall		
# Visited	FPF	M-FPF	M-FPF-MD	FPF	M-FPF	M-FPF-MD
3	0.733	0.756	0.755	5.324	5.688	5.612
6	0.780	0.799	0.786	6.044	6.352	6.068
9	0.785	0.809	0.796	6.12	6.556	6.28
12	0.787	0.815	0.808	6.156	6.692	6.448
15	0.791	0.823	0.810	6.196	6.752	6.488
18	0.794	0.829	0.812	6.236	6.824	6.548
21	0.798	0.834	0.816	6.284	6.892	6.616
24	0.801	0.837	0.818	6.32	6.94	6.628

Table 6.3. Comparison of the FPF, M-FPF, M-FPF-MD algorithms on TS2. Recall is a number in [0,10], Normalized Aggregated Goodness is a number in [0,1].

Query time of the 3 considered algorithms is very similar, thus we do not report detailed comparative results. Only note that query on clusterings made using M-FPF-MD is typically 0.1 second slower than the others because cluster cardinalities with this scheme tends to be not as well balanced as those obtained by the other clustering algorithms. Query time varies in the range of 0.3 seconds visiting 3 clusters up to 0.9 seconds visiting 24 clusters.

6.4.4 Fast multi-clustering for precision improvement

Results in the previous section show that M-FPF achieves a better performance with respect to the other algorithms. In this section we discuss experimental results that show how multi-clustering works better than simple clustering.

As explained in section 6.3.2 there are two main ways to query multi-clustering: querying independently each clustering or considering them as a single clustering. In most of the cases both these querying strategies return exactly the same set of objects in comparable time, thus we do not report experimental details.

In table 6.4 we report a comparison between multi-clustering and M-FPF in terms of Normalized Aggregate Goodness, Recall and query time.

Results show that multi-clustering achieves better quality than M-FPF but spends more time. Clearly, visiting the same number of clusters, multi-clustering querying is slower than querying a single clustering. This is due to the higher number of centers against which the query must be compared. In our case we made three

	Normalized Aggr. Goodness		Recall		Time	
# Visited	Multi	M-FPF	Multi	M-FPF	Multi	M-FPF
3	0.842	0.755	6.884	5.612	0.619	0.298
6	**0.887**	0.786	**7.688**	6.352	**0.692**	0.384
9	0.907	0.796	8.096	6.28	0.828	0.440
12	0.915	0.808	8.292	6.448	0.886	0.539
15	0.921	0.810	8.408	6.488	0.942	0.551
18	0.925	0.812	8.508	6.548	0.988	0.593
21	0.927	0.816	8.528	6.616	1.068	0.636
24	0.928	**0.818**	8.548	**6.628**	1.087	**0.691**

Table 6.4. Comparison of multi-clustering (multi) and M-FPF algorithms on TS2. Highlighted entries are results obtained with the same query time.

independent clusterings of 1000 elements in the case of TS2 (500 in the case of TS1). Thus this means an additional cost of 2000 distance invocations for an average cost of about 0.4 seconds. It is interesting to note that querying multi-clustering remains qualitatively better than M-FPF even in the case when the querying processes on the two clusterings are constrained to spend the same amount of time. In table 6.4 we highlight those results in which M-FPF and multi-clustering spend about the same time for querying. In this case multi-clustering had time to visit only 6 clusters, while M-FPF visited 24 clusters. Also in this case multi-clustering performs better than M-FPF in terms of both Normalized Aggregated Goodness and Competitive Recall.

6.4.5 Comparison with baseline algorithms

In this section we compare our Multi-clustering algorithm against the two baseline algorithms described in section 6.4.1 in the most general setting in which dynamically user-defined score are allowed.

Preprocessing time: as shown in table 6.5, the simple clustering strategy in [Chierichetti *et al.*, 2007] has preprocessing time close to ours (but quality/cost performance inferior to our scheme and to that in [Singitham *et al.*, 2004]), while [Singitham *et al.*, 2004] is noteworthy much slower. In a test with 100,000 documents, we gain a factor 30. In practice we could complete the preprocessing in one day compared to one month required by [Singitham *et al.*, 2004].

The difference in preprocessing performance between our solution and CellDec is due to three main factors:

- **The clustering algorithms**: as already seen in previous chapters, k-means algorithm is slower than M-FPF. This is still true also using (like in our implementation) a faster version of k-means [Phillips, 2002].

Dataset	TS1		
Algorithm	Our	CellDec	PODS07
Preprocessing time	5:28	215:48	7:18
Space (MB)	332.078	1407.656	1402.140
Dataset	TS2		
Algorithm	Our	CellDec	PODS07
Preprocessing time	20:13	636.80	22:56
Space (MB)	645.765	2738.671	2725.078

Table 6.5. Preprocessing time (in hours and minutes) and storage (in Megabytes) of the data structures generated by *CellDec*, PODS07 and our algorithm.

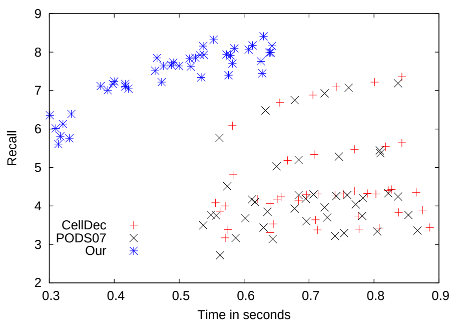
- **The embedding scheme**: the embedding of weights in the preprocessing forces CellDec to make a different clustering for each weight assignment. Note that in the CellDec space decomposition scheme, the number of weight assignments is proportional to the number of sources of evidence in the dataset. In our case, multi-clustering requires to make three clusterings independently from the number of sources of evidence presents in the data domain.

- **The use of centroids**: the general idea that a distance computation between two objects takes constant time is not always true for all the data types and distances. Clearly this assumption holds in cases like video data presented in chapter 5, where frames and centroids have the same number of features. Instead, in the case of tf-idf schema, to compute the cosine similarity between two objects, one needs to find the features in common between them. This can be efficiently done storing features in a hash table and, for each feature of the smaller object, looking in the bigger object if the feature is present. This minimizes the number of hashes needed, but does not resolve the problem that hashing cost depends on the number of elements in the table.

