Università
Ca'Foscari
Venezia

MSc (*ex D.M. 270/2004*)
in Computer Science

Thesis

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Transductional Dominant-Set Clustering using Euler Kernels

An experimental study

Supervisor
Prof. Marcello Pelillo

Candidate
Marco Signori
Id 811848

Academic Year
2012 / 2013

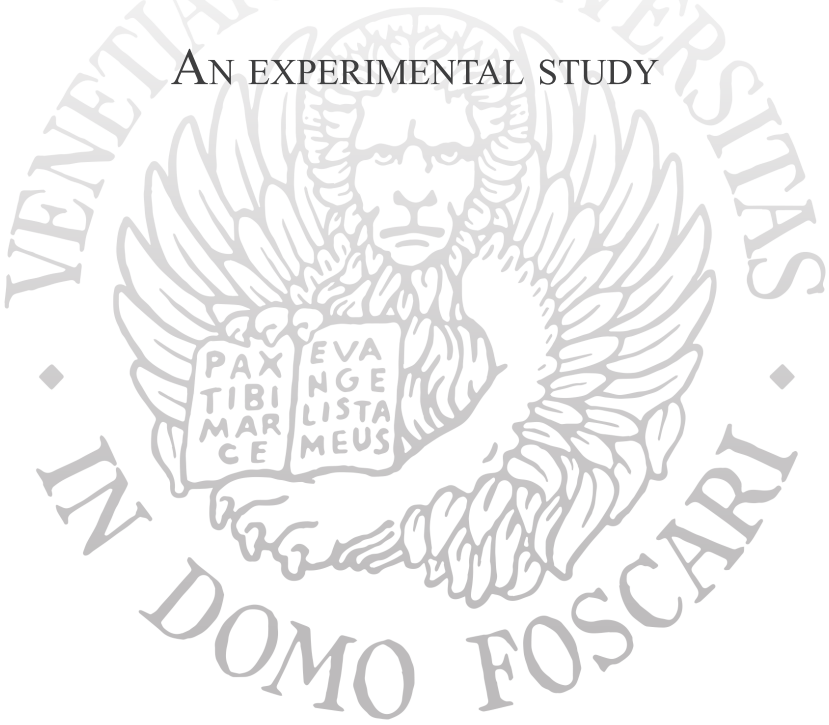
Ca' Foscari University

Faculty of Mathematical, Physical and Natural Sciences

MSc in Computer Science

Transductional Dominant-Set Clustering using Euler Kernels

An experimental study



Candidate

Marco Signori
ID 811848

Supervisor

Prof. Marcello Pelillo

Academic Year 2012/2013

Contents

Abstract

This thesis is about a study of the behavior of the Dominant-Set clustering (DS) using a measure of similarity that derives from the Euler Kernel (Euler-Gauss DS), a Kernel which relies on a nonlinear and robust cosine metric that is less sensitive to outliers. Moreover, in order to create a partitional clustering we use graph tranduction to propagate the membership information from the dominant sets to unlabeled data. We perform an extensive experimental evaluation, using both synthetic and real-world datasets, in order to compare Euler-Gauss DS with the DS algorithm using the classic Gaussian Kernels. Furthermore, we compare Euler-Gauss DS with other clustering algorithms, among which another method that relies on the Euler Kernel (Euler k-means).

Chapter 1

Introdution

Machine learning is the part of Artificial Intelligence that concerns the costruction and the study of systems that are able to learn from data. Machine Learning methods can be divided into three main categories: supervised learning, unsupervised learning and semi-supervised learning methods. Supervised learning methods use a training set, i.e. a set of objects of a chosen domain espressed as a list of features and a label that indicates the membership of the objects to a particular subcategory of the domain, to create a model that can be used to predict the label of objects not in the training set by their features. Semi-supervised learning uses a training set of both labeled and unlabeled data to improve the model. Unsupervised learning studies how systems can extract information about the structure of data without any a priori knowledge about it: Unsupervised learning algorithms manage only unlabeled data.

1.1 The clustering problem

Clustering is the most important unsupervised learning problem, and cluster analysis is used in a variety of scientific disciplines such as biology, psychology, medicine, marketing, computer vision and many others. Organizing data into coherent groupings is indeed one of the most fundamental modes of understanding and learning.
Given a set of objects, clustering analysis is the task to find information of their structure by generating clusters from this set: a cluster is a subset di objects in which its members are more similar to each other (in a certain sense) than those outside it. Everitt [8] documents the following definitions of a cluster:

1. A cluster is a set of entities which are alike, and entities from different clusters are not alike.

2. A cluster is an aggregation of points in the test space such that the

distance between any two points in the cluster is less than the distance between any point in the cluster and any point not in it.

3. Clusters may be described as connected regions of a multi-dimensional space containing a relatively low density of points.

The first one is similar to the one we give to the beginning: it gives to the notion of cluster a qualitative meaning without any assumption of the representation of the data. The other two definitions assume that objects to be clustered are represented as points in the measurement space; in particular, the former gives a distance-based definition of clusters while the latter gives a density-based one. We can note that if we assume the alikeness between two objects as a function that assumes values inversely proportional to the distance between the two respective points in some space the first definition becomes the second.

1.2 Clustering approaches

The need of solving very differents clustering problems in different fields has leaded up to the implementation of many different clustering algorithms that differs primarly in what they mean as a cluster and the way in which they represent the data. We now list the principal properties that distiguish them.

Partitional and Hierarchical approaches

Hierarchical clustering algorithm create a nested sequence of partitions of the data whereas the partitional approach create only a single partition. Hierarchical techniques can be agglomerative if they start putting each object in a different cluster and then they gradually merge them into larger clusters until all objects are in a single big cluster. By converse, if they start with an all-inclusive cluster that was subdivided into smaller and smaller groups until each set contains only one object, they are called divisive hierarchical clustering methods. The hierarchical approach permits a multiscale interpretation of the data.

Exclusive and nonexclusive approaches

An exclusive clustering algorithm assigns each object to only a cluster and so generate a partition of the data. The nonexclusive approach can assign the same object to several clusters: an example of technique that is nonexclusive is the fuzzy clustering (see [16]), a method that assigns to each object a degree of belonginess to each clusters created. There are even clustering algorithms that can consider some objects as noise and do not assign them to any cluster: it is the case of DBSCAN (see [7]) and the other density

based methods derived by it like OPTICS (see [1]).

Data Representation

Different clustering algorithms are designed to manage different types of data and different representations of them.
The most used approach is to project each object in some real normed vector space (the feature space): the data assumes the form of a np matrix where n is the number of objects and p is the number of features. This representation permits to easily visualize the data but it requires a wise feature selection to permit good results in an efficient way.
A second type of representation of the data is called pairwise representation (see [14]) and consists, given a set of n objects, in a n^2 real valued matrix $\mathbf{A}$ where A_{ij} represents the similarity between objects i and j. This form permits to violate metric properties such as the triangular inequality and to use asymmetric measures of similarity. An example of clustering technique that manage such type of data is the Normalized Cut Algorithm (see [31]).

1.3 Dominant-Set Clustering

In this thesis we focus on the Dominant-Set Clustering algorithm introduced by Pavan and Pelillo in 2007 (see [26]), a pairwise clustering technique that permits to extract iteratively groups of similar objects called dominant sets using tools provided by Evolutionary Game Theory. The notion of dominant set is a generalization of maximal clique in graph theory and under some assumptions has a corrispective concept in the game theoretic domain: the Evolutionary Stable Strategy. This method can manage nonsymmetric measures of similarity or that assume negative values and it is even able to create overlapped clusters (i.e. subsets of objects whose intersection is not empty). In the third chapter we will give a deeper look to this Clustering algorithm and we will explain the connection between dominant sets and ESS.

1.4 Using Graph Transduction to assigning unlabeled Data

The Dominant-Set Clustering Algorithm does not necessary label all the objects in a dataset: if we want to partion it (i.e. we know that there isn't noise and so all points have to belong to a cluster) the classic way is to assign each unlabeled object to the same cluster of the nearest labeled object. In order to achieve this job in this thesis we propose a different method: the propagation of the label information from the dominant sets to the unlabeled data using a graph transduction technique that arises from the same game

theoretic background of Dominant Sets. It was introducted by Erdem and Pelillo in 2012 (see [6]) as a semi-supervised learning algorithm. In our implementation the part of the "supervisor" is done by the DS-Clustering Algorithm that extracts the "cores" of the clusters and so it becomes part of an unsupervised learning technique. In the third chapter we will give a deeper look to this method and how can be implemented in our case.

1.5 The Euler kernel

As said above, Dominant-Set clustering is a pairwise clustering technique and so it does not manage objects explicitely defined by their features but only a squared similarity matrix. If our initial dataset is structured as in the former case we have to find a some kind of function that, given two objects of the dataset, it returns a measure of similarity between them. There are other techniques that, to manage n objects, use a n^2 matrix of pairwise comparisons instead of explicit projections of the data in a vector space: they are called kernel methods and permit the implicit projection of the data in different vector spaces using different pairwise functions called kernels: in this way the same algorithm can manage data in different spaces changing only the Kernel. If such Kernel is even a measure of similarity between two objects in a certain domain then it can be used to compute the similarity matrix for a pairwise clustering algorithm as Dominant-Set.

The second contribution of this thesis is the study of the Euler Kernel: a particular Kernel arisen from a cosine-based dissimilarity measure introducted in [10] and used in [20] and [34] to create respectively more robust versions of Principal Component Analysis (PCA) and K-means algorithm. In both cases its utilization has improved in a significative way the performances of the methods and this fact suggests that it can enhance even other clustering methods as Dominant-Set algorithm. We will see in chapter 5 that it cannot be used directly as a measure of similarity but it has to be firstly combined with other Kernels.

1.6 Goal of the thesis

To sum up, in this work we will study the performances of the Dominant-Set Clustering Algorithm:

- using as similarity measure a Kernel derived from the Euler Kernel instead of the classic Gaussian Kernel.
- using graph transduction to label the unclustered data instead of assigning them to the nearest cluster.

Moreover we try to combine Euler Kernels and graph transduction in order to check if their simultaneous use achieves improvements in clustering. We

scheme the steps of our implementation of partitional clustering algorithm in fig 1.1.

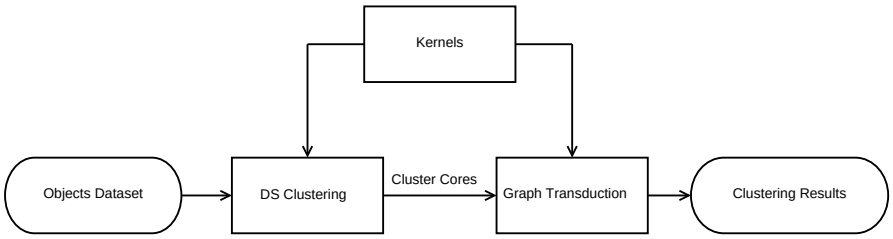


Figure 1.1: The scheme of the partitional clustering proposed in this thesis

1.7 Structure of the thesis

The rest of the thesis is structured as follows: in the second chapter we give an introdution of the Evolutionary Game Theory useful to better understand the following parts. In the second chapter we explain the notion of Dominant Set, its affiliation with the Game Theory and how to use it to extract clusters from a dataset. The third chapter is about the graph transduction algorithm and its implementation in order to assign the points that are not labeled by the Dominant-Set algorithm. In the fourth part we give a brief introduction of kernel methods and their properties. In the fifth chapter we focus on the Euler Kernel and show how we can implement it as a measure of similarity. Finally in the sixth chapter we show the results of several experiments and in the last chapter we give some conclusions about the work done.

Chapter 2

The Game Theoretic Framework

In this chapter we introduce the theoretic and mathematical concepts that will be used in the implementation of the Dominant-Set Clustering and Graph Transduction techniques. Both algorithms are indeed based on some important notions obtained by Evolutionary Game Theory, the branch of study that analyzes games where all the players are biologically or socially conditioned (i.e. pre-programmed) to some behavior, and a selection mechanism operates over time changing the distribution of behaviors of the population. This field differs from the classical game theory because the players in the former are not rational and do not have a complete knowledge of the game. For a deeper look into this fast-growing field see [33].

The chapter is structured as follow: section 2.1 defines the structure of finite games in normal form; section 2.2 considers the notions of Best Reply and Nash Equilibrium; section 2.3 explain the more restrictive concept of Evolutionary Stable Strategy; the final section introduces the Replicator Dynamics, a category of evolutionary dynamics used to model an evolutionary selection mechanism.

2.1 Finite Games in normal form

Let $I = \{1, 2, ..., n\}$ be the set of players and let $S_i = \{1, 2, ..., m_i\}$ (where $m \geq 2$ and finite) be the set of pure strategies available for player $i \in I$. The vector $\mathbf{s} = (s_1, s_2, ..., s_n)$, where s_i is a pure strategy of player i ($s_i \in S_i$), is called a pure-strategy profile. The cartesian product $S = \times_i S_i$ is the set of all possible pure-strategy profiles and it is called the pure-strategy space of the game. Moreover, let $\pi_i(\mathbf{s}) \in \mathbb{R}$ be the payoff associated to player i when strategy profile $\mathbf{s}$ is played: its meaning depends by the context in which the game is modeled, as a title of example, in economics it usually represents the firms' profits or the consumers' utility, while in biology it

represents individual fitness. $\pi_i : S \to \mathbb{R}$ for each player $i \in I$ is called the payoff function for player i; $\pi : S \to \mathbb{R}^n$ is the combined payoff function, it assigns to each pure-strategy profile $s \in S$ the corresponding vector $\pi(\mathbf{s}) = (\pi_1(\mathbf{s}), \pi_2(\mathbf{s}), ..., \pi_n(\mathbf{s}))$ of payoffs.

Given these premises we can now define the notion of *finite game in normal form* as a triplet $G = (I, S, \pi)$ where I is the player set, S its finite pure-strategy space (indeed each player has only a finite number of avaiable strategies) and π its combined payoff function. If $|I| = 2$ the game is denoted as a two-player game: in this case the payoff functions π_1 and π_2 can be represented as two $m_1 m_2$ matrices $\mathbf{A} = (a_{hk})$ and $\mathbf{B} = (b_{hk})$ where $a_{hk} = \pi_1(h, k)$ and $b_{hk} = \pi_2(h, k)$ for each $h \in S_1$ and $b \in S_2$. In this setting the first player is called *row player* and the second one *column player*.
A two-player game where $\mathbf{A} = \mathbf{B}^T$ is denoted as a Symmetric Game: in this class of games the players have the same set of $m = m_1 = m_2$ strategies; if in addition $\mathbf{A}$ is symmetric (i.e. $\mathbf{A} = \mathbf{A}^T = \mathbf{B}$) then it is called Double Symmetric Game.