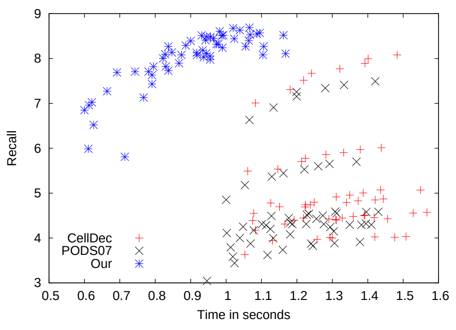
Query quality and speed: figure 6.7 shows the query time/recall tradeoff of the three methods. In the graph each dot represents the average of 250 queries for a given choice of clusters to visit and user weights (see table 6.2). The 250 queries were selected only once and submitted to each algorithm and weights assignment. Our method is clearly dominant giving consistently better quality results in less time. Quality data are also given in tabular form in table 6.6, table 6.7, table 6.8 and table 6.9.

The query time as a function of the number of visited clusters is reported in figure 6.8 and shows clearly a speed up factor of two. Also in this case the speed up is due to the choice of avoiding to use centroids.

The top portion of table 6.6, table 6.7, table 6.8 and table 6.9. correspond to the case of equal weights, that is equivalent to the unweighted case as already

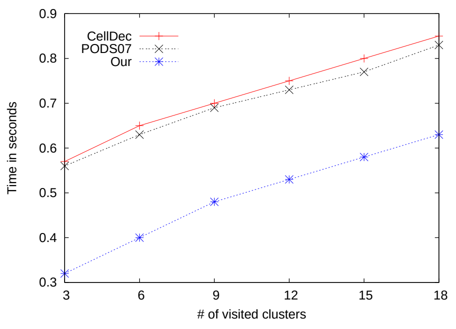


(a) TS1

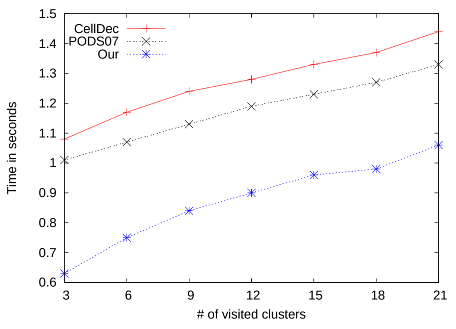


(b) TS2

Figure 6.7. Recall of 10 nearest neighbors as a function of query time. Each point in the graph is the average of measurements of all queries for a class of weights and a number of visited clusters. The points in the upper left corner of the graphs corresponding to our algorithm show clear dominance.



(a) Data Set TS1



(b) Data Set TS2

Figure 6.8. Average query time (in seconds) over all queries in function of the number of visited clusters.

partially shown in sections 6.4.3 and 6.4.4. In the entries of table 6.6 and table 6.8, for unequal weights our scheme is vastly superior in recall, even doubling the number of true k-nearest neighbors found using less time over both baselines. The overall quality of the retrieved nearest neighbors, as measure via the *normalized aggregated goodness*, is also improved: this indicates that our method is robust and stable relative to the baselines.

6.5 Conclusions

In this chapter we tackled the similarity searching problem in semi-structured text documents. We provided an approximated solution based on cluster pruning. Our method is based on the observation that there is a relationship between the k-center problem and the similarity searching problem, thus the former can be helpful to approximate the latter. We also introduced multi-clustering that, by introducing redundancy in the input data, makes the whole system more stable in terms of output quality. Moreover, we have shown that a difficult searching problem with dynamically chosen weights can be reduced, thanks to the linearity properties of the cosine similarity metric, to a simpler static search problem. For this problem we provide an efficient and effective method that is competitive with the state of the art techniques for large semi-structured textual databases.

		Data Set TS1 = 50K docs.					
Visited clusters		3	6	9	12	15	18
		Weights: query 0.33-0.33-0.34 - CellDec 1-1-1					
Recall	CellDec	6.088	6.688	6.884	7.096	7.22	7.36
	PODS07	5.768	6.484	6.752	6.928	7.072	7.188
	Multi	6.016	7.172	7.64	7.852	7.94	7.992
		Weights: query 0.4-0.4-0.2 - CellDec 1-1-1					
Recall	CellDec	4.812	5.184	5.336	5.472	5.544	5.644
	PODS07	4.512	5.032	5.196	5.284	5.372	5.444
	Multi	6.128	7.168	7.64	7.832	7.916	7.984
		Weights: query 0.2-0.4-0.4 - CellDec 1-1-1					
Recall	CellDec	3.864	4.06	4.148	4.284	4.312	4.404
	PODS07	3.772	4.168	4.284	4.3	4.284	4.328
	Multi	6.356	7.116	7.516	7.624	7.704	7.76
		Weights: query 0.4-0.2-0.4 -CellDec 1-1-1					
Recall	CellDec	4.0	4.176	4.292	4.312	4.324	4.352
	PODS07	3.752	4.104	4.188	4.256	4.204	4.244
	Multi	5.608	7.048	7.664	7.932	8.096	8.176
		Weights: query 0.2-0.6-0.2 - CellDec 0.5-1-0.5					
Recall	CellDec	4.084	4.18	4.236	4.312	4.388	4.428
	PODS07	3.496	3.684	3.848	3.932	3.964	4.04
	Multi	6.392	7.008	7.22	7.344	7.4	7.448
		Weights: query 0.6-0.2-0.2 - CellDec 1-0.5-0.5					
Recall	CellDec	3.172	3.308	3.376	3.396	3.424	3.44
	PODS0 7	2.716	3.14	3.216	3.292	3.336	3.36
	Multi	5.76	7.236	7.848	8.156	8.32	8.412
		Weights: query 0.2-0.2-0.6 - CellDec 0.5-0.5-1					
Recall	CellDec	3.384	3.532	3.64	3.736	3.832	3.892
	PODS07	3.168	3.436	3.604	3.7	3.74	3.764
	Multi	5.812	7.108	7.728	7.92	8.064	8.164

Table 6.6. Quality results of the compared algorithms on TS1. Recall is a number in [0,10], Data as a function of the number of visited clusters.