2.1.1 Mixed Strategy Spaces

Let $G = (I, S, \pi)$ be a finite game, we define the mixed strategy for player i as a probability distribution over the set S_i of pure strategies. It can be represented as an m_i-dimensional real vector $\mathbf{x}_i \in \Delta_i$ where

$$\Delta_i = \{\mathbf{x}_i \in \mathbb{R}_+^{m_i} : \sum_{h=1}^{m_i} x_{ih} = 1\} \tag{2.1}$$

is the $(m_i - 1)$-dimensional standard simplex in the m_i-space.

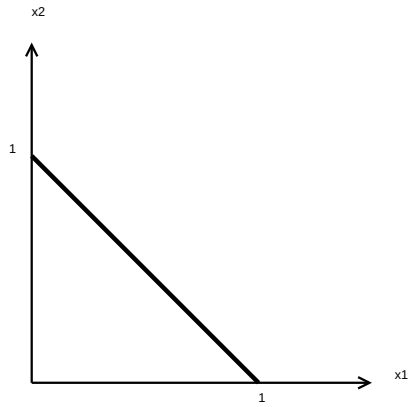


Figure 2.1: Unidimensional Simplex

Each vertex of the simplex Δ_i corresponds to a unit vector $\mathbf{e}_i^h$ (with $h = 1, ..., m_i$) in m_i-space and it represents the mixed strategy for player

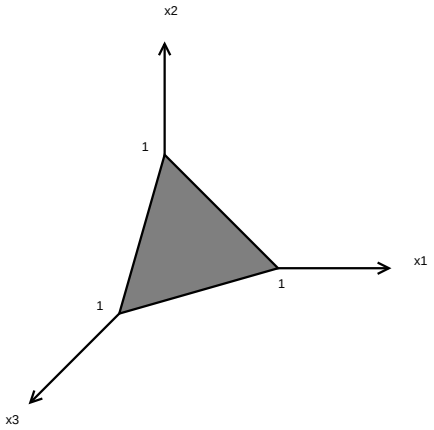


Figure 2.2: 2-dimensional simplex

i that chooses always the strategy h. An arbitrary $\mathbf{x}_i \in \Delta_i$ is a convex combination of the unit vectors:

$$\mathbf{x}_i = \sum_{h=1}^{m_i} x_{ih} \mathbf{e}_i^h \tag{2.2}$$

If $\mathbf{x}_i$ is a linear combination of only a subset of unit vectors (i.e. player i uses not all the avaiable strategies) then it is located in a face of the simplex and so it is a boundary point of Δ_i. Otherwise if $\mathbf{x}_i$ is a linear combination of all the unit vectors (i.e. player i uses all the avaiable strategies) it is located in the interior of the simplex and it is called *completely mixed strategy*. In order to denote which pure strategies a player can use given her mixed strategy $\mathbf{x} \in \Delta$ we define the support of $\mathbf{x}$ as

$$\sigma(\mathbf{x}) = \{h \in S : \mathbf{x}_h \neq 0\} \tag{2.3}$$

The vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ where $\mathbf{x}_i$ is the mixed strategy of player $i \in I$ is called a mixed strategy profile and it is a point of the mixed-strategy space

$$\Theta = \times_{i \in I} \Delta_i \tag{2.4}$$

that is a $(m - n)$-dimensional polyhedron in $\mathbb{R}^m$ where $m = m_1 + m_2 + ... + m_n$. In figure 2.3 and 2.4 we can see the projections of the polyhedrons corrispondent to the mixed strategy spaces of two-player games where in the former the players have the same number of strategies while in the latter the first player has one more. In the next sections we will denote with $(\mathbf{x}_i, \mathbf{y}_{-i})$ the mixed strategy profile in which players play according to $\mathbf{y} \in \Theta$ except i that plays $\mathbf{x}_i \in \Delta_i$. In the case of two-player games it will be denoted simply with $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ and $\mathbf{y}$ are respectively the mixed strategies of player 1 and player 2.

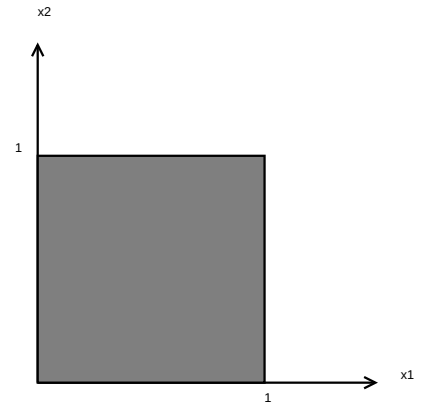


Figure 2.3: Projection of the polyhedron Θ when $n = m_1 = m_2 = 2$.

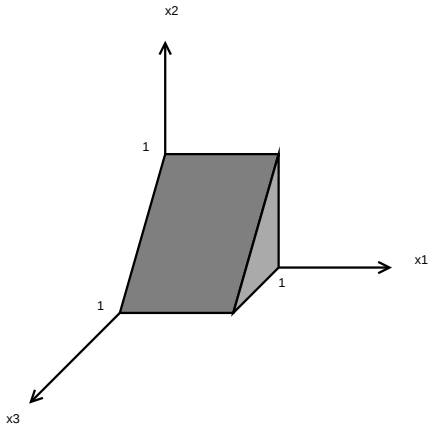


Figure 2.4: Projection of the polyhedron Θ when $n = 2$, $m_1 = 3$ and $m_2 = 2$.

2.1.2 Mixed-Strategy Payoff functions

We assume that all players's randomization are statistically independent (i.e. the i player's chosen strategy does not depend from the other player's chosen strategies). In this setup we can compute the probability that a specific pure strategy profile $\mathbf{s} = (s_1, s_2, ...s_n)$ is used when players play a mixed-strategy profile $\mathbf{x} \in \Theta$ as follows:

$$x(\mathbf{s}) = \prod_{i=1}^{n} x_{is_i}. \tag{2.5}$$

Once we have the probabilities of all the pure-strategy profiles we can evaluate the expected value of the payoff obtained by a chosen player when is played the profile $\mathbf{x}$:

$$u_i(\mathbf{x}) = \sum_{\mathbf{s}\in S} x(\mathbf{s})\pi_i(\mathbf{s}) \tag{2.6}$$

called the ith player payoff from strategy profile $\mathbf{x}$. In the case of two-player game (with the respective payoffs matrices $\mathbf{A}$ and $\mathbf{B}$ as seen above) we have

$$u_1(\mathbf{x}) = \sum_{h=1}^{m_1}\sum_{k=1}^{m_2} x_{1h}a_{hk}x_{2k} = \mathbf{x}_1\mathbf{A}\mathbf{x}_2 \tag{2.7}$$

$$u_2(\mathbf{x}) = \sum_{h=1}^{m_1}\sum_{k=1}^{m_2} x_{1h}b_{hk}x_{2k} = \mathbf{x}_1\mathbf{B}\mathbf{x}_2 = \mathbf{x}_2\mathbf{B}^T\mathbf{x}_1 \tag{2.8}$$

In a symmetric game the computation of the second player payoff can be performed using the payoff matrix of the first player:

$$u_2(\mathbf{x}) = \mathbf{x}_2\mathbf{B}^T\mathbf{x}_1 = \mathbf{x}_2\mathbf{A}\mathbf{x}_1 \tag{2.9}$$

Moreover, if the game is double-symmetric we have

$$u_2(\mathbf{x}) = \mathbf{x}_2 A\mathbf{x}_1 = u_1(\mathbf{x}) \tag{2.10}$$

so in this special case the two players earns always the same payoff. In a two-player game the operator u can be intended as a bilinear operator $u(\mathbf{x}_1, \mathbf{x}_2)$ indeed it is easy to prove that

$$\begin{aligned} u(c\mathbf{x}_1 + \mathbf{y}, \mathbf{x}_2) &= c(u(\mathbf{x}_1, \mathbf{x}_2) + u(\mathbf{y}, \mathbf{x}_2)) \\ u(\mathbf{x}_1, c\mathbf{x}_2 + \mathbf{y}) &= c(u(\mathbf{x}_1, \mathbf{x}_2) + u(\mathbf{x}_1, \mathbf{y})) \end{aligned} \tag{2.11}$$

for each $c \in \mathbb{R}$ and $\mathbf{y} \in \mathbb{R}^K$.

2.2 Best Replies and Nash Equilibrium

2.2.1 Best Replies

The pure strategy $h \in S_i$ that gives to player i the higher payoff when other players play the strategy profile $\mathbf{y}$ is called pure best reply for player i against $\mathbf{y}$. Formally for each $k \in S_i$, h is a pure best reply if

$$u_i(\mathbf{e}_i^h, \mathbf{y}_{-i}) \geq u_i(\mathbf{e}_i^k, \mathbf{y}_{-i}). \tag{2.12}$$

This link between strategy profiles and relative best replies for player i defines the ith-player pure-strategy best-reply corrispondence $\beta_i : \Theta \to S_i$:

$$\beta_i(\mathbf{y}) = \{h \in S_i : u_i(\mathbf{e}_i^h, \mathbf{y}_{-i}) \geq u_i(\mathbf{e}_i^k, \mathbf{y}_{-i}) \forall k \in S_i\} \tag{2.13}$$

There may be more than a pure strategy that are best replies to $\mathbf{y}$: in this case all the mixed strategies that are a linear combination of a subset of them (and so they are located in a face of the simplex) are best replies to $\mathbf{y}$ too. In figure 2.5 we can see that in the case of a two-dimensional simplex, if the pure best replies are x_1 and x_2 all the mixed strategies in the face between them are best replies too. The mixed-strategy best-reply corrispondence $\tilde{\beta}_i : \Theta \to \Delta_i$ defines the subset of mixed strategies that are best replies for $\mathbf{y}$:

$$\begin{aligned} \tilde{\beta}_i(\mathbf{y}) &= \{\mathbf{x}_i \in \Delta_i : u_i(\mathbf{x}_i, \mathbf{y}_{-i}) \geq u_i(\mathbf{z}_i, \mathbf{y}_{-i}) \forall \mathbf{z}_i \in \Delta_i\} = \\ &\quad \{\mathbf{x}_i \in \Delta_i : x_{ih} = 0 \forall h \notin \beta_i(\mathbf{y})\} = \\ &\quad \{\mathbf{x}_i \in \Delta_i : \sigma(\mathbf{x}_i) \subseteq \beta_i(\mathbf{y})\} \end{aligned} \tag{2.14}$$

The cartesian product of all players' $\tilde{\beta}_i$ is the combined mixed strategy best-reply corrispondence:

$$\tilde{\beta}(\mathbf{y}) = \times_{i \in I} \tilde{\beta}_i(\mathbf{y}) \subset \Theta. \tag{2.15}$$

The combined pure-strategy best-reply corrispondence is

$$\beta(\mathbf{y}) = \times_{i \in I} \beta_i(\mathbf{y}) \subset S. \tag{2.16}$$

2.2.2 Nash Equilibrium

The notion of Nash Equilibrium was introduced by John Nash in 1950 (see [25]) and it is one of the most important concepts in Game Theory. There is a Nash Equilibrium when players use a strategy profile $\mathbf{z} \in \Theta$ in which each component strategy $\mathbf{z}_i$ is optimal against $\mathbf{z}$ or, in other words, if $\mathbf{z}$ is the best reply of itself:

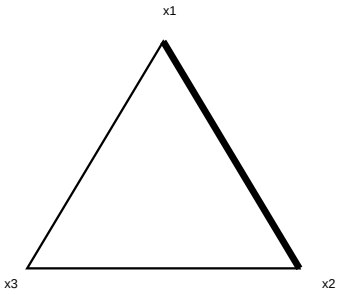


Figure 2.5: Best reply mixed strategy set when pure best reply are x_1 and x_2.

Definition 2.1. (Nash Equilibrium)
A strategy profile $\mathbf{z} \in \Theta$ is a Nash Equilibrium if $\mathbf{z} \in \tilde{\beta}(x)$ that is equivalent to

$$u_i(\mathbf{z}_i, \mathbf{z}_{-i}) \geq u_i(\mathbf{x}_i, \mathbf{z}_{-i}) \tag{2.17}$$

for each $i \in I$ and $\mathbf{x}_i \in \Delta_i$.
If strict inequality holds for all $\mathbf{x}_i \in \Delta_i$ then is said to be a Strict Nash Equilibrium, in this case $\tilde{\beta}(\mathbf{z}) = \{\mathbf{z}\}$.

It is easy to see that all the elements in the support of each component $\mathbf{z}_i$ is a best reply to $\mathbf{z}$. In his paper Nash proved the following theorem:

Theorem 2.1.
For any finite game G its set of Nash Equilibria Θ^{NE} is always nonempty.

Proof: See [25].

□

In the special case of a two-player Symmetric Game, the mixed strategy profile $(\mathbf{x}, \mathbf{y}) \in \Theta = \Delta^2$ is a Nash Equilibrium if and only if $(\mathbf{x}, \mathbf{y}) \in \tilde{\beta}(\mathbf{x}, \mathbf{y})$ and it is called symmetric if $\mathbf{x} = \mathbf{y}$: the set of such strategies is Δ^{NE}. Not all Nash Equilibria in a symmetric game are symmetric but every symmetric game has at least a symmetric Nash Equilibrium, indeed the following assertion can be proved:

Proposition 2.2.
For any finite symmetric game G its set of symmetric Nash Equilibria Δ^{NE} is always nonempty.

Proof: See [33].

□

2.3 Evolutionary Stable Strategies

From this section we focuses only in two-player symmetric games. The Evolutionary Stable Strategy (ESS) is another key notion in evolutionary game theory and it was defined by Maynard Smith and Price in 1973 (see [23]): an ESS denotes a mixed strategy that is robust to evolutionary pressures (i.e it is resistent to invasion by new strategies). We consider the situation in which a population playing the same incumbent strategy $\mathbf{x}$ is invaded by a small group of individuals that plays the mutant strategy $\mathbf{y} \in \Delta$. If the share of the invaders in the post-entry population is $\epsilon \in (0, 1)$ the payoff in a match in this bimorphic population is equal to the payoff in a match with an individual playing the mixed strategy

$$\mathbf{w} = \epsilon\mathbf{y} + (1 - \epsilon)\mathbf{x} \in \Delta \tag{2.18}$$

so the payoff after invasion of the two populations is respectively $u(\mathbf{x}, \mathbf{w})$ and $u(\mathbf{y}, \mathbf{w})$. Under these assumptions we can give a formal definition of ESS:

Definition 2.2 (Evolutionary Stable Strategy)**.**
A strategy $\mathbf{x} \in \Delta$ is an evolutionary stable strategy if for all $\mathbf{y} \in \Delta - \{\mathbf{x}\}$ there exists $\iota \in (0, 1)$, such that for al $\epsilon \in (0, \iota)$ we have:

$$u(\mathbf{x}, \epsilon\mathbf{y} + (1 - \epsilon)\mathbf{x}) > u(\mathbf{y}, \epsilon\mathbf{y} + (1 - \epsilon)\mathbf{x}) \tag{2.19}$$

By this definition we can infer that a Evolutionary Stable Strategy $\mathbf{x}$ is always optimal against itself, otherwise there would be a strategy $\mathbf{y}$ obtaining a higher payoff against $\mathbf{x}$ and so the condition under which a strategy can be denoted as evolutionary stable falls. This fact means that the set of Evolutionary Stable Strategies is a subset of the symmetric Nash Equilibria:

$$\Delta^{ESS} \subset \Delta^{NE} \tag{2.20}$$

The inverse is not true, ESS is a more restrictive concept and requires not only the optimality of $\mathbf{x}$ against itself but, given an alternative best reply $\mathbf{y}$, in order to be an ESS $\mathbf{x}$ has to be a better reply to $\mathbf{y}$ than $\mathbf{y}$ itself:

Proposition 2.3. *In order to be an ESS a mixed strategy $\boldsymbol{x} \in \Delta$ has to meet the following first-order and second-order best-reply conditions:*

$$u(\boldsymbol{y}, \boldsymbol{x}) \leq u(\boldsymbol{x}, \boldsymbol{x}) \qquad \forall \boldsymbol{y} \in \Delta, \tag{2.21}$$

$$u(\boldsymbol{y}, \boldsymbol{x}) = u(\boldsymbol{x}, \boldsymbol{x}) \Rightarrow u(\boldsymbol{y}, \boldsymbol{y}) < u(\boldsymbol{x}, \boldsymbol{y}) \quad \forall \boldsymbol{y} \neq \boldsymbol{x} \tag{2.22}$$

If $\mathbf{x}$ is a symmetric Strict Nash Equilibrium it is even evolutionary stable, indeed in this case there are not alternative best replies.

2.4 Replicator Dynamics

The replicator dynamics process is a specific type of evolutionary dynamic introduced by Taylor and Jonker in 1978 (see [32]) in order to model the evolution of behavior in animal conflicts and, afterwards, it was used succesfully in various fields not directly correlated with biology like economics. A replicator, as the name may suggest, is an entity that can make approximately accurate copies of itself in some ways: real entities that can be seen as replicators are, for example, genes, believes, conventions, and other behaviour phenotipes: in a game theoretic context they are pure strategies.
An evolutionary dynamic of a replicator system is a process of change over time in the frequency distribution of the replicators in which higher payoff strategies reproduces faster than the lower ones. Such dynamics assume that a population of individuals of the same species compete for a specific limited resource: the competition is modelled as a symmetric two-player game played over and over by two individuals, each time randomly chosen from the population, that don't act rationally but according to a pre-programmed behavioural pattern, or pure strategy. Reproduction is assumed to be asexual, hence the offspring will inherit the same genetic material (and even the same behavior) as its parent without any mutation.
Let $J = \{1, ..., n\}$ be the set of avaiable pure strategies and let $x_i(t)$ with $i \in J$ be the ratio of the population that plays strategy i at time t (so at each t the complessive state of the population is given by the vector $\mathbf{x} \in \Delta$ of $\mathbb{R}^n$). Then:

$$\dot{x_i} = x_i(u(\mathbf{e}^i, \mathbf{x}) - u(\mathbf{x}, \mathbf{x})) \tag{2.23}$$

where a dot signifies derivative with respect to time, $u(\mathbf{e}^i, \mathbf{x}) = \sum_j x_j \pi(i, j)$ is the fitness of type i (in this case it is the payoff of strategy i) and $u(\mathbf{x}, \mathbf{x}) = \sum_i \sum_j x_i x_j \pi(i, j)$ is the total weighted payoff and $\pi(i, j)$ is the payoff of strategy i against strategy j. This set of differential equations is called replicator dynamics.
If we set a matrix $\mathbf{A} \in \mathbb{R}^{nn}$ where $a_{ij} = \pi(i, j)$ the equation can be rewritten in this way:

$$\dot{x_i} = x_i((A\mathbf{x})_i - \mathbf{x}^T A\mathbf{x}) \tag{2.24}$$

It can be seen that in the replicator dynamics the rate of the strategies with payoff greater than $u(\mathbf{x}, \mathbf{x})$ increases and, by converse, it decreases if its strategy payoff is lesser than $u(\mathbf{x}, \mathbf{x})$. The subset of population that play pure best replies to the current population state are the ones with the highest growth rate. Another important property is that if we sum up all the equations, we get $\sum_i \dot{x_i} = 0$ so, if $\mathbf{x}(0) \in \Delta$, the standard simplex Δ is invariant under replicator dynamics. Hence $\mathbf{x}$ follows a trajectory over time in the simplex depending by its initial value and the payoff matrix $\mathbf{A}$.

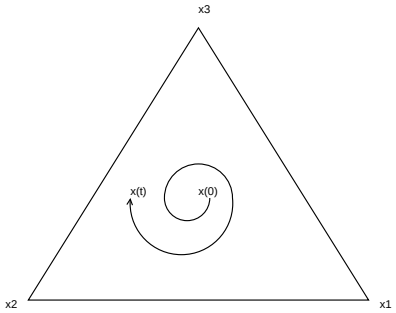


Figure 2.6: A possible trajectory of replicator dynamics in the simplex.

Another important property of replicator dynamics is its invariance under positive affine transformations of payoffs. Given a payoff function u we can substitute it with $\hat{u} = \lambda u + \mu$ where $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$: the Replicator Dynamics differential equation system becomes

$$\dot{x_i} = x_i(\hat{u}(\mathbf{e}^i, \mathbf{x}) - u(\mathbf{x}, \mathbf{x})) = x_i(\lambda u(\mathbf{e}^i, \mathbf{x}) - u(\mathbf{x}, \mathbf{x})). \tag{2.25}$$

Such tranformations of payoffs change the time scale by a factor $\lambda > 0$ i.e. the solution orbits remain the same but the velocity of the modifications of the population state changes; moreover, we can see that the parameter μ does not affect at all the dynamic system.

2.4.1 Limit points and Nash Equilibria

The replicator dynamic does not always converge, its trajectory obviously depends by its starting point $\mathbf{x}^{(0)}$ in the simplex and the payoff matrix $\mathbf{A}$. From definition a point $\mathbf{x}$ in the simplex is a stationary point for a replicator dynamics if and only if all the not extincted (i.e. belonging to the support $\sigma(\mathbf{x})$) pure strategies in $\mathbf{x}$ have the same payoff: we can formally define the set of stationary states $\Delta^o \in \Delta$ as

$$\Delta^o = \{\mathbf{x} \in \Delta : u(\mathbf{e}^i, \mathbf{x}) = u(\mathbf{x}, \mathbf{x}) \ \forall i \in \sigma(\mathbf{x})\}. \tag{2.26}$$

It is easy to see that Δ^{NE} is a subset of Δ^o: the opposite is not true because even all the vertices of the simplex belong to Δ^o, if we focus only in the set of stationary points in the interior of the simplex $\Delta^{oo} = \Delta^o \cap \text{int}(\Delta)$ they all belong to Δ^{NE}, moreover the Δ^{oo} is equal to the set of interior Nash equilibria: $\Delta^{oo} = \Delta^{NE} \cap \text{int}(\Delta)$.

If a replicator dynamics converge to a point $\mathbf{x}$ and $\mathbf{x}^{(0)} \in \text{Int}(\Delta)$ the following statement due to Nachbar can be proved:

Proposition 2.4.
$\boldsymbol{x}$ is the limit point of a replicator dynamics trajectory starting from the interior of Δ if and only if $\boldsymbol{x} \in \Delta$ is a Nash equilibrium point.

Proof: See [33].

□

Hence every reachable stability state is even a Nash Equilibrium point, the starting point must be in the interior of the simplex because, as said above, all the stability points x_i than may not be Nash Equilibria are in the boundary of Δ and so they are reachable if $x^{(0)} = x_i$: the starting condition assures that the trajectory of the replicator dynamics will always converge in a Nash Equilibrium.

2.4.2 Asymptotic Stability and ESS

In a dynamical system a point x is said asymptotically stable equilibrium point if $\dot{x}_i = 0$ $(i = 1, ..., n)$ and if any trajectory starting in its prossimity will converge to it as $t \to \infty$. In 1979 Hofbauer, Schuster and Sigmund proved the following statement:

Proposition 2.5.
If $\boldsymbol{x} \in \Delta$ is an ESS, then it is an asymptotically stable equilibrium point for the replicator dynamics.

Proof: See [12].

□

The proposition does not guarantee that every equilibrium point for the replicator dynamics is an ESS: the viceversa, as we will see in the next paragraph, is true only for a particular subset of games.

2.4.3 Replicator Dynamics in Doubly Symmetric Games

It can be demostrated that the special case of doubly symmetric games represents a situation in which the fundamental theorem of natural Selection is valid: this theorem initially formulated by Fisher in 1930 (see [9]) states that in some contexts the average fitness of a population monotonically increases during time. The reformulation of the theorem in terms of evolutionary game theory due to Losert and Akin (1985) shows indeed that such contexts are the doubly symmetric games:

Theorem 2.6.
For any doubly symmetric game, the average payoff $f(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$ is strictly increasing along any non-constant trajectory of replicator dynamics, namely, $d/dt f(\boldsymbol{x}(t)) \geq 0$, with equality if and only if $\boldsymbol{x}(t) \in \Delta^o$.

See [33]

□

Even if the average payoffs monotonically increases during time this does not mean that it will always approach its global maximum value: the theorem assures only that $u(\mathbf{x}, \mathbf{x})$ will reach a local maximum depending by its starting point. Hofbauer and Sigmund (1988) show that in the specific field of doubly symmetric games the following statements are equivalent:

1. $\mathbf{x} \in \Delta^{ESS}$.
2. $\mathbf{x} \in \Delta$ is a strict local maximizer of the average payoff $u(\mathbf{x}, \mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$ over the standard simplex Δ.
3. $\mathbf{x} \in \Delta$ is asimptotically stable in the replicator dynamic.

Hence if $\mathbf{A} = \mathbf{A}^T$ every stable point in the replicator dynamics is an Evolutionary Stable Strategy.

2.5 Multi-population Replicator Dynamics

We consider now the situation there are n different populations of (infinite) individuals: let $\mathbf{x}_i \in \Delta_i$ be the state of population i where each element x_{ih} corresponds to the ratio of individuals programmed to play pure strategy h. Over and over an individual from each population is chosen at random to play a n-player game. The replicator dynamics differential equation used to model the selection mechanism in such multi-population case are

$$\dot{x}_{ih} = [u_i(\mathbf{e}_i^h, \mathbf{x}_{-i}) - u_i(\mathbf{x})]x_{ih} \tag{2.27}$$

The next statement is the multi-population version of the propostion 2.4:

Proposition 2.7.
$\boldsymbol{x} \in \Theta$ is the limit point of a multi-population replicator dynamics trajectory starting from the interior of Θ if and only if $\boldsymbol{x}$ is a Nash equilibrium point.

Proof: See [33].

□

Moreover it can be demostrated that the limit point is even asimptotically stable then it is a Strict Nash Equilibrium point.

In this chapter we have shown two of the most important notions in evolutionary game theory: the Evolutionary Stable Strategy and the Nash Equilibrium: the former is even the key concept of the Dominant-Set Clustering (see chapter 3) while the latter is the key concept of the game theoretic graph transduction (see chapter 4). In the following chapters we show how the clustering problem and the graph tranduction can be formulated as evolutionary games and so how they can be solved by using game theoretic tools, as replicator dynamics.

Chapter 3

Dominant-Set Clustering

Dominant-Sets algorithm is a pairwise data clustering technique that iteratively extracts clusters from the data. The particolarity of this graph-theoretic approach is the way in which it defines the notion of cluster:

A cluster is a Dominant Set.

In this chapter we explain what such definition means and how the algorithm works to find clusters. In particular we show how dominant sets can be seen as Evolutionary Stable Strategies in a specific two-player game setting, and hence how they can be extracted using a special version of replicator dynamics.

3.1 The maximum clique problem

The notion of dominant set arises from the generalization of the maximum clique problem for weighted graphs.
Given an unweighted graph $G = (V, E)$ where $V = \{1, ..., n\}$ is the vertex set, $E \subseteq V \times V$ is the edge set, a subset of vertex $C \subseteq V$ is a *clique* if $\forall v_1, v_2 \in C, (v_1, v_2) \in E$.
A *maximal clique* is a clique not contained in a more comprehensive one and a *maximum clique* is the complete subgraph with the highest number of nodes in a graph.

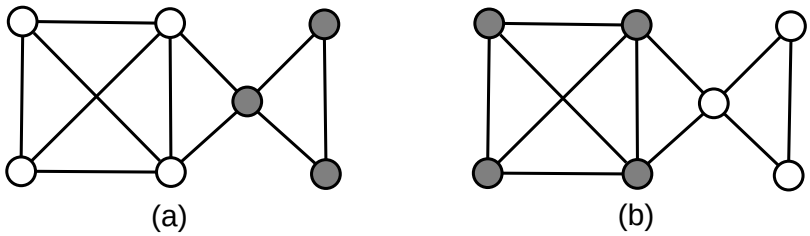


Figure 3.1: In grey, respectively, (a) a maximal clique, (b) a maximum clique

The maximum clique problem consists, given a graph, in finding the maximum clique inside it. This is an NP-complete problem and several algorithms that find maximal cliques have been implemented. In order to give a first connection between graphs and objects to be clustered we can consider the nodes of an unweighted graph as a set of objects, such nodes are connected by arcs if they are similar (in this case the notion of similarity is binary, two objects can be only similar or not similar). If we find a maximal clique we can extract conseguently the corresponding subset of mutually similar objects. If our measure of similarity assumes not binary but continuous values the isomorphism written above does not work anymore: we have to give weights to the arcs, corresponding to the values of similarity and so we have to use weighted graphs.

3.2 Dominant Sets

Before defining the notion of a dominant set we need to give some definitions:
given an edge-weighted (similarity) graph with no self-loops $G = (V, E, \omega)$ where $V = \{1, ..., n\}$ is the vertex set, $E \subseteq V \times V$ is the edge set and $\omega : E \to \mathbb{R}_+^*$ is a positive function of the weights, it can be represented as a similarity matrix $\mathbf{A} = (a_{ij})$ where $a_{ij} = \omega(i, j)$ if $(i, j) \in E$, othewise $a_{ij} = 0$. In this setup, in order to create a correspondent notion of clique in the weighted graphs field we have to define some measures of similarity between nodes and set of nodes.