		Data Set TS1 = 50K docs.					
Visited clusters		3	6	9	12	15	18
		Weights: query 0.33-0.33-0.34 - CellDec 1-1-1					
NAG	CellDec	0.779	0.822	0.841	0.854	0.865	0.876
	PODS07	0.753	0.816	0.831	0.842	0.852	0.863
	Multi	0.776	0.838	0.863	0.876	0.879	0.882
		Weights: query 0.4-0.4-0.2 - CellDec 1-1-1					
NAG	CellDec	0.769	0.811	0.830	0.844	0.855	0.866
	PODS07	0.743	0.807	0.821	0.832	0.842	0.853
	Multi	0.778	0.833	0.856	0.869	0.872	0.875
		Weights: query 0.2-0.4-0.4 - CellDec 1-1-1					
NAG	CellDec	0.698	0.737	0.756	0.774	0.786	0.797
	PODS07	0.679	0.738	0.753	0.763	0.772	0.783
	Multi	0.762	0.807	0.827	0.836	0.840	0.842
		Weights: query 0.4-0.2-0.4 -CellDec 1-1-1					
NAG	CellDec	0.791	0.830	0.845	0.851	0.858	0.865
	PODS07	0.757	0.815	0.828	0.839	0.849	0.856
	Multi	0.786	0.869	0.901	0.916	0.922	0.926
		Weights: query 0.2-0.6-0.2 - CellDec 0.5-1-0.5					
NAG	CellDec	0.770	0.802	0.818	0.828	0.842	0.848
	PODS07	0.668	0.702	0.734	0.751	0.762	0.769
	Multi	0.740	0.775	0.788	0.799	0.801	0.805
		Weights: query 0.6-0.2-0.2 - CellDec 1-0.5-0.5					
NAG	CellDec	0.809	0.845	0.861	0.867	0.870	0.874
	PODS07	0.725	0.793	0.823	0.839	0.849	0.856
	Multi	0.795	0.883	0.913	0.930	0.936	0.939
		Weights: query 0.2-0.2-0.6 - CellDec 0.5-0.5-1					
NAG	CellDec	0.773	0.806	0.828	0.840	0.856	0.866
	PODS07	0.737	0.785	0.812	0.825	0.835	0.840
	Multi	0.773	0.859	0.887	0.896	0.902	0.908

Table 6.7. Quality results of the compared algorithms on TS1. Normalized Aggregated Goodness is a number in [0,1]. Data as a function of the number of visited clusters.

		Data Set TS2 = 100K docs.						
Visited clusters		3	6	9	12	15	18	21
		Weights: query 0.33-0.33-0.34 - CellDec 1-1-1						
Recall	CellDec	7.008	7.308	7.516	7.672	7.772	7.892	7.996
	PODS07	6.044	6.632	6.908	7.156	7.256	7.34	7.412
	Multi	6.884	7.688	8.096	8.292	8.408	8.508	8.528
		Weights: query 0.4-0.4-0.2 - CellDec 1-1-1						
Recall	CellDec	5.492	5.536	5.704	5.776	5.86	5.904	5.972
	PODS07	4.852	5.18	5.368	5.444	5.528	5.6	5.652
	Multi	6.848	7.708	8.08	8.268	8.392	8.448	8.48
		Weights: query 0.2-0.4-0.4 - CellDec 1-1-1						
Recall	CellDec	4.78	4.692	4.796	4.916	4.956	5.004	5.072
	PODS07	4.0	4.2	4.344	4.428	4.5	4.552	4.58
	Multi	6.96	7.708	8.004	8.076	8.184	8.24	8.268
		Weights: query 0.4-0.2-0.4 -CellDec 1-1-1						
Recall	CellDec	4.388	4.396	4.444	4.4	4.412	4.444	4.456
	PODS07	3.792	4.172	4.284	4.312	4.312	4.308	4.296
	Multi	5.988	7.272	7.82	8.136	8.44	8.516	8.608
		Weights: query 0.2-0.6-0.2 - CellDec 0.5-1-0.5						
Recall	CellDec	4.548	4.696	4.74	4.74	4.792	4.828	4.848
	PODS07	4.112	4.252	4.308	4.444	4.492	4.516	4.54
	Multi	7.024	7.632	7.824	7.976	8.028	8.056	8.08
		Weights: query 0.6-0.2-0.2 - CellDec 1-0.5-0.5						
Recall	CellDec	3.632	3.944	3.968	4.012	4.0	4.024	4.016
	PODS07	3.044	3.44	3.62	3.736	3.824	3.876	3.884
	Multi	5.808	7.132	7.728	8.128	8.32	8.488	8.632
		Weights: query 0.2-0.2-0.6 - CellDec 0.5-0.5-1						
Recall	CellDec	4.176	4.312	4.424	4.48	4.5	4.508	4.556
	PODS07	3.584	3.876	3.996	4.08	4.148	4.244	4.292
	Multi	6.52	7.432	7.896	8.116	8.32	8.4	8.52

Table 6.8. Quality results of the compared algorithms on TS2. Recall is a number in [0,10], Data as a function of the number of visited clusters.

		Data Set TS2 = 100K docs.						
Visited clusters		3	6	9	12	15	18	21
		Weights: query 0.33-0.33-0.34 - CellDec 1-1-1						
NAG	CellDec	0.858	0.876	0.891	0.905	0.914	0.919	0.924
	PODS07	0.779	0.827	0.851	0.867	0.874	0.881	0.887
	Multi	0.842	0.887	0.907	0.915	0.921	0.925	0.927
		Weights: query 0.4-0.4-0.2 - CellDec 1-1-1						
NAG	CellDec	0.852	0.869	0.884	0.899	0.908	0.914	0.918
	PODS07	0.771	0.819	0.843	0.860	0.867	0.875	0.881
	Multi	0.836	0.883	0.903	0.909	0.916	0.919	0.921
		Weights: query 0.2-0.4-0.4 - CellDec 1-1-1						
NAG	CellDec	0.798	0.811	0.828	0.847	0.857	0.863	0.868
	PODS07	0.725	0.763	0.785	0.800	0.808	0.817	0.825
	Multi	0.819	0.870	0.883	0.887	0.896	0.898	0.900
		Weights: query 0.4-0.2-0.4 -CellDec 1-1-1						
NAG	CellDec	0.834	0.855	0.863	0.868	0.874	0.877	0.880
	PODS07	0.762	0.813	0.834	0.848	0.852	0.855	0.858
	Multi	0.817	0.895	0.924	0.934	0.943	0.946	0.949
		Weights: query 0.2-0.6-0.2 - CellDec 0.5-1-0.5						
NAG	CellDec	0.870	0.900	0.914	0.923	0.927	0.928	0.930
	PODS07	0.770	0.795	0.816	0.835	0.844	0.852	0.859
	Multi	0.814	0.849	0.861	0.867	0.873	0.876	0.878
		Weights: query 0.6-0.2-0.2 - CellDec 1-0.5-0.5						
NAG	CellDec	0.803	0.852	0.861	0.865	0.869	0.874	0.874
	PODS07	0.702	0.784	0.823	0.836	0.849	0.860	0.862
	Multi	0.812	0.891	0.921	0.936	0.945	0.953	0.957
		Weights: query 0.2-0.2-0.6 - CellDec 0.5-0.5-1						
NAG	CellDec	0.853	0.869	0.884	0.889	0.894	0.896	0.903
	PODS07	0.755	0.807	0.834	0.845	0.852	0.862	0.866
	Multi	0.837	0.889	0.914	0.923	0.933	0.936	0.939