Definition 3.1 (Average Weighted degree)**.**
Let $S \subseteq V$ be a not-empty vertex set and $i \in S$, the Average Weighted Degree of i w.r.t S is defined as:

$$\text{avgdeg}_S(i) = \frac{1}{|S|} \sum_{j \in S} a_{ij} \tag{3.1}$$

so, by definition, $\text{avgdeg}_{\{i\}}(i) = 0$ for any $i \in S$.
Moreover, given $j \notin S$:

$$\phi_S(i,j) = a_{ij} - \text{awgdeg}_S(i) \tag{3.2}$$

represents the relative similarity between nodes i and j with respect to the average similarity between node i and the nodes in S: it can assume either positive or negative values.

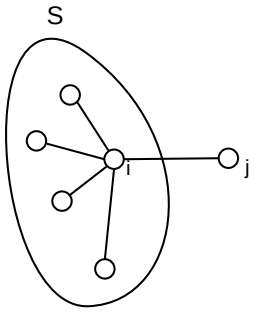


Figure 3.2: A visual interpretation of $\phi_S(i,j)$

Definition 3.2 (Weight of a node)**.**
Let $S \subseteq V$ be a not-empty vertex set and $i \in S$, the weight of i w.r.t S is defined as:

$$w_S(i) = \begin{cases} 1, & \text{if } |S| = 1 \\ \sum_{j \in S\setminus\{i\}} \phi_{S\setminus\{i\}}(j,i) w_{S\setminus\{i\}}(j), & \text{otherwise.} \end{cases} \tag{3.3}$$

$w_s(i)$ represents the relative similarity between node i and the nodes of $S\setminus\{i\}$ with respect to the overall similarity among the nodes in S.

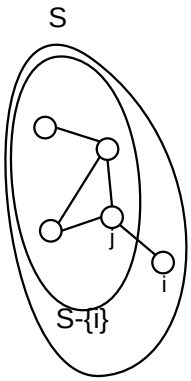


Figure 3.3: A visual interpretation of $w_S(i)$

Definition 3.3 (Total Weight of a Subgraph)**.**
The total weight of S is defined as

$$W(S) = \sum_{i \in S} w_S(i) \tag{3.4}$$

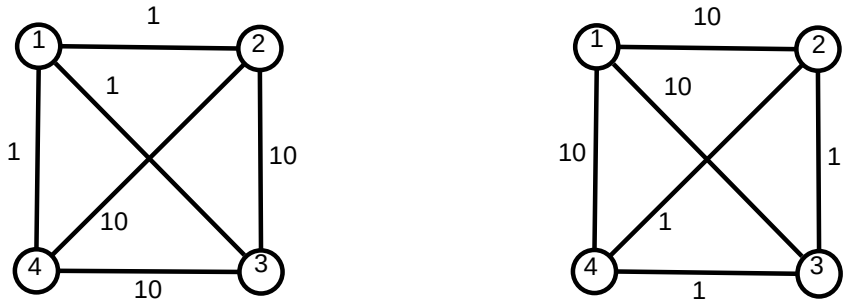


Figure 3.4: Given S={1,2,3,4} we can see that in (a) $w_S(1) < 0$, indeed 1 is poorly correlated with the other nodes in S. Conversely, in (b) $w_S(1) > 0$.

Finally we can define the notion of dominant set:

Definition 3.4 (Dominant Set)**.**
A non-empty vertex set $S \subseteq V$ such as $W(T) > 0$ for any not-empty set $T \subseteq S$ is called dominant if

$$w_S(i) > 0 \quad \forall i \in S \tag{3.5}$$

$$w_{S \cup \{i\}}(i) < 0 \quad \forall i \in S. \tag{3.6}$$

In such way dominant sets have the properties we want in order to consider them as clusters: indeed (3.5) can be seen as the condition of internal homogeneity of a cluster (each $i \in S$ is similar to all other nodes in S) and (3.6) represents, by converse, the condition of external heterogeneity. The subset of node in $\{2, 4, 3\}$ in figure 3.4(a), for example, is a dominant set because the internal edges are larger than those between internal and external nodes: the main property of a dominant set is that the overall similarity among internal vertices is higher than between internal and external vertices and it implies that the structures captured by dominant sets are very compact.

3.3 Dominant Set as a quadratic programming problem

Given a similarity graph $G = (V, E, \omega)$ with n vertices and the correspondenting similarity matrix A, a canonical way to represent a cluster $C \subseteq V$ of nodes is a n-dimensional vector $\mathbf{x}$ associated to it in which $\mathbf{x}(i)$ shows the partecipation of the ith node to cluster C: $x(i) = 0$ means that node i is not a member of C. Using this rapresentation permits the definition of a measure of cluster cohesion as a quadratic function:

$$f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x} \tag{3.7}$$

hence in order to find the most cohese cluster the following quadratic program has to be solved:

$$\begin{aligned} &\text{maximize} \quad f(\mathbf{x}) \\ &\text{subject to} \quad \mathbf{x} \in \Delta \end{aligned} \tag{3.8}$$

where $\Delta = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0} \wedge \mathbf{e}^T\mathbf{x} = 1\}$ is the (n-1)-dimensional standard simplex.
In this section we show that this notion of a cluster and dominant sets are strongly correlated and how this fact permits the formulation of the dominant sets identification problem as a standard quadratic program.

A point $\mathbf{x} \in \Delta$ satisfies the Karush-Khun-Tucker (KKT) conditions (the 1st-order conditions for the local optimality) for the problem (2.8) if exist $n + 1$ real costants (laplacian multipliers) $\mu_1, ..., \mu_n$ and λ with $\mu_i \geq 0$ for all $i = 1, ..., n$ such as

$$\begin{aligned} &(A\mathbf{x})_i - \lambda + \mu_i = 0 \quad \forall i = 1, ..., n \\ &\text{and } \textstyle\sum_{i=1}^{n} \mathbf{x}_i \mu_i = 0 \end{aligned} \tag{3.9}$$

The last constrain of 3.9 implies that if $i \in \sigma(\mathbf{x})$ then $\mu_i = 0$ so the conditions may be rewrited in this way:

$$(A\mathbf{x})_i \begin{cases} = \lambda & \text{if } i \in \sigma(\mathbf{x}) \\ \leq \lambda & \text{otherwise} \end{cases} \tag{3.10}$$

Definition 3.5 (Weighted Characteristic Vector)**.**
a weighted characteristic vector (w.c.v.) $\mathbf{x}_i^S \in \Delta$ of a subset of nodes is

defined as:

$$\mathbf{x}_i^S = \begin{cases} \frac{w_s(i)}{W(S)}, & \text{if } i \in S \\ 0, & \text{otherwise.} \end{cases} \tag{3.11}$$

By definition, a subset S admits a w.c.v., only if $W(S) \neq 0$ so dominant sets always admit it.
In order to connect the weighted characteristic vectors with the KKT condition of program (3.8) the next propositions can be proved :

Proposition 3.1.
Let $\sigma = \sigma(\boldsymbol{x})$ be the support of a vector $\boldsymbol{x} \in \Delta$ which admits weighted characteristic vector $\boldsymbol{x}^\sigma$. Then $\boldsymbol{x}$ satisfies the KKT equality conditions in (3.10) if and only if $\boldsymbol{x} = \boldsymbol{x}^\sigma$. Moreover, in this case:

$$\frac{w_{\sigma\cup\{j\}}(j)}{W(\sigma)} = (A\boldsymbol{x})_j - (A\boldsymbol{x})_i = -\mu_j \tag{3.12}$$

for all $i \in \sigma$ and $j \notin \sigma$ where the μ_js are the nonnegative Lagrange multipliers of program (3.8).

Proof: See [26].

□

Proposition 3.2.
Let $\boldsymbol{x} \in \Delta$ be a vector with support σ, $W(\sigma) > 0$ and weighted characteristic vector $\boldsymbol{x}^\sigma$, then $\boldsymbol{x}$ is a KKT point for program (3.8) if and only if:

1. $\boldsymbol{x} = \boldsymbol{x}^\sigma$
2. $w_{\sigma\cup\{j\}}(j) \leq 0, \forall j \notin \sigma$

Proof: See [26].

□

Finally the next theorem explains the relation between dominant sets and local solutions of program (3.8):

Theorem 3.3.
If S is a dominant subset of nodes, then its weighted characteristic vector $\boldsymbol{x}^S$ is a strict local solution of program (3.8).
Conversely, if $\boldsymbol{x}^*$ is a strict local solution of program (3.8) then its support $\sigma = \sigma(\boldsymbol{x}^*)$ is a dominant set, provided that $w_{\sigma\cup\{i\}}(i) \neq 0 \ \forall i \notin \sigma$.

Proof: See [26].

□

The last costraint avoids the presence of spurious solutions in (3.8), i.e. solutions with support that doesn't admit a weighted characteristic vector. The theorem shows that dominant sets correspond to strict local solutions of the quadratic program. One of the ways to find such solutions is by using replicator dynamics.

3.4 Dominant Sets as Evolutionary Stable Strategies

In the last section we have formulated the problem of finding a dominant set in a quadratic problem: it is easy to see that the function to be maximized is the same that is maximized by replicator dynamics in a doubly symmetric game:

$$f(\mathbf{x}) = u(\mathbf{x}, \mathbf{x}) \tag{3.13}$$

Moreover we have shown that such dynamics implicitely respect the costraint $\mathbf{x} \in \Delta$. We now reformulate the quadratic problem as a two-player symmetric game in normal form where the set of pure strategies is the set of objects (the vertices of the graph) and $\mathbf{A}$ is the payoff matrix. If $\mathbf{A}$ is even symmetric the weighted characteristic vector that locally maximizes $f(\mathbf{x})$ in a game theoretic context is an Evolutionary Stable Strategy and so is a stationary point of the replicator dynamics.

3.4.1 Replicator Dynamics: discrete-time

In order to implement the dominant-set clustering using replicator dynamics one can use numerical methods for the resolution of differential equation systems like Runge-Kutta algorithm. A less computational-expensive algorithm can be obtained by discretizing time. In this setup each time period $t = 1, 2, ...$ represents a generation. If we assume that each offspring inherits its single parent strategy and that there are no overlapping generations (i.e. for each t, the corrisponding $\mathbf{x}(t)$ represent only the state of the last generation), the discrete-time first-order replicator equations can be derived from (2.23) by setting $1/\Delta t = \mathbf{x}(t)^T A \mathbf{x}(t)$:

$$x_i(t + \Delta t) = x_i(t) \frac{A(\mathbf{x}(t))_i}{\mathbf{x}(t)^T A \mathbf{x}(t)} \tag{3.14}$$

And represent the population state of the offsprings of the previous generation.
This discrete version inherits most of the dynamical properties of the continuous time version, including fundamental theorem of natural selection and it has the same set of stationary points: hence even this one can find

local solutions for the quadratic problem (3.8). In fig 3.5 we can see the behavior of discrete time replicator dynamics using an asymmetric similarity matrix: the use of symmetric matrices avoids strange trajectories like cycles, invariant tori etc..

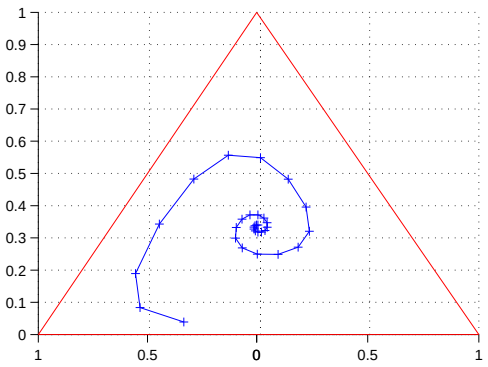


Figure 3.5: A possible trajectory of discrete time replicator dynamics in the simplex.

3.4.2 The algorithm

The algorithm of discrete time replicator dynamics is very simple and can be written in a few lines using an high-level programming language. It takes as input the payoff matrix $\mathbf{A} \in \mathbb{R}^{n^2}$, a vector $\mathbf{x} \in \Delta$ (possibily near the baricenter of the simplex), an error threshold $\epsilon > 0$, a iteration limiter T_{MAX} and returns an equilibrium point of the dynamics.

1. $t = 0$
2. $t = t + 1$
3. for all i, compute $\mathbf{x}_i(t) = \mathbf{x}_i(t-1)\frac{\mathbf{A}(\mathbf{x}(t))_i}{\mathbf{x}(t)^T\mathbf{A}\mathbf{x}(t)}$
4. if $t < T_{MAX}$ and $||\mathbf{x}(t) - \mathbf{x}(t-1)|| > \epsilon$ go to (2), else return $\mathbf{x}(t)$ and **END**.

Adding some random pertubation on the initial $\mathbf{x}(0)$ gives che chance to reach different Evolutionary Stable Strategies and so to extract even overlapped different clusters. If one does not want them to be overlapped she may pell off the elements of the cluster from the dataset before extracting the next one.

Chapter 4

Graph Transduction

4.1 Introdution

Dominant Set clustering tends to create very compact structures but, if we continue to peel off clusters from dataset, the last aggregations of data points will tend to be not very significative: if we want to partition the dataset, rather than iterate the algorithm until there are not points unclustered, it is better to peel off only few significative cluster and then to use other methods to decide in which existing clusters the unclassified points belong: the classic approach is to measure the distances between clusterized data and unclusterized data and then associate the latter to the cluster of the nearest classified point. In this chapter we want to use a different method come from Semi-supervised learning that uses the same game theoric framework of the Dominant-Set algorithm: the graph transduction method introducted by Erdem and Pelillo in [6].

Semi-supervised Learning is a class of machine learning techniques that, in order to model the geometry of the data, uses both labeled and unlabeled data. This approach is more recent than unsupervised and supervised learning and it gained popularity only in the last decade ([4] and [36]). The Graph transduction techniques propagate information from labeled to unlabeled data to achieve a better modelization (See fig 4.1) treating the data as nodes of a graph. In our case the labeled data are the points clustered by Dominant-Set algorithm while the unlabeled data are the remaining points. The task consists in the propagation of the cluster membership information in a way that, to be consistent, has to rely on a common a priori assumption known as the cluster assumption ([35] and [4]): this assumption states the following two propositions:

1. Points that are close to each other are expected to have the same label.

2. Points in the same cluster are expected to have the same label.

Figure 4.1: (a): Learning using only labeled points (supervised learning); (b): Learning using both labeled and unlabeled data (semi supervised learning)

While the classic graph-transduction techniques formalize the task as a regularized function estimation problem on a undirected graph the one we take account of in this chapter models the problem as a multiplayer noncooperative game where the players are the objects in the dataset that take part in the game to decide their class membership.