Table 6.9. Quality results of the compared algorithms on TS2. Normalized Aggregated Goodness is a number in [0,1]. Data as a function of the number of visited clusters.

Chapter 7

Conclusions

Abstract

This thesis completes a Ph.D course of three years in which we focused our studies in information retrieval. The majority of our studies were devoted on the clustering problem with particular attention to the Web scenario. We designed a family of algorithms for the k-center problem, gave a novel definition of the concept of medoid and tested our algorithms on three important applications of Web information retrieval: web snippets clustering, video summarization and similarity searching. These applications raised some important related problems for which we found novel solutions. Snippets clustering is very often conjugated with the task of cluster labelling, that is the problem of distilling a synthetic and descriptive label for the cluster content. Video summarization required to study many aspects: the video features and distance to use, the optimal number of clusters and the selection of the most representative element for each cluster. In similarity searching of semi-structured text documents one should want to allow the user to assign different weights to each field at query time. This requirement raises the problem of vector score aggregation which complicates preprocessing because at that time weights are unknown and thus one should build a data structure able to handle all the possible weight assignments. We conclude this thesis discussing some preliminary ideas about on-going research directions. We observed that clustering algorithms spend most of the processing time in adding points to the clusters, searching for the closest center. This time can be reduced using a keen similarity searching scheme.

7.1 Results

In this thesis we focused on the problem of clustering in the web scenario. We began our studies from the Furthest Point First heuristic [Gonzalez, 1985] for the k-center problem by Gonzalez. This algorithm returns a solution which is 2-competitive with respect to the optimal solution and runs in linear time. This result is demonstrated to be the best possible approximation unless $P = NP$. For its small computational cost we considered the Furthest Point First as a good starting point for our studies. Our first result was FPF, a variant of the Furthest Point First heuristic in which we exploit the triangular inequality to reduce the overall number of distance computations without changing the final output. Despite in practice FPF is faster than the original Gonzalez's algorithm, in theory the number of distance computations saved by this algorithm depends from data, therefore FPF has the same worst case complexity bound of the original algorithm.

Later we investigated a different clustering strategy. We observed that, building the k-center set, the choice of FPF to elect as the i-th center the furthest point from the $i - 1$ already selected centers tends to select points that stand close to the boundaries of the space. This also means that if the dataset contains many outliers they are likely to be elected as cluster centers. We empirically observed that a random sample of the input data, enough large to be representative of the whole dataset is likely to have in proportion less outliers. For our purpose the size of the random sample should be proportional to the number n of objects and the number k of desired clusters, thus we choose a sample of size $\sqrt{nk}$. We run FPF over this sample in order to obtain a k-center set, then we add one by one the remaining points to complete the clustering. We called this scheme M-FPF.

Experimentally M-FPF was shown to be more robust than FPF with respect to noise. The tendency of centers to be "not in the center" of their own clusters is mitigated, but it is still present. Thus we replaced centers with medoids in the scheme of M-FPF. Given a certain set of points P, let a and b be its diametral elements. Consider now the point m, not necessarily in P, which is the middle point of a and b. The medoid is the point of P which is closest to m. The exact computation of the medoid is quadratic in the number of elements in the cluster, but it can be computed in linear time using an approximate method for finding the diametral points. Using medoids we modified the M-FPF in the following manner: firstly from the input points we extract a random sample of $\sqrt{nk}$ elements, then we cluster them using FPF and for each cluster we compute its medoid. We add one by one the remaining points at each step updating the medoid of the involved cluster. We called this scheme M-FPF-MD.

As shown in chapters 4, 5 and 6 we applied the clustering algorithm for different practical applications of web information retrieval. Each of these applications raised some interesting problems.

In chapter 4 we described *armil* a completely working clustering meta-search engine that exploits M-FPF-MD to cluster snippets collected from auxiliary search engines. In this case cluster labelling is not less important than clustering. In fact a good cluster with a poor descriptive label is likely to be ignored by the user. We designed a novel labelling method that is completely independent from the data domain and clustering algorithm. It works in two steps: in the first phase it collects local statistics inside each cluster to extract the most relevant terms of the cluster, then it uses a variation of the information gain measure to disambiguate keywords that are considered relevant for more than one cluster. At the end it ranks all the sentences of the cluster according with the relevant terms and returns the most appropriate piece of text as label.

In chapter 5 we described *ViSto* a web application to create visual storyboards. In this task, clustering is used to select the most representative frames (or scenes) that will form the storyboard. We faced both the problem of static and dynamic storyboard production. A first important issue we addressed is the automatic detection of a possible reasonable size for the storyboard to be suggested to the final user. We designed an adaptive method that analyzes the distribution of distances between pairs of consecutive frames to detect and count video cuts in order to elaborate a suggestion for the user. Another important aspect we studied concerns the use of frames or scenes as input for the clustering algorithm for dynamic summaries. We found that the use of frames introduces an unjustified bias in favor of longer scenes, moreover, despite scenes have proportionally a less informative representation than frames, this does not cause a performance degradation. Finally the use of scenes should be preferable because they are less with respect to frames yielding a resulting shorter clustering time. A final important result, we obtained coping with this task, is a faster approximation of M-FPF-MD for this type of data. In fact, by observing that consecutive frames are likely to be very close and, clearly, the closer are two frames the more probable is that they will be assigned to the same cluster, if a certain frame is enough close to its predecessor, we can assign it to the same cluster without comparing it with all the centers.