4.2 Propagating information in a unweighted graph

As in the Dominant-Set clustering chapter we start with the simplest case of graph transduction i.e. the propagation of information in a unweighted undirected graph (a graph whose similarity matrix $\mathbf{S}$ is symmetric and binary: $S_{ij} = 1$ if nodes i and j are similar, otherwise $S_{ij} = 0$). In such setup the problem can be formulated as a binary constraint satisfaction problem (CSP), a class of problems used widely in artificial intelligence and computer vision domain. This binary CSP can be defined by a triple (V, D, R) where $V = \{v_1, ..., v_n\}$ is a set of variables; $D = \{D_{v_1}, ..., D_{v_n}\}$ is a set of domains of the variables (i.e. D_{v_i} is the set of labels that can be assigned to v_i); $R = \{R_{ij} | R_{ij} \subseteq D_{v_i} \times D_{v_j}\}$ is a set of binary constraints where R_{ij} represents the compatible pairs of labels for variables v_i and v_j: each R_{ij} can be seen as a binary pq matrix where p and q are, respectively, the cardinality of D_{v_i} and D_{v_j}: $\mathbf{R}_{ij}(\lambda, \lambda') = 1$ if the assignments $v_i = \lambda$ and $v_j = \lambda'$ are compatible, otherwise $\mathbf{R}_{ij}(\lambda, \lambda') = 0$. Resolving a binary CSP consists in assigning to each variable in V a label in its domain respecting all the constraints: this problem is known to be NP-complete (see [11]). In our case the setup is slighty different, indeed in our problem V is composed by a set of l clustered objects (whose label domain D has cardinality 1) and a set of non-clustered points (whose domain has cardinality equal to c where c is the number of clusters). Given a set $\phi = \{1, 2, ..., c\}$ of labels, such case can be formalized as a binary constraints satisfaction problem as follows:

- A set of variables: $V = \{v_1, ..., v_n\}$

- Domains: $D_{v_i} = \begin{cases} \{\phi_i\} & \text{for all } 1 \leq i \leq l \\ \phi & \text{for all } l+1 \leq i \leq n \end{cases}$
- Binary constraints: given a similarity matrix $\mathbf{S}$, if $S_{ij} = 1$ then $v_i = v_j$: the correspondenting $\mathbf{R}_{ij}$ in a two class problem is $\mathbf{R}_{ij} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

The existence of any solution in such problem depends by the structure of the graph and the positions of the labeled points in it: in an unweighted undirected graph, if exists a path between two different labeled points then the contraints cannot be satisfied. In order to manage real world problems we have to use softer constraints and so, instead of model them as binary graph we have to use weighted graphs.

4.3 The graph transduction game

We can traspose the elements of the CSP problem in a multiplayer game whose players are the objects (variables in the CSP) and the strategies for each player are the labels in their domains. The players can so be divided in two disjoint groups: those that play always the same strategy (the labeled points that have $|D| = 1$) and the players that play mixed strategies lied in the standard c-dimensional simplex (the unlabeled points that have $|D| = c$); we denote the first group with I_l and the second with I_u. I_l is the union of the sets $I_{l|1}, ..., I_{l|c}$ where $I_{l|k}$ is the subset of points in I_l playing only pure strategy k that can be seen as an extreme mixed strategy $\mathbf{e}_i^k \in \Delta_i$. In such terms the points in I_l are not really players because they do not try to maximize their payoffs but act as bias over the choices of unlabeled players.

By theorem 2.1 we know that this setup has always a Nash Equilibrium in mixed strategies (see [25]) hence once such stable point is reached we can label the unclustered data points using the pure strategy with the highest probability in its equilibrium mixed strategy:

$$\phi_i = \arg_{h=1...c} \max x_{ih} \tag{4.1}$$

In this graph transduction game only pairwise interactions are allowed and they are independent: this fact means that the game belongs to a special class of non-cooperative games called polymatrix games.

Definition 4.1. (Polimatrix game)
A polymatrix game is a non-cooperative game in which the relative influence of the selection of a pure strategy by any one player on the payoff to any other player is always the same, regardless of what the rest of the players do.

The indipendence between partial payoffs implies that each pair of players i and j has a relative payoff matrix $\mathbf{A}_{ij}$ the payoff of player i in such game is

$$\pi_i(\mathbf{s}) = \sum_{j=1}^{n} \mathbf{A}_{ij}(s_i, s_j) \tag{4.2}$$

where s is a pure strategy profile. Even the expected payoff the payoffs is additively separable: given a mixed strategy profile $\mathbf{x}$ if $i \in I_l$ then it plays only a pure strategy h:

$$u_i(\mathbf{e}_i^h, \mathbf{x}_{-i}) = \sum_{j=1}^{n} (\mathbf{A}_{ij}\mathbf{x}_j)_h \tag{4.3}$$

otherwise, if $i \in I_u$ then it plays a mixed strategy

$$u_i(\mathbf{x}) = \sum_{j=1}^{n} \mathbf{x}_i^T \mathbf{A}_{ij}\mathbf{x}_j \tag{4.4}$$

We can even split the sum of (4.3) and (4.4) in two parts, the first defining the expected payoff against players that play mixed strategies and the second the payoff against players that play only a pure strategy:

$$u_i(\mathbf{e}_i^h, \mathbf{x}_{-i}) = \sum_{j \in I_u} (\mathbf{A}_{ij}\mathbf{x}_j)_h + \sum_{k=1}^{c} \sum_{j \in I_{D|k}} \mathbf{A}_{ij}(h, k) \tag{4.5}$$

$$u_i(\mathbf{x}) = \sum_{j \in I_u} \mathbf{x}_i^T \mathbf{A}_{ij}\mathbf{x}_j + \sum_{k=1}^{c} \sum_{j \in I_{D|k}} \mathbf{x}_i^T (\mathbf{A}_{ij})_k \tag{4.6}$$

4.3.1 Partial payoff matrix

In the polimatrix games there is a partial payoff matrix for each pair of players, in the binary unweighted case we use a 0/1 matrix: if we consider a weighted graph $G = (D, \epsilon, \omega)$ with the weight matrix $\mathbf{W} = (w_{ij})$, the partial payoff between players i and j can be set as $\mathbf{A}_{ij} = w_{ij} \times \mathbf{I}_c$, so the whole A matrix is $\mathbf{A} = \mathbf{I}_c \otimes \mathbf{W}$ where $\otimes$ is the Kronecker product. If the graph is unweighted $\mathbf{A}_{ij}$ coincides with the compatibility matrices of the CSP. As a title of example in a 3 class case we have

$$\mathbf{A}_{ij} = \begin{bmatrix} w_{ij} & 0 & 0 \\ 0 & w_{ij} & 0 \\ 0 & 0 & w_{ij} \end{bmatrix}. \tag{4.7}$$

The use of $\hat{\mathbf{W}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ where $\mathbf{W}$ is the original weight matrix of the dataset and $\mathbf{D} = (d_{ii})$ is the diagonal degree matrix of $\mathbf{W}$ with $d_{ii} = \sum_j w_{ij}$ to perform the graph transduction achieves usually better performance of the direct use of $\mathbf{W}$.

4.3.2 Multi-Population Replicator Dynamics: Discrete time

In order to compute a Nash Equilibrium of the polimatrix game we can use

$$x_{ih}(t+1) = x_{ih}(t)\frac{u_i(\mathbf{e}_i^h, \mathbf{x}_{-i})}{u(\mathbf{x})} \tag{4.8}$$

that is the discrete time version of the multi-population replicator dynamics (2.27) Even this dynamic has the same stationary points of it continuous counterpart are Nash equilibria.

4.4 Graph transduction as a relaxation labeling process

We now show that the graph transduction game is equivalent to a particular setup of a context-based classification method introducted by Rosenfeld, Hummel, and Zucker in 1976: the relaxation labeling processes. Such techniques are a class of algorithms that exploits the contestual information in order to perform a better classification of the data. They are used to solve many pattern recognition and computer vision problems like region based segmentation, graph matching, perceptual organization, stereo matching and so on.

4.4.1 The labeling problem

The classic labeling problem can be defined as follows:

Definition 4.2. (labeling problem)
Given:

- A set of n objects $B = \{b_1, ..., b_n\}$
- A set of m labels $L = \{1, ..., m\}$

A labeling problem consists to label each object in B with a label in L.

Relaxation labeling algorithms use two different sources of infomation to achieve this goal:

- non-contextual local information i.e. the salient features of each object in B;
- pairwise contextual information between each pair of objects in B, represented as an n^2m^2 real-valued 4-dimensional matrix $\mathbf{R} = (\mathbf{r}_{ij}(\lambda, \mu))$ called compatibilty coefficient matrix.

$\mathbf{r}_{ij}(\lambda, \mu)$ represents the measure of compatibility between the propositions "Object b_i is labeled with label λ" and "Object b_j is labeled with label μ". The relaxation labeling process begins by creating an m-dimensional probability vector $\mathbf{p}_i$ for each object, where $\mathbf{p}_i(\lambda)$ represents the probability that the object b_i is labeled with label λ given only its local features. If we concatenate the vectors $\mathbf{p}_i$ we obtain the $n \times m$ matrix $\mathbf{P} \in \mathbb{K}$ where $\mathbb{K}$ is the space of the weighted label assignments:

$$\mathbb{K} = \underbrace{\Delta \times \Delta \times \cdots \times \Delta}_{n \text{ times}} \tag{4.9}$$

where Δ is the standard simplex of $\mathbb{R}^m$: we have unambiguous labeling assignments in vertices of $\mathbb{K}$.
Once we have the initial $\mathbf{P}^{(0)} = \mathbf{P}$ it is iteratively updated using the Rosenfeld-Hummel-Zucker Rule (or RHZ operator):

$$\mathbf{p}_i^{(t+1)}(\lambda) = \frac{\mathbf{p}_i^{(t)}(\lambda)\mathbf{q}_i^{(t)}(\lambda)}{\sum_\mu \mathbf{p}_i^{(t)}(\mu)\mathbf{q}_i^{(t)}(\mu)} \tag{4.10}$$

where $q_i^{(t)}(\mu)$ represents the support that the whole context gives to the hypotesis "b_i is labeled with label μ" and is computed taking account of compatibility coefficient matrix $\mathbf{R}$:

$$\mathbf{q}_i^{(t)}(\lambda) = \sum_j \sum_\mu \mathbf{r}_{ij}(\lambda, \mu)\mathbf{p}_i^{(t)}(\mu) \tag{4.11}$$

the denominator of (4.10) is used only to normalize the results. The relaxation labeling process ends when, for each i and λ, $|p_i^{(t)}(\lambda) - p_i^{(t-1)}(\lambda)| < \epsilon$ where ϵ is a chosen positive number. The algorithm can be schemed as follows.

Relaxation Labeling Algorithm

Input: a list of n objects, a list of m labels, a n^2m^2 4-dimensional matrix $\mathbf{R}$.

1. $t = 0$
2. Create $p^{(0)}$ using the local non-contextual information of the objects.
3. $t = t + 1$.
4. Compute all $\mathbf{q}_i^{(t-1)}$ using (3.10).
5. Compute all $\mathbf{p}_i^{(t)}$ using (3.9).
6. if, for each i and λ, $|p_i^{(t)}(\lambda) - p_i^{(t-1)}(\lambda)| < \epsilon$ **END**, else go to 3.

In the next section we check which conditions assure the convergence of the algorithm and if it converges always in a consistent labeling.

4.4.2 Consistency of a labeling

While the task to decide if an unambiguous labeling is consistent is straightfoward, it is not so trivial define the consistency in a weighted labeling assignment. Hummel and Zucker give in [15] the following definition:

Definition 4.3. (consistent labeling)
A weighted labeling assignment $\mathbf{P} \in \mathbb{K}$ is consistent if

$$\sum_{\lambda} \mathbf{p}_i(\lambda)\mathbf{q}_i(\lambda) \geq \sum_{\lambda} \mathbf{v}_i(\lambda)\mathbf{q}_i(\lambda) \; i = 1, ..., n \tag{4.12}$$

for all $\mathbf{V} \in \mathbb{K}$. If strict disequality always holds then $\mathbf{P}$ is strictly consistent.

Strict consistence holds only if the labeling is unambiguous.
Another way to characterize the consistency of a labeling assignment is explained by the following theorem:

Theorem 4.1.
A labeling $\boldsymbol{P} \in \mathbb{K}$ is consistent if and only if for all $i = 1...n$ the following conditions hold:

1. $\boldsymbol{q}_i(\lambda) = c_i$, *whenever* $\boldsymbol{p}_i(\lambda) > 0$
2. $\boldsymbol{q}_i(\lambda) \leq c_i$, *whenever* $\boldsymbol{p}_i(\lambda) = 0$

for some nonnegative constants $c_1, ..., c_n$.

Proof: See [27].

□

In the same paper, Hummel and Zucker define the average local consistency as

$$A(\mathbf{P}) = \sum_{i=1}^{n} \sum_{\lambda=1}^{m} \mathbf{p}_i(\lambda)\mathbf{q}_i(\lambda) \tag{4.13}$$

and prove the following theorem:

Theorem 4.2.
Suppose that the compatibility matrix $\boldsymbol{R}$ is symmetric (i.e. $\boldsymbol{r}_{ij}(\lambda, \mu) = \boldsymbol{r}_{ji}(\mu, \lambda)$ for all i, j, λ, μ) then any local maximum $\boldsymbol{P} \in \mathbb{K}$ of A is consistent.

Proof: See [27].

□

Finally we enunciate a fundamental result that explicitely show the connection, under some assumptions, between the RHZ operator and the average local consistency:

Theorem 4.3.
The non linear relaxation RHZ operator is a growth transformation for the average local consistency A, provided that compatibility coefficients are non-negative and Symmetric:

$$A(\boldsymbol{P}^{(t+1)}) > A(\boldsymbol{P}^{(t)}) \tag{4.14}$$

for $t = 0, 1, ...$

Proof: See [27].

□

Hence, to sum up, if $\mathbf{R}$ is not negative and symmetric, the consistent labeling assignments are the local maximum of the average local consistency A, the RHZ operator increases A during time and, consequently, it reaches a consistent labeling assignment when it converges.

4.4.3 RHZ operator and replicator dynamics

If we use a relaxation labeling process to achieve the graph tranduction task the matrix $\mathbf{R}$ becomes simply an n^2 matrix in which the element $\mathbf{R}_{ij}$ is the m^2 partial payoff diagonal matrix between players i and j. With such R the computation of $q_i^{(t)}$ becomes simpler:

$$\begin{aligned} q_i^{(t)}(\lambda) &= \sum_j \sum_\mu r_{ij}(\lambda, \mu) p_j^{(t)}(\mu) \\ &= \sum_j r_{ij}(\lambda, \lambda) p_j^{(t)}(\lambda) = \\ &= \sum_j w_{ij} p_j^{(t)}(\lambda) \end{aligned} \tag{4.15}$$

In this setup the RHZ operator can be seen as a multipopulation version of the replicator dynamics explained in chapter 3. If we consider each element $P_{ih}^{(t)}$ as the ratio of the population i that plays the pure strategy h at time t.

$$\begin{aligned} q_i(\lambda) &= \sum_j w_{ij} p_j^{(t)}(\lambda) = \\ &= \sum_{j \in I_u} (w_{ij} x_j)_h + \sum_{k=1}^{c} \sum_{j \in I_{D|k}} A_{ij}(h, k) = \\ &= u_i(e_i^h, x_{-i}) \end{aligned} \tag{4.16}$$

In the same way it can be demostrated that $\sum_\mu p_i^{(t)}(\mu)q_i^{(t)} = u_i(x)$ so the RHZ operator becomes

$$P_{i\lambda} = P_{i\lambda}\frac{u_i(e_i^\lambda, x_{-i})}{u_i(x)}. \tag{4.17}$$

This equivalence implies that replicator dynamics limit point are not only Nash Equilibrium but even consistent labeling.

4.5 Implementation

Now we see how relaxation labeling algorithms can be implemented in order to achieve a graph transduction task. Our initial data are l labeled objects, $n-l$ unlabeled objects, m labels (the clusters) and a n^2 zero-diagonal similarity matrix $\mathbf{S}$. The first step is to compute the initial $\mathbf{p}_i^{(0)}$ and the compatibility coefficient matrix $\mathbf{R}$.

In our case we don't have to extract features from objects: the value of $\mathbf{p}_i^{(0)}$ depends by the membership of the object i: if $i \in I_l$ its initial $\mathbf{p}_i$ will be $\mathbf{e}^k$ where k is the cluster it belongs; otherwise, if $i \in I_u$, $\mathbf{p}_i^{(0)} = 1/m * \mathbf{1}$. Obviously $p_i^{(0)} \in \Delta_m$ for each $i = 1, ..., n$. Now, such that $w_{ij} = S_{ij}$, we are able to compute the nm matrix $\mathbf{Q}$ with $Q_{i\lambda} = q_i(\lambda)$ as follows:

$$\mathbf{Q}^{(t)} = \mathbf{S} \times \mathbf{P}^{(t)} \tag{4.18}$$

$\mathbf{p}_i^{(t)}$ can be computed using (4.9). The Graph Transduction Algorithm using relaxation labeling techniques can be schemed in this way:

Input: a list of l labeled objects, n-l unlabeled objects, a list of m clusters, a n^2 matrix $\mathbf{S}$.

1. $t = 0$
2. Create $\mathbf{P}^{(0)}$ as explained above.
3. $t = t + 1$.
4. Compute $\mathbf{Q}^{(t-1)}$ using (4.15).
5. Compute $\mathbf{P}^{(t)}$ using (4.9).
6. if $|\mathbf{P}^{(t)} - \mathbf{P}^{(t-1)}| < \epsilon$ exit, else go to 3.

Chapter 5

Kernel Methods

In the previous chapters we have shown the Dominant-Set clustering and the game-theoeretic graph transduction algorithms. Both methods take as input a zero-diagonal matrix of pairwise similarities but we have omitted how this data structure can be generated given a set of objects: there are many measures of similarity and obviously the choice among them depends on the nature of the data.
There is another class of algorithms that uses a square matrix of pairwise comparisons to process the data, and it is the class of kernel methods. Such pairwise functions used as comparisons are called kernels. In this chapter we show what are the kernels and the way in which they can be used as measures of pairwise similarity for objects $\mathbf{x}_i \in \mathbb{R}^p$ in order to use them to compute different payoff matrices for the algorithms.

This chapter is structured as follows: section 5.1 defines the notion of kernel; section 5.2 gives a brief introduction to the Reproducing Kernel Hilbert Spaces (RKHS). Section 5.3 explains the kernel trick, that permits to transform classic algorithms in kernel methods. Section 5.4 defines some kernels for objects represented as vectors. Section 5.5 give an example to how to trasform the classic K-means algorithm in a "kernel K-means". Section 5.6 shows how kernels can be combined to create new kernels and finally section 5.7 present a subset of kernels called "translation invariant kernels" that better than others exprime the notion of similarity between objects.

5.1 Kernels

Let be $S = \{x_1, ..., x_n\}$ a dataset of n objects, with x_i element of a set χ that can be thought as the set of all the elements of a certain type (e.g. images, molecules, strings, ...). In order to analyze in some way the set S we have to choose how to represent the objects in it. The classical way is to represent each data x_i explicitely in a pre-determined way $\phi(x_i) \in \mathbb{F}$ (the so called feature space) defined for all $x \in \chi$: the resulting dataset

$\phi(S) = \{\phi(x_1), ..., \phi(x_n)\}$ can so be processed by some sort of algorithm able to manage this kind of data. The rapresentation of the objects used by kernel methods is radically different: instead of individually represent each data they represent the comparisons between each pair $(x_i, x_j) \in S \times S$ using a real (or complex) valued comparison function $k : \chi \times \chi \to \mathbb{C}$ so the new representation of the data is a n^2 matrix (called Gram Matrix) $K_{ij} = k(x_i, x_j)$. Kernel methods are the class of algorithms that use the gram matrix $\mathbf{K}$ to process the data.
Using pairwise comparisons instead of explicit representations of the objects has some advantages: first of all, the gram matrix representation does not depend from the nature of data to be analyzed: S could be a set of every kind of objects, its gram matrix $\mathbf{K}$ will be always a real (or complex) valued matrix. Such fact means that kernel methods can manage every dataset that can be represented as $\mathbf{K}$ without changes in implementation. Secondly (but not less important), the size of the gram matrix is always n^2 whatever the complexity of the objects in S: it implies that kernel methods are able to manage datasets with a small number of complex objects more efficently than classic methods. As a title of example, if a dataset S is composed by 20 objects each of them represented by a 1000-dimensional vector, kernel methods have to process only a $20 \times 20 = 400$ matrix while the other methods have to manage a $20 \times 1000 = 40000$ dataset. By converse it is a disadvatage if the dataset is composed by many simple objects: if S has 1000 obects represented by 20-dimensional vectors the gran matrix will have size $1000 \times 1000 = 1000000$ while the dataset $\phi(S)$ will have only a size $1000 \times 20 = 20000$.
The third important advantage is that in many cases comparing certain types of data is simpler than finding how to represent the objects in a relevant way.

5.1.1 Positive Definite Kernels

We give now a formal definition of the Gram Matrix and of a particular class of them, the positive definite gram matrices.

Definition 5.1 (Gram Matrix)**.**
Given a kernel k and a dataset $S = x_1, ...x_n \subset \chi$, the n^2 matrix

$$\mathbf{K} := (k(x_i, x_j))_{ij} \tag{5.1}$$

is called the Gram matrix (or kernel matrix) of k with respect of S.

Most of kernel methods are designed to process only a particolar class of Gram Matrices, those that are positive definite. We recall below the mathematical definition of positive definiteness:

Definition 5.2 (Positive definite matrix)**.**
Let $\mathbf{K}$ be a n^2 symmetric matrix: if it satisfies

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K_{ij} \geq 0 \tag{5.2}$$

for any $n > 0$, $x_1, ..., x_n \in \chi$ and $c_1, ..., c_n \in \mathbb{R}$ it is called positive definite. If equality in (3.2) only occurs for $c_1, ..., c_n = 0$ the matrix is strictly positive definite.

This class of symmetric gram matrices implicitely defines a class of kernels:

Definition 5.3 (positive definite kernel)**.**
Let χ be a non-empty set. A function $k : \chi \times \chi \to \mathbb{R}$ that gives rise to a positive definite gran matrix is called positive definite kernel. If it give rise to a strict positive definite gram matrix is called a strictly positive definite kernel.

In order to create symmetric gram matrices, positive definite kernels (for the sake of simplicity we will call them only kernels) need to have the following property:

$$k(x_i, x_j) = k(x_j, x_i) \tag{5.3}$$

for all x_i, x_j in S.

5.1.2 Kernels as inner products

In the previous section we have definited (positive definite) kernels by means of what they give rise. Now we see how this class of kernel is. In order to do this job we recall the definition of Hilbert Space:

Definition 5.4 (Hilbert Space)**.**
A Hilbert Space is an inner product space that is complete and separable with respect to the norm defined by the inner product.

An Hilbert space is not restricted by the number of dimensions: an example of finite-dimensional Hilbert space is $\mathbb{R}^p$ with inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T\mathbf{y}$ when $p \in \mathbb{R}$, an infinite dimensional Hilbert space is $\mathbb{R}^p$ when $p = \infty$. Let $\chi = \mathbb{R}^p$ (i.e the objects are real vectors $\mathbf{x} = (x_1, ..., x_p)^T$. We can see that the inner product between pairs of vectors $\mathbf{x}, \mathbf{x'} \in \mathbb{R}^p$

$$k_L(\mathbf{x}, \mathbf{x'}) = \mathbf{x}^T\mathbf{x'} = \sum_{i=1}^{p} x_i x'_i \tag{5.4}$$

is a kernel. The inner product between real vector is indeed commutative and the kernel generates positive definite gram matrices:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_L(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathbf{x}_i^T \mathbf{x}_j = ||\sum_{i=1}^{n} c_i \mathbf{x}_i||^2 \tag{5.5}$$

for any $n > 0$, $x_1, ..., x_n \in \mathbb{R}^p$ and $c_1, ..., c_n \in \mathbb{R}$.
The kernel (5.4) is called *Linear Kernel* and is the simplest type of kernel. It has some limitations, indeed it can manage only real valued vectors. this fact may suggest that if we want to process other type of objects $x \in \chi$ we can rapresent such objects as vectors by means of some mapping function $\phi : \chi \to \mathbb{R}^p$ and then we can define a new kernel:

$$k(x, x') = \phi(x)^T \phi(x') \tag{5.6}$$

for all $x, x' \in \chi$.
Every inner product of objects projected in a real vector space is a kernel. Aronzajn (see [24]) proved that positive definite kernels more general than these don't exist.

Theorem 5.1.
For any kernel k on a space χ, there exist a Hilbert space $\mathbb{F}$ and a mapping $\phi : \chi \to F$ such that, for any $x, x' \in \chi$

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{5.7}$$

where $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ represent the dot product in the Hilbert space between any two points $\boldsymbol{u}, \boldsymbol{v} \in F$.

Proof: See [24].

□

The theorem states that all kernels can be seen as dot products in some space $\mathbb{F}$ (the so called feature space). The fact that we are interested only in the dot products to generate the gram matrix make the esplicit mapping of every object unnecessary. In addition the feature space associated to some kernels is infinite dimensional and so kernel methods permit to manage objects in feature spaces infeasible to be managed by classic methods.

5.1.3 Kernels as similarity measures

Is right to consider a dot product between two vectors a measure of their similarity? Let $\mathbf{x} = \mathbf{x'} = (2, 0, 2, 0)^T$, $\mathbf{y} = (20, 10, 20, 10)^T$ and $\mathbf{y} = (20, 0, 20, 0)^T$, we have that $\mathbf{x}^T\mathbf{x'} = 2^2 + 2^2 = 8$ and $\mathbf{y}^T\mathbf{y'} = 20^2 + 20^2 = 800$: $\mathbf{x}$ is equal to $\mathbf{x'}$ but, if we consider the dot product as measure of similarity

they seems to be less similar from one another than the pair $\mathbf{y}, \mathbf{y}'$ even if $\mathbf{y} \neq \mathbf{y'}$ so not all the kernels can be used directly as similarity measures, but have to be normalized.

$$k_N(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)}\sqrt{k(x', x')}} \tag{5.8}$$

Rather than dot product, similarity is a concept more related with the notion of distance. Some kernels that incorporate the notion of distance exist: given $\chi = \mathbb{R}^p$ the function

$$K_G(\mathbf{x}, \mathbf{x'}) = \exp(-\frac{d(\mathbf{x}, \mathbf{x'})^2}{2\sigma^2}), \tag{5.9}$$

where σ is a parameter and d is the euclidean distance is the Gaussian Radial Basis Function (RBF) kernel. It can be shown that this kernel represent the dot product between two vectors in a infinite dimensional space. If we examine this kernel we can see that it assumes the maximum value ($= 1$) then the vectors $\mathbf{x}$ and $\mathbf{x'}$ are equals (their distance is 0) and decreases when the distance increases, so it can be seen as a proper measure of similarity.

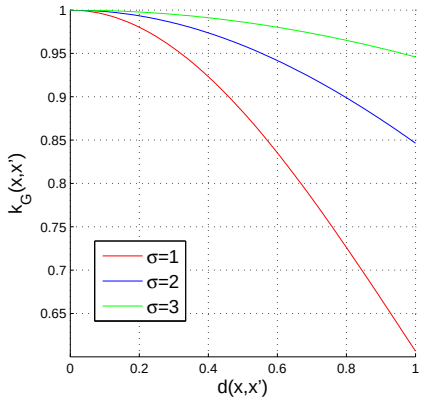


Figure 5.1: Gauss Kernel values changing the distance between points $\mathbf{x}$ and $\mathbf{x'}$

5.2 The Reproducing Kernel Hilbert Space

Each kernel k on a space χ is associated with a set of real-valued function on χ:$H_k \subset \{f : \chi \to \mathbb{R}\}$ endowed with a structure of Hilbert space and, in particular, with a dot product and a norm. Such set is defined as the set of functions $f : \chi \to \mathbb{R}$ of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \tag{5.10}$$

for $n > 0$, $x_1, ..., x_n \in \chi$ and $\alpha_1, ..., \alpha_n \in \mathbb{R}$.
The norm associated to such set of function is

$$||f||^2_{H_k} := \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \tag{5.11}$$

The dot product associed to H_k between two elements $f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$ and $g(x) = \sum_{j=1}^{m} \alpha'_i k(x'_j, x)$ is

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \alpha'_j k(x_i, x'_j) \tag{5.12}$$

We can see that if we define $f(.) = \sum_{i=1}^{n} \alpha_i k(x_i, .)$ and $g(.) = k(x, .)$ we have the equality

$$\langle f, g \rangle = f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \tag{5.13}$$

If we set $f(.) = k(x', .)$ we have

$$k(x, x') = \langle k(x, .), k(x', .) \rangle \tag{5.14}$$

Such reproducing property makes H_k the reproducing kernel Hilbert space (RKHS) associated with k.