In chapter 6 we cope with the similarity searching problem following the cluster pruning approach. In this model we observed that if the query point is inside a cluster, there is a relationship between the k-center problem and the similarity searching problem. This is the case for example of the search for related documents in Citeseer or similar documents in Google. This relationship justifies our choice in using our algorithms for similarity searching. We also studied the benefits in terms of marginal utility due to the visit of more clusters or due to the choice of making more clusters. We observed that most of the nearest neighbors are found in the first few clusters, thus, after the visit of three or four clusters, the time spent in visiting a new cluster is not justified by the corresponding improvement for the recall. We proposed a novel scheme for cluster pruning in which, instead of having just a clustering, we make a small number of independent clusterings, then we visit a certain (smaller than the standard scheme) number of clusters for each clustering and merge the solutions. In order to avoid duplicate distance computations due to redundancy we cache distance computations. The overall result of this scheme is an improvement of recall in less, at most the same, time of the standard scheme. The last issue related to similarity searching we coped with, is the vector score aggregation problem. Let us consider to have a semi-structured text, you may want to allow the user to give a different weight to each field at query time. In this case the problem is that each weights assignment affects differently the aggregate distance function; moreover at preprocessing time, that is at clustering time, weights are not known. Thus one should make many clusterings with different weights assignments to cover all the spectrum of possible assignments. We exploited some linearity properties of the cosine similarity that allowed us to handle weights only at query time without any clustering replication.

7.2 Future research directions

The great majority of the running time for FPF and related clustering algorithms is spent searching the closest center of a certain point we want to insert into a cluster. In fact, once the k-center set is established, for each of the remaining points we must find the closest center to insert it in the appropriate cluster. This operation requires k distance computations for each item, thus, the higher is k, the slower is the clustering algorithm. In chapter 5 we were able to reduce this time taking advantage of the peculiar distribution of data in that domain. We observed that the operation of finding the closest center is in practice a similarity searching problem, similar to that we addressed in chapter 6, where the item to be inserted into the clustering is the query point. We plan to develop a general scheme for approximate FPF that takes advantage from the approximate similarity searching to reduce the algorithm running time without any data dependence. Approximate clustering finds application in those cases in which we must deal with a huge amount of documents, the number of requested clusters is very high and a certain degree of error in the clustering can be tolerated. This is the case for example of recommended systems. In this problem, a company has huge database of items and wants to create a large number of user profiles. When a new user matches a profile the system recommends the articles of the matched profile. This problem can be reformulated as a clustering problem in which the database is clustered and each cluster is a profile. The distance between two items depends from the users feedback. The insertion of an item into the wrong profile will be ignored by the customer in the worst case .

The outline of the approximated FPF should be:

1. Split the input points into two sets S and T such that $|S| = \sqrt{nk}$ and cluster S with FPF,
2. let $C = \{c_1, \ldots, c_k\}$ be the obtained cluster centers; cluster C using FPF and make $\sqrt{k}$ clusters with centers $P = \{p_1, \ldots, p_{\sqrt{k}}\}$,
3. for each point $q \in T$:
    - scan P and find the closest p_i,
    - scan the corresponding cluster and find the closest c_i,

- add q to the corresponding cluster.

The above procedure running time is the summation of the following three components:

$$\begin{array}{ccc} \overbrace{\sqrt{n}kk}^{Step1} & + \overbrace{\sqrt{k}k}^{Step2} & + \overbrace{(n-\sqrt{n}k)\sqrt{k}}^{Step3} \\ \Downarrow & \Downarrow & \Downarrow \\ O((\sqrt{n}k)\sqrt{k}) & O(n\sqrt{k}) & O(n\sqrt{k}) \end{array}$$

The overall running time is $O(\max(\sqrt{n}k, n)\sqrt{k})$ which has minimum for $k = \sqrt{n}$.

Bibliography

[Achlioptas, 2003] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[Agirre and Rigau, 1996] E. Agirre and G. Rigau. Word sense disambiguation using conceptual density. In *Proceedings of COLING'96*, pages 16–22, 1996.

[Bawa *et al.*, 2005] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 651–660, 2005.

[Bingham and Mannila, 2001] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, New York, NY, USA, 2001. ACM.

[Bradley and Fayyad, 1998] Paul S. Bradley and Usama M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 91–99, Madison, US, 1998.

[Brin, 1995] Sergey Brin. Near neighbor search in large metric spaces. In *The VLDB Journal*, pages 574–584, 1995.

[Brown *et al.*, 1991] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. Word-sense disambiguation using statistical methods. In *ACL 29*, pages 264–270, 1991.

[Can *et al.*, 2004] Fazli Can, Ismail Sengor Altingovde, and Engin Demir. Efficiency and effectiveness of query processing in cluster-based retrieval. *Inf. Syst.*, 29(8):697–717, 2004.

[Cantone *et al.*, 2005] Domenico Cantone, Alfredo Ferro, Alfredo Pulvirenti, Diego Reforgiato Recupero, and Dennis Shasha. Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):535–550, 2005.

[Charikar *et al.*, 1997] Moses Charikar, Chandra Chekuri, Tomas Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635, New York, NY, USA, 1997. ACM Press.

[Charikar, 2002] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of STOC-02, 34th Annual ACM Symposium on the Theory of Computing*, pages 380–388, Montreal, CA, 2002.

[Chávez and Navarro, 2000] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the 6th International Symposium on String Processing and Information Retrieval (SPIRE'2000)*, pages 75–86. IEEE CS Press, 2000.

[Chávez *et al.*, 2001] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 3(3):273–321, 2001.

[Cheng *et al.*, 2003] D. Cheng, R. Kannan, S. Vempala, and G. Wang. On a recursive spectral algorithm for clustering from pairwise similarities. Technical Report MIT-LCS-TR-906, Massachusetts Institute of Technology, Cambridge, US, 2003.

[Chierichetti *et al.*, 2007] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. Finding near neighbors through cluster pruning. In *Proceedings of ACM PODS*, 2007. To appear.