We show as a title of example the H_k and the associated norm of the Linear Kernel and the Gaussian RBF kernel.

Linear Kernel

Let k_L be the linear kernel in a space $\chi = \mathbb{R}^p$. Its functional space is the space of linear function $f : \mathbb{R}^p \to \mathbb{R}$:

$$H_{k_L} = \{f(x) = w^T x : w \in \mathbb{R}^p\} \tag{5.15}$$

and its norm is the slope of the linear function:

$$||f||_{H_{k_l}} = ||w|| \text{ for } f(x) = w^T x. \tag{5.16}$$

Gaussian Kernel

The functional space of Gaussian RBF kernel in the same space $\chi = \mathbb{R}^p$ is the set of functions $f : \mathbb{R}^p \to \mathbb{R}$ with Fourier Transform $\hat{f}$ satisfying:

$$N(f) = \frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \int_{\mathbb{R}^p} |\hat{f}(\omega)|^2 e^{\frac{\sigma^2}{2}||\omega||^2} d\omega < +\infty \tag{5.17}$$

and the corrisponding norm is $N(f)$ itself.

5.3 The Kernel Trick

In order to develop variants of existing algorithms that use kernels instead of explicit representations of the objects a simple principle can be defined:

Proposition 5.2.
Any algorithm for vectorial data that can be expressed only in terms of dot products between vectors can be performed implicitily in the feature space associated with any kernel, by replacing each dot product by a kernel evaluation.

This principle shows that if we have an linear method able to manage objects in the form of pairwise dot products, we can transform it in a non-linear algorithm simply replacing the dot products with a non-linear kernel without increasing the computational cost.
Another consequence of this statement is that even if χ is not a vectorial space, we can use such "dot-product methods" using a kernel $k : \chi \to \mathbb{R}$. In this way kernel methods are able to manage every type of data.
We now show how to use the kernel trick in order to compute distance between points. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$ we know that the euclidean distance between two point in a multidimensional space is

$$d(x, y) = ||x - y||_2 \tag{5.18}$$

we have to transform such formula in another one in which the the objects are espressed only in the form of dot products:

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_2 = \\ &= \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} = \\ &= \sqrt{\sum_{i=1}^{p} (x_i^2 + y_i^2 - 2x_i y_i)} = \\ &= \sqrt{\sum_{i=1}^{p} x_i^2 + \sum_{i=1}^{p} y_i^2 - 2\sum_{i=1}^{p} x_i y_i} = \\ &= \sqrt{\mathbf{x}^T\mathbf{x} + \mathbf{y}^T\mathbf{y} - 2\mathbf{x}^T\mathbf{y}} \end{aligned} \tag{5.19}$$

Now by replacing the dot products with any kernel we can compute the distance in the space defined by the kernel:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{y}, \mathbf{y}) - 2k(\mathbf{x}, \mathbf{y})}. \tag{5.20}$$

Hence one can compute the distance between points without knowing their ubications but using only the gran matrix in a more efficient way, expecially if the space of the objects has high dimensionality.

5.4 Other Kernels for Vectors

We have seen the linear kernel and the gaussian RBF kernel, now we introduce other two classes of kernels that have $\chi = \mathbb{R}^p$:

Polynomial Kernels:
Let $c \geq 0$ and $p \in N^+$. The kernels in the form

$$k_p(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^p \tag{5.21}$$

are called polinomial kernels. The linear kernel (5.4) is a particolar instance of this class of kernel when $p = 1$ and $c = 0$.
Let $p = 2$, $c = 0$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$. We can solve the kernel $k_P(\mathbf{x}, \mathbf{y})$ by explicing the feature mapping ϕ:

$$\begin{aligned} k_P(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T\mathbf{y})^2 = \\ &= (\mathbf{x}(1)\mathbf{y}(1) + \mathbf{x}(2)\mathbf{y}(2))^2 = \\ &= \mathbf{x}(1)^2\mathbf{y}(1)^2 + \mathbf{x}(2)^2\mathbf{y}(2)^2 + 2\mathbf{x}(1)\mathbf{y}(1)\mathbf{x}(2)\mathbf{y}(2) = \\ &= (\mathbf{x}(1)^2, \mathbf{x}(2)^2, \sqrt{2}\mathbf{x}(1)\mathbf{x}(2))(\mathbf{y}(1)^2, \mathbf{y}(2)^2, \sqrt{2}\mathbf{y}(1)\mathbf{y}(2))^T \\ &= \phi(\mathbf{x})^T\phi(\mathbf{y}) \end{aligned} \tag{5.22}$$

We see that the function $\phi(\mathbf{x}) = (\mathbf{x}(1)^2, \mathbf{x}(2)^2, \sqrt{2}\mathbf{x}(1)\mathbf{x}(2))$ comprises all possible second order terms of the element of $\mathbf{x}$. Generalizing, given $p > 0$ and a vector $\mathbf{x} \in \mathbb{R}^d$, the mapping ϕ associated to the polynomial kernel project $\mathbf{x}$ in a $\binom{d+p-1}{p}$-dimensional space and $\phi(\mathbf{x})$ comprises all possible p-order terms of the elements of $\mathbf{x}$.

Sigmoid Kernels:
The Sigmoid kernels are defined as

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa\mathbf{x}^T\mathbf{x}' + \theta) \tag{5.23}$$

where $\kappa > 0$ is a parameter called gain and $\theta < 0$ is called threshold. They are used with success in support vector machines because simulate well particular types of sigmoidal neural networks, even if it they are not always positive definite.

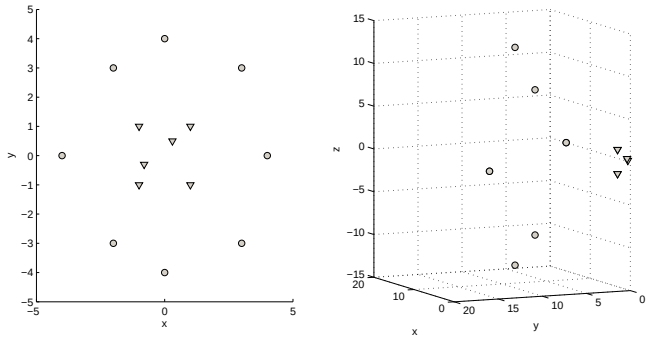


Figure 5.2: (a): Original dataset composed by two classes of objects not linearly separable; (b), projection of the data using a polynomial kernel with $p = 2$: in this space the two classes are linearly separable

5.5 An Example of Kernel Method: Kernel K-means

K-mean is the most famous clustering algorithm. Given a dataset S with n objects $\mathbf{x}_i \in \mathbb{R}^p$ this EM algorithm splits the objects $\mathbf{x}_j$ in C partitions P_i each of them associated to a prototype $\mathbf{c}_i \in \mathbb{R}^p$ called centroid.
The objective function that k-means try to minimize is

$$\sum_{i=1}^{C} \sum_{\mathbf{x}_j \in P_i} ||\mathbf{x}_j - \mathbf{c}_i||^2 \tag{5.24}$$

The classic K-means algorithm is:

K-means Algorithm
Input: a dataset of objects $\mathbf{x}_i \in \mathbb{R}^p$, the number of clusters C.

1. $t = 0$.
2. choose C random points $\mathbf{c}_i^{(t)} \in \mathbb{R}^p$.
3. $t = t + 1$.
4. compute squared distances between every point of the dataset $\mathbf{x}_j$ with every centroid $\mathbf{c}_j$: $d(\mathbf{x}_j, \mathbf{c}_i)^2 = ||\mathbf{x}_j - \mathbf{c}_i||^2$.
5. assign each point $\mathbf{x}_j$ to the partition P_i associated to the nearest centroid $\mathbf{c}_i$.
6. re-compute $\mathbf{c}_i$ as the means of the points in its partition: $\mathbf{c}_i^{(t)} = \frac{1}{|P_i|} \sum_{\mathbf{x}_j \in P_i} \mathbf{x}_j$.

7. for each $\mathbf{c}_i$, if $||\mathbf{c}_i^{(t)} - \mathbf{c}_i^{(t-1)}|| < \epsilon$:**END** else go to 3.

In order to transform it in a kernel method we have to make some considerations. First of all the error function that Kernel K-means has to minimize becomes:

$$\sum_{i=1}^{K} \sum_{\mathbf{x}_j \in P_i} ||\phi(\mathbf{x}_j) - \mathbf{c}_i||^2 \tag{5.25}$$

where ϕ is the mapping implicitely defined by the kernel and $\mathbf{c}_i$ are the centroid in the RKHS space defined by the kernel.
The data we have as input is the gran matrix $K = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$ for any $\mathbf{x}_i, \mathbf{x}_j \in S$ that contains only the dot products between pairs of points in S: we don't have any information of the implicit space defined by k so we cannot initialize explicitely the centroids. We can only define them implicitely by means of K. The squared distance between a point $\mathbf{x}$ and a centroid $\mathbf{c}$ can be rewrited as

$$\begin{aligned} d(\mathbf{x}, \mathbf{c})^2 &= ||\phi(\mathbf{x}) - \mathbf{c}||^2 = \\ &= ||\phi(\mathbf{x}) - \frac{1}{|P|} \sum_{\mathbf{x}_j \in P} \phi(\mathbf{x}_j)||^2 = \\ &= \phi(\mathbf{x})^2 + \frac{1}{|P|^2} (\sum_{\mathbf{x}_i \in P} \phi(\mathbf{x}_i))^2 - \frac{2}{|P|} \phi(\mathbf{x}) \sum_{\mathbf{x}_i \in P} \phi(\mathbf{x}_i) = \\ &= k(\mathbf{x}, \mathbf{x}) + \frac{1}{|P|^2} \sum_{\mathbf{x}_i \in P} \sum_{\mathbf{x}_j \in P} k(\mathbf{x}_i, \mathbf{x}_j) - \frac{2}{|P|} \sum_{\mathbf{x}_i \in P} k(\mathbf{x}, \mathbf{x}_i) \end{aligned} \tag{5.26}$$

In such way we can compute distances without explicitely knowing the positions of the centroids in the space. The first generation of the centroids can be chosen at random from the objects in S.
The kernel version of the K-means algorithm is:

Kernel K-means algorithm
Input: a gram matrix $\mathbf{K}$, the number of clusters C.

1. $t = 0$.
2. choose C random points $\mathbf{c}_i$ from S (the first generation of centroids).
3. $t = 1$.
4. compute squared distances between every point of the dataset $\mathbf{x}_j$ with every centroid using kernel trick:
   $d(\mathbf{x}_j, \mathbf{c}_i)^2 = k(\mathbf{x}_j, \mathbf{x}_j + k(\mathbf{c}_i, \mathbf{c}_i) - 2k(\mathbf{x}_j, \mathbf{c}_i)$

5. assign each point $\mathbf{x}_j$ to the partition P_i associated to the nearest centroid $\mathbf{c}_i$.

6. t=t+1

7. recompute the squared distances using
$d(\mathbf{x}_j, \mathbf{c}_i)^2 = k(\mathbf{x}, \mathbf{x}) + \frac{1}{|P|^2} \sum_{\mathbf{x}_i \in P} \sum_{\mathbf{x}_j \in P} k(\mathbf{x}_i, \mathbf{x}_j) - \frac{2}{|P|} \sum_{\mathbf{x}_i \in P} k(\mathbf{x}, \mathbf{x}_i)$

8. assign each point $\mathbf{x}_j$ to the partition P_i associated to the nearest centroid $\mathbf{c}_i$.

9. if $< \epsilon$ **END**. Else return to 6.

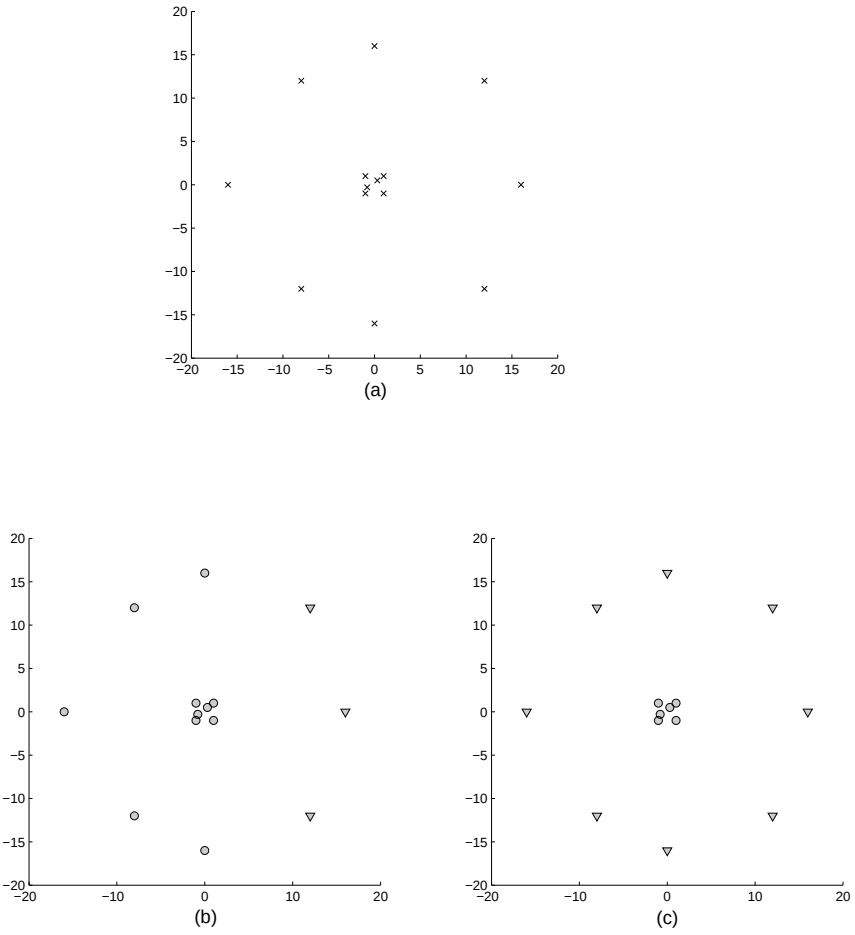


Figure 5.3: Clustering Results of the set (a) using (b):classic k-means algorithm, (c): kernel k-means with a polynomial kernel (P=2).