[Ciaccia and Patella, 2002] Paolo Ciaccia and Marco Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.*, 27(4):398–437, 2002.

[Clarkson, 2006] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[Cover and Thomas, 1991] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, New York, US, 1991.

[Cutting *et al.*, 1992] Douglass R. Cutting, Jan O. Pedersen, David Karger, and John W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of SIGIR-92, 15th ACM International*

Conference on Research and Development in Information Retrieval, pages 318–329, Kobenhavn, DK, 1992.

[Deerwester *et al.*, 1990] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[DeMenthon *et al.*, 1998] D. DeMenthon, V. Kobla, and D. Doermann. Video summarization by curve simplification. In *Proc. of Computer Visualization and Pattern Recognition*, 1998.

[Dhillon *et al.*, 2002] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. Enhanced word clustering for hierarchical text classification. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 191–200, New York, NY, USA, 2002. ACM.

[Elkan, 2003] Charles Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.

[Fagin *et al.*, 2003] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 301–312, New York, NY, USA, 2003. ACM Press.

[Farnstrom *et al.*, 2000] Fredrik Farnstrom, James Lewis, and Charles Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explor. Newsl.*, 2(1):51–57, 2000.

[Feder and Greene, 1988] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, New York, NY, USA, 1988. ACM Press.

[Ferragina and Gulli, 2005] Paolo Ferragina and Antonio Gulli. A personalized search engine based on Web-snippet hierarchical clustering. In *Special Interest Tracks and Poster Proceedings of WWW-05, 14th International Conference on the World Wide Web*, pages 801–810, Chiba, JP, 2005.

[Figueroa *et al.*, 2006] Karina Figueroa, Edgar Chávez, Gonzalo Navarro, and Rodrigo Paredes. On the least cost for proximity searching in metric spaces. In *5th International Workshop on Experimental Algorithms (WEA)*, volume 4007 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2006.

[Furini *et al.*, 2007] M. Furini, F. Geraci, M. Montangero, and M. Pellegrini. VISTO: VIsual Storyboard for Web Video Browsing. In *CIVR '07: Proceedings of the ACM International Conference on Image and Video Retrieval*, July 2007.

[Furini, 2007] M. Furini. On ameliorating the perceived playout quality in chunk-driven p2p media streaming systems. In *ICC '07: Proceedings of the IEEE International Conference on Communications*, 2007.

[G. W. Milligan, 1985] M. C. Cooper G. W. Milligan. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[Gale *et al.*, 2004] William A. Gale, Kenneth W. Church, and David Yarowsky. A method for disambiguating word senses in a large corpus, 2004.

[Geraci *et al.*, 2006a] F. Geraci, M. Pellegrini, F. Sebastiani, and M. Maggini. Cluster generation and cluster labelling for web snippets. In *Proceedings of the 13th Symposium on String Processing and Information Retrieval (SPIRE 2006)*, pages 25–36, Glasgow, UK., October 2006. Volume 4209 in LNCS.

[Geraci *et al.*, 2006b] Filippo Geraci, Marco Pellegrini, Paolo Pisati, and Fabrizio Sebastiani. A scalable algorithm for high-quality clustering of Web snippets. In *Proceedings of SAC-06, 21st ACM Symposium on Applied Computing*, pages 1058–1062, Dijon, FR, 2006.

[Geraci *et al.*, 2007] Filippo Geraci, Mauro Leoncini, Manuela Montangero, Marco Pellegrini, and M. Elena Renda. Fpf-sb: a scalable algorithm for microarray gene expression data clustering. In *Proceedings of 1st International Conference on Digital Human Modeling*, 2007.

[Gong and Liu, 2003] Yihong Gong and Xin Liu. Video summarization and retrieval using singular value decomposition. *Multimedia Syst.*, 9(2):157–168, 2003.

[Gonzalez, 1985] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2/3):293–306, 1985.

[Guha *et al.*, 1998] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM Press.

[Guha *et al.*, 2003] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[Hadi *et al.*, 2006] Youssef Hadi, Fedwa Essannouni, and Rachid Oulad Haj Thami. Video summarization by k-medoid clustering. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1400–1401, New York, NY, USA, 2006. ACM Press.

[Hafner *et al.*, 1998] J. Hafner, N. Megiddo, and E. Upfal. Fast query search in large dimansion database. US patent 5848404, 1998.

[Hanjalic and Zhang, 1999a] A. Hanjalic and H. Zhang. An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis. *IEEE Trans. Cir. and Sys. for Video Tech.*, 9(8):1280–1289, 1999.

[Hanjalic and Zhang, 1999b] A Hanjalic and H J Zhang. An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis. In *IEEE Trans. on Circuits and Systems for Video Technology*, volume 9, pages 1280–1289, 1999.

[Haveliwala *et al.*, 2000] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web. In *WebDB (Informal Proceedings)*, pages 129–134, 2000.

[Haveliwala *et al.*, 2002] Taher H. Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk. Evaluating strategies for similarity search on the Web. In *Proceedings of WWW-02, 11th International Conference on the World Wide Web*, pages 432–442, Honolulu, US, 2002.

[Hearst and Pedersen, 1996] Marti A. Hearst and Jan O. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 76–84, Zürich, CH, 1996.

[Hochbaum and Shmoys, 1985] D. S. Hochbaum and D. B. Shmoys. A best possible approximation algorithm for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[Ide and Veronis, 1998] Nancy Ide and Jean Veronis. Word sense disambiguation: The state of the art, 1998.

[Indyk and Motwani, 1998] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[Indyk, 1999] Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of STOC-99, ACM Symposium on Theory of Computing*, pages 428–434, 1999.

[Kalantari and McDonald, 1983] Iraj Kalantari and Gerard McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Trans. Software Eng.*, 9(5):631–634, 1983.

[Karypis *et al.*, 1999] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar NEWS. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[Kaski, 1998] Samuel Kaski. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Proceedings of IJCNN'98, International Joint Conference on Neural Networks*, volume 1, pages 413–418, Piscataway, NJ, 1998. IEEE Service Center.

[Kaufman and Rousseeuw, 1990] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley, 1990. A Wiley-Interscience publication.

[Kohonen, 2001] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[Kowalski, 1997] Gerald Kowalski. *Information Retrieval Systems: Theory and Implementation*. Boston Kluwer Academic Publishers, 1997.

[Kummamuru *et al.*, 2004] Krishna Kummamuru, Rohit Lotlikar, Shourya Roy, Karan Singal, and Raghu Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of WWW-04, 13th International Conference on the World Wide Web*, pages 658–665, New York, NY, 2004.

[Kural *et al.*, 1993] Yasemin Kural, Stephen Robertson, and Susan Jones. Deciphering cluster representations. *Information Processing and Management*, 37:593–601, 1993.

[Kural *et al.*, 1999] Yasemin Kural, Stephen Robertson, and Susan Jones. Clustering information retrieval search outputs. In *Proceedings of the 21st BCS IRSG Colloquium on Information Retrieval*, Glasgow, UK, 1999. http://www.bcs.org/server.php?show=ConWebDoc.4252.

[Lamrous and Tailerb, 2006] Sid Lamrous and Mounira Tailerb. Divisive hierarchical k-means. In *CIMCA '06: Proceedings of the International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce*, page 18, Washington, DC, USA, 2006. IEEE Computer Society.

[Lance and Williams, 1967] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies 1. Hierarchical systems. *The Computer Journal*, 9(4):373–380, February 1967.

[Larsen and Aone, 1999] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of KDD-99, 5th ACM International Conference on Knowledge Discovery and Data Mining*, pages 16–22, 1999.

[Lawrie and Croft, 2003] Dawn J. Lawrie and W. Bruce Croft. Generating hierarchical summaries for Web searches. In *Proceedings of SIGIR-03, 26th ACM In-*

ternational Conference on Research and Development in Information Retrieval, pages 457–458, 2003.

[Lesk, 1986] Michael Lesk. Automatic sense disambiguation: How to tell a pine cone from an ice cream cone. In *Proc. of the 1986 SIGDOC Conference*, pages 24–26, New York, 1986. Association for Computing Machinery.

[Li and Abe, 1998] Hang Li and Naoki Abe. Word clustering and disambiguation based on co-occurrence data. In *Proceedings of the 17th international conference on Computational linguistics*, pages 749–755, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[Lin and Gunopulos, 2003] J. Lin and D. Gunopulos. Dimensionality reduction by random projection and latent semantic indexing. In *Proceedings of the Text Mining Workshop at the 3rd SIAM International Conference on Data Mining*, 2003.

[Linden, 2005] Krister Linden. *Word Sense Discovery and Disambiguation*. PhD thesis, University of Helsinki, Faculty of Arts, Department of General Linguistics, 2005.

[Lloyd, 1957] S.P. Lloyd. Least squares quantization in PCM. Technical report, Bell Laboratories, 1957. Reprinted in *IEEE Transactions on Information Theory* IT-28(2), 1982, pp. 129–137.

[Maarek *et al.*, 2000] Yoelle Maarek, Ron Fagin, Israel Ben-Shaul, and Dan Pelleg. Ephemeral document clustering for Web applications. Technical Report RJ 10186, IBM, San Jose, US, 2000. http://citeseer.ist.psu.edu/maarek00ephemeral.html.

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

[Manjunath *et al.*, 2001] B S Manjunath, J R Ohm, V V Vasudevan, and A Yamada. Color and texture descriptors. *IEEE Transactions on Circuits and Systems For Video Technology*, 11:703–715, 2001.

[Meng *et al.*, 2002] Weiyi Meng, Clement T. Yu, and King-Lup Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[Miller *et al.*, 2006] George A. Miller, Christiane Fellbaum, Randee Tengi, Pamela Wakefield, Rajesh Poddar, Helen Langone, and Benjamin Haskell. Wordnet: A lexical database for english, 2006.

[Miller, 1990] George A. Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 1990.

[Milligan and Cooper, 1985] Glenn W. Milligan and Martha C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, V50(2):159–179, June 1985.

[Mundur *et al.*, 2006a] P. Mundur, Y. Rao, and Y. Yesha. http://www.csee.umbc.edu/ yongrao1/delaunay.html, 2006.

[Mundur *et al.*, 2006b] P. Mundur, Y. Rao, and Y. Yesha. Keyframe-based video summarization using delaunay clustering. *International Journal on Digital Libraries*, 6(2):219–232, 2006.

[Nam and Tewfik, 1999] J. Nam and A.H. Tewfik. Video abstract of video. In *IEEE 3rd Workshop on Multimedia Signal Processing*, pages 117–122, 1999.

[Ng and Han, 1994] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[Ngo *et al.*, 2005] Chong-Wah Ngo, Yu-Fei Ma, and HongJiang Zhang. Video summarization and scene detection by graph modeling. *IEEE Trans. Circuits Syst. Video Techn.*, 15(2):296–305, 2005.

[NIST, 2005] NIST. http://trec.nist.gov/data/reuters/reuters.html, 2005.

[Oh *et al.*, 2004] JungHwan Oh, Quan Wen, JeongKyu Lee, and Sae Hwang. *Video Abstraction*, chapter XIV, pages 321–346. Idea Group Inc.and IRM Press, 2004.

[Ömer Egeciolu and Kalantari, 1989] Ömer Egeciolu and Bahman Kalantari. Approximating the diameter of a set of points in the euclidean space. *Inf. Process. Lett.*, 32(4):205–211, 1989.

[Open-Video, 2002] Open-Video. The open video projec, http://www.open-video.org, 2002.

[Osinski and Weiss, 2004] Stanislaw Osinski and Dawid Weiss. Conceptual clustering using Lingo algorithm: Evaluation on Open Directory Project data. In *Proceedings of IIPWM-04, 5th Conference on Intelligent Information Processing and Web Mining*, pages 369–377, Zakopane, PL, 2004.

[Papineni, 2001] Kishore Papineni. Why inverse document frequency? In *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, pages 1–8, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[Peña *et al.*, 1999] José Manuel Peña, Jose Antonio Lozano, and Pedro Larrañaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20(10):1027–1040, 1999.

[Phillips, 2002] Steven J. Phillips. Acceleration of k-means and related clustering algorithms. In *Proceedings of ALENEX-02, 4th International Workshop on Algorithm Engineering and Experiments*, pages 166–177, San Francisco, US, 2002.

[Porter, 1980] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Rousseeuw, 1987] P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal ofCcomputational and Applied Mathematics*, 20:53–65, 1987.

[Sanderson, 2000] Mark Sanderson. Retrieving with good sense. *Information Retrieval*, 2(1):49–69, 2000.

[Savaresi *et al.*, 2002] Sergio M. Savaresi, Daniel Boley, Sergio Bittanti, and Giovanna Gazzaniga. Cluster selection in divisive clustering algorithms. In Robert L. Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *SDM*. SIAM, 2002.

[Selim and Ismail, 1984] S.Z. Selim and M.A. Ismail. K-means type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):81–87, 1984.

[Shahraray and Gibbon, 1995] B. Shahraray and D. C. Gibbon. Automatic generation of pictorial transcripts of video programs. In A. A. Rodriguez and J. Maitan, editors, *Proc. SPIE Vol. 2417, p. 512-518, Multimedia Computing and Networking 1995, Arturo A. Rodriguez; Jacek Maitan; Eds.*, pages 512–518, March 1995.

[Shamir and Sharan, 2002] R. Shamir and R. Sharan. *Algoritmic Approaches to Clustering Gene Expression Data, Current Topics in Computational Molecular Biology*. MIT Press, 2002.

[Singitham *et al.*, 2004] Pavan Kumar C. Singitham, Mahathi S. Mahabhashyam, and Prabhakar Raghavan. Efficiency-quality tradeoffs for vector score aggregation. In *VLDB Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada*, pages 624–635, 2004.

[Sitarama *et al.*, 2003] Shankaran Sitarama, Uma Mahadevan, and Mani Abrol. Efficient cluster representation in similar document search. In *WWW (Alternate Paper Tracks)*, 2003.

[Smellie, 2004] A. Smellie. Accelerated k-means clustering in metric spaces. *Journal of Chemical Information and Modeling*, 44(6):1929–1935, 2004.

[Stokoe *et al.*, 2003] Christopher Stokoe, Michael P. Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 159–166. ACM Press, 2003.

[Strehl *et al.*, 2000] Alexander Strehl, Joydeep Ghosh, and Raymond J. Mooney. Impact of similarity measures on Web page clustering. In *Proceedings of the AAAI Workshop on AI for Web Search*, pages 58–64, Austin, US, 2000.

[Strehl, 2002] Alexander Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*. PhD thesis, University of Texas, Austin, US, 2002.

[Tan and Lu, 2003] Yap-Peng Tan and Hong Lu. Video scene clustering by graph partitioning. *Electronics Letters*, 39(11):841–842, 2003.

[Tan *et al.*, 2005] Xiaoyang Tan, Songcan Chen, Zhi-Hua Zhou, and Fuyan Zhang. Feature selection for high dimensional face image using self-organizing maps. In *PAKDD*, pages 500–504, 2005.

[Tibshirani *et al.*, 2005] R. Tibshirani, G. Walther, D. Botstein, and P. Brown. Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14:511–528, 2005.

[Tombros *et al.*, 2002] Anastasios Tombros, Robert Villa, and Cornelis J. van Rijsbergen. The effectiveness of query-specific hierarchic clustering in information retrieval. *Information Processing and Management*, 38(4):559–582, 2002.

[Tou and Gonzalez, 1977] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, Reading, MA, 1977.

[Truong and Venkatesh, 2007] Ba Tu Truong and Svetha Venkatesh. Video abstraction: A systematic review and classification. *ACM Trans. Multimedia Comput. Commun. Appl.*, 3(1):1–37, 2007.

[Ueda *et al.*, 1991] Hirotada Ueda, Takafumi Miyatake, and Satoshi Yoshizawa. Impact: an interactive natural-motion-picture dedicated multimedia authoring system. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 343–350, New York, NY, USA, 1991. ACM Press.

[Uhlmann, 1991] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[Van Rijsbergen, 1979] Cornelis J. Van Rijsbergen. *Information Retrieval*. Butterworths, London, UK, second edition, 1979.

[Voorhees, 1986] Ellen M. Voorhees. The efficiency of inverted index and cluster searches. In *SIGIR'86, Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, September 8-10*, pages 164–174, 1986.

[Wilbur and Sirotkin, 1992] W. John Wilbur and Karl Sirotkin. The automatic identification of stop words. *J. Inf. Sci.*, 18(1):45–55, 1992.

[W.Johnson and j. Lindenstrauss, 1984] W.Johnson and j. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemp. Math*, 26:189–206, 1984.

[Yang and Pedersen, 1997] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, US, 1997.

[Yang and Wilbur, 1996] Yiming Yang and John Wilbur. Using corpus statistics to remove redundant words in text categorization. *J. Am. Soc. Inf. Sci.*, 47(5):357–369, 1996.

[Yeung and Yeo, 1996] M. M. Yeung and B-L. Yeo. Time-constrained clustering for segmentation of video into story unites. In *ICPR '96: Proceedings of the International Conference on Pattern Recognition (ICPR '96)*, pages 375–380, Washington, DC, USA, 1996. IEEE Computer Society.

[Yianilos, 1999] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search, 1999.

[Zamir and Etzioni, 1998] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 46–54, Melbourne, AU, 1998.

[Zamir *et al.*, 1997] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of Web documents. In *Proceedings of KDD-97, 3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290, Newport Beach, US, 1997.

[Zeng *et al.*, 2004] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma. Learning to cluster Web search results. In *Proceedings of SIGIR-04, 27th ACM International Conference on Research and Development in Information Retrieval*, pages 210–217, Sheffield, UK, 2004.

[Zhang and Couloigner, 2005] Qiaoping Zhang and Isabelle Couloigner. A new and efficient k-medoid algorithm for spatial clustering. In *ICCSA (3)*, pages 181–189, 2005.

[Zhang and Dong, 2004] D. Zhang and Y. Dong. Semantic, hierarchical, online clustering of Web search results. In *Proceedings of APWEB-04, 6th Asia-Pacific Web Conference*, pages 69–78, Hangzhou, CN, 2004.

[Zhang *et al.*, 1996] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 103–114. ACM Press, 1996.

[Zhuang *et al.*, 1998] Yueling Zhuang, Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *IEEE International Conference on Image Processing*, pages 866–870, 1998.