5.6 Designing and Combining Kernels

There are many approaches in order to create new kernels. One of them is, for example, starting from a feature space ϕ and then using it to find the corrisponding kernel. By converse one can directly create a new kernel and then check its positive definiteness.
A new kernel can be generated from a combinations of existing kernels. The simplest way is a weighted sum of them:

$$k = \sum_{i=1}^{c} \mu_i k_i \tag{5.27}$$

where k_i $(i = 1, ..., c)$ are different kernels and μ_i are opportune weights.
Other techniques for costructing new kernels, given two valid kernels k_1 and k_2 on a set χ are the following:

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') && (5.28)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') && (5.29)\\
k(\mathbf{x}, \mathbf{x}') &= \exp(k_1(\mathbf{x}, \mathbf{x}')) && (5.30)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') && (5.31)\\
k(\mathbf{x}, \mathbf{x}') &= k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) && (5.32)
\end{aligned}$$

where c is a positive constant, f is a generic function, ϕ is a function from χ to $\mathbb{R}^p$ and k_3 is a valid kernel for vectors.

5.7 Translation Invariant Kernels

In section 5.1.3 we have introducted the RBF Gaussian kernel and we have shown that it can be seen as a proper similarity measure rather than other kernels because its value depends by the distance between the objects rather than the objects themselves. Kernels that have this property are called Translation Invariant kernels. Given $\chi = \mathbb{R}^p$, a kernel is said transational invariant if for some function $\psi : \mathbb{R}^p \to \mathbb{R}$ we have

$$k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x} - \mathbf{x}') \tag{5.33}$$

for any $\mathbf{x}, \mathbf{x}' \in \chi$. If we change the euclidean distance used in the gaussian kernel with a measure of distance in another space (even a feature space of another kernel), it is always a translation invariant-kernel but it measure the similarity in the new space.

Chapter 6

The Euler Kernel

In this chapter we introduce a kernel that assumes complex values: the Euler Kernel.
In [20] Liwicki, Tzimiropoulos, Zafeiriou and Pantic presented a new implementation of Principal Component Analysis algorithm that uses a different dissimilarity measure between vectors than the l_2-norm. This measure, introducted by Fitch in [10], is based on the Euler representation of complex numbers and results, at least in the classes of problem esaminated in the paper, more robust:

$$d_e(\mathbf{x}_j, \mathbf{x}_q) = \sum_{c=1}^{p} \{1 - cos(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c)))\} \tag{6.1}$$

where $\mathbf{x}_j$ and $\mathbf{x}_q$ are p-dimensional vectors with values in the range [0,1] and a parameter $\alpha \in \mathbb{R}^+$. This dissimilarity measure is equivalent to the squared $l_2 - norm$ applied to the difference between $\mathbf{z}_j$ and $\mathbf{z}_q$, projections of the original vectors in a p-dimensional complex space with this mapping:

$$\begin{aligned} \mathbf{z}_j = \phi(\mathbf{x}_j) &= \frac{1}{\sqrt{2}} e^{\imath\alpha\pi\mathbf{x}_j} = \\ &= \frac{1}{\sqrt{2}} [\cos(\alpha\pi\mathbf{x}_j) + \imath \sin(\alpha\pi\mathbf{x}_j)] \end{aligned} \tag{6.2}$$

We now prove that $||\mathbf{z}_j - \mathbf{z}_q||^2 = d_e(\mathbf{x}_j, \mathbf{x}_q)$:

$$
\begin{aligned}
&||z_j - z_q||^2 = \\
&= ||\frac{1}{\sqrt{2}}[\cos(\alpha\pi\mathbf{x}_j) + i\sin(\alpha\pi\mathbf{x}_j)] - \frac{1}{\sqrt{2}}[\cos(\alpha\pi\mathbf{x}_q) + i\sin(\alpha\pi\mathbf{x}_q)]||^2 = \\
&= \frac{1}{2}||\cos(\alpha\pi\mathbf{x}_j) + i\sin(\alpha\pi\mathbf{x}_j) - \cos(\alpha\pi\mathbf{x}_q) - i\sin(\alpha\pi\mathbf{x}_q)||^2 = \\
&= \frac{1}{2}\sum_{c=1}^{p} 2 - 2\cos(\alpha\pi\mathbf{x}_j(c))\cos(\alpha\pi\mathbf{x}_q(c)) - 2\sin(\alpha\pi\mathbf{x}_j(c))\sin(\alpha\pi\mathbf{x}_q(c)) = \\
&= \sum_{c=1}^{p}\{1 - \cos(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c)))\} = d_e(\mathbf{x}_j, \mathbf{x}_q)
\end{aligned}
$$

The next graph shows how the distance in the complex space changes while changing difference between two points in the original space, using different value of α:

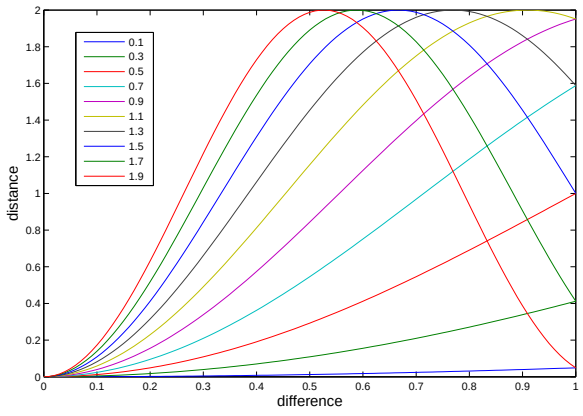


Figure 6.1: Cosine dissimilarity measure with changing α

by normalizing all distances we have:

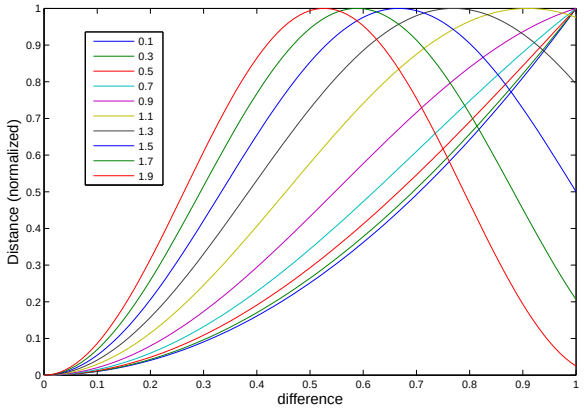


Figure 6.2: Normalized Cosine dissimilarity measure with changing α

We can see that when the value of α is small enough the behaviour of the fuction is very similar to the l_2-norm but with higher values of α the maximum distance does not coincide with the maximum difference anymore and so very different values (using an opportune α) can be less distant than more similar values. As a title of example, using $\alpha = 1.9$ two values whose difference is 0.5 are more distant in the complex space than two values whose difference is 0.8 (see figures 6.3 and 6.4).

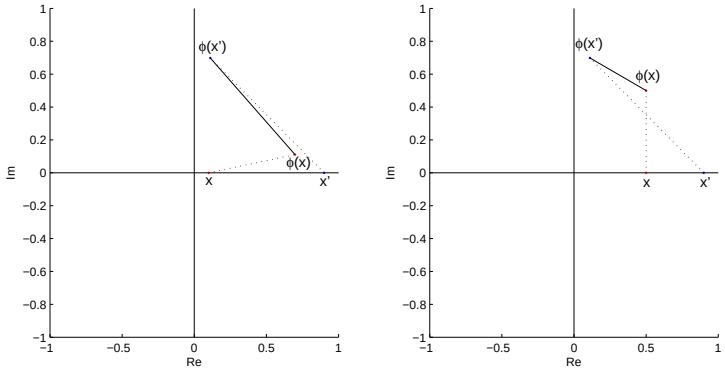


Figure 6.3: Comparison of Distance between points x and x' and the distance between their projections in the complex space using $\alpha = 0.5$

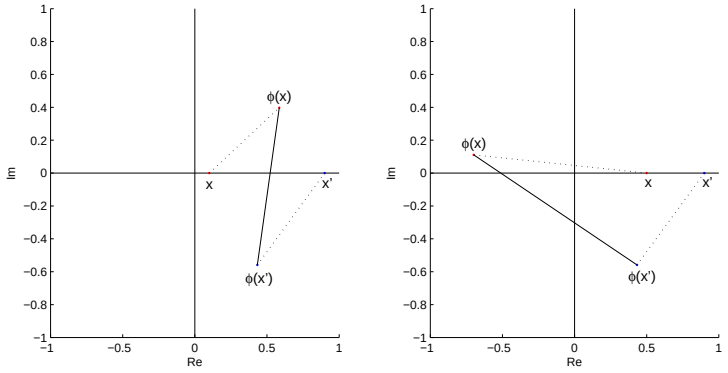


Figure 6.4: Comparison of Distance between points x and x' and the distance between their projections in the complex space using $\alpha = 1.9$

6.1 The Proper Euler Kernel

The Euler kernel can be generated by solving the dot product of $\mathbf{z}_j$ and $\mathbf{z}_q$ in terms of $\mathbf{x}_j$ and $\mathbf{x}_q$. We now are in a multidimensional complex space so the dot product is between the complex conjugate of $\mathbf{z}_j$ ($\mathbf{z}_j^\dagger$) and $\mathbf{z}_q$:

$$\begin{aligned}
\langle \mathbf{z}_q, \mathbf{z}_j \rangle &= \mathbf{z}_q^\dagger \mathbf{z}_j = \\
&= \frac{1}{2}\sum_{c=1}^{p}[\cos(\alpha\pi\mathbf{x}_j(c)) - i\sin(\alpha\pi\mathbf{x}_j(c))][\cos(\alpha\pi\mathbf{x}_q(c)) + \\
&\quad + i\sin(\alpha\pi\mathbf{x}_q(c))] = \\
&= \frac{1}{2}\sum_{c=1}^{p}[\cos(\alpha\pi\mathbf{x}_j(c))\cos(\alpha\pi\mathbf{x}_q(c)) + \sin(\alpha\pi\mathbf{x}_j(c))\sin(\alpha\pi\mathbf{x}_q(c)) - \\
&\quad + i(\sin(\alpha\pi\mathbf{x}_j(c))\cos(\alpha\pi\mathbf{x}_q(c)) - \cos(\alpha\pi\mathbf{x}_j(c))\sin(\alpha\pi\mathbf{x}_q(c)))] =
\end{aligned}$$

and finally

$$k(\mathbf{x}_j, \mathbf{x}_q) = \frac{1}{2}\sum_{c=1}^{p}\cos(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c))) - i\frac{1}{2}\sum_{c=1}^{p}\sin(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c))) \tag{6.3}$$

6.2 Other Properties of Euler Kernels

The Euler Kernel has some interesting properties that can be exploited to compute efficently what we need. First of all we have seen that the complex Hilbert space defined by the kernel has the same dimensionality of the original space: this fact implies that we can always explicitely project the data in the complex space if it results to be convenient. Another important property consists in the fact that each component of the projected vector depends only by an unique component of the original vector in data space. The next proposition shows that all the points projected in the space defined by euler kernel have the same norm value:

Proposition 6.1.
Given a real valued vector $x \in \mathbb{R}^p$ its projection in the complex space $\phi(x)$ defined by an Euler Kernel has euclidean squared norm equal to p/2.

Proof:

$$\begin{aligned} ||\phi(x)||^2 &= \frac{1}{2}|||[\cos(\alpha\pi\mathbf{x}_j) + \imath\sin(\alpha\pi\mathbf{x}_j)]||^2 = \\ &= \frac{1}{2}\{|||[\cos(\alpha\pi\mathbf{x}_j)||^2 + ||\sin(\alpha\pi\mathbf{x}_j)]||^2\} = \\ &= \frac{1}{2}\sum_{c=1}^{p}(\cos^2(\alpha\pi\mathbf{x}(c)) + \sin^2(\alpha\pi\mathbf{x}(c))) = \\ &= \frac{p}{2} \end{aligned} \tag{6.4}$$

□

In the next section we will exploit this property to compute distances between points in this complex vector space.

6.3 Kernel Trick for Distances for Complex Kernels

The kernel trick for distance in chapter 3 works only for real valued kernels: in the complex case the trick is different and it does not use directly the complex kernel but two partial kernels that correspond respectively to the dot product of the real and imaginary parts of the projections of the data

in the complex space:

$$\begin{aligned} d(\mathbf{z}_j, \mathbf{z}_q)^2 &= ||\mathbf{z}_j - \mathbf{z}_q||^2 = \\ &= ||Re(\mathbf{z}_j) - Re(\mathbf{z}_q)||^2 + ||Im(\mathbf{z}_j) - Im(\mathbf{z}_q)||^2 = \\ &= \sum_{k=1}^{p} Re(\mathbf{z}_j(k))^2 + \sum_{k=1}^{p} Re(\mathbf{z}_q(k))^2 - 2\sum_{k=1}^{p} Re(\mathbf{z}_j(k))Re(\mathbf{z}_q(k)) + \\ &\quad + \sum_{k=1}^{p} Im(\mathbf{z}_j(k))^2 + \sum_{k=1}^{p} Im(\mathbf{z}_q(k))^2 - 2\sum_{k=1}^{p} Im(\mathbf{z}_j(k))Im(\mathbf{z}_q(k)) = \\ &= k_r(\mathbf{x}_j, \mathbf{x}_j) + k_r(\mathbf{x}_q, \mathbf{x}_q) + k_r(\mathbf{x}_q, \mathbf{x}_q) - 2k_r(\mathbf{x}_j, \mathbf{x}_q) + \\ &\quad + k_i(\mathbf{x}_j, \mathbf{x}_j) + k_i(\mathbf{x}_q, \mathbf{x}_q) + k_i(\mathbf{x}_q, \mathbf{x}_q) - 2k_i(\mathbf{x}_j, \mathbf{x}_q) \end{aligned} \tag{6.5}$$

where

$$\begin{aligned} k_r(\mathbf{x}_j, \mathbf{x}_q) &= Re(\mathbf{z}_j)^T Re(\mathbf{z}_q) = \\ &= \frac{1}{4}\sum_{c=1}^{p}\{\cos(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c))) + \cos(\alpha\pi(\mathbf{x}_j(c) + \mathbf{x}_q(c)))\} \end{aligned} \tag{6.6}$$

$$\begin{aligned} k_i(\mathbf{x}_j, \mathbf{x}_q) &= Im(\mathbf{z}_j)^T Im(\mathbf{z}_q) = \\ &= \frac{1}{4}\sum_{c=1}^{p}\{\cos(\alpha\pi(\mathbf{x}_j(c) - \mathbf{x}_q(c))) - \cos(\alpha\pi(\mathbf{x}_j(c) + \mathbf{x}_q(c)))\} \end{aligned} \tag{6.7}$$

so it becomes

$$d(\mathbf{z}_j, \mathbf{z}_q)^2 = p - 2k_r(\mathbf{x}_j, \mathbf{x}_q) - 2k_i(\mathbf{x}_j, \mathbf{x}_q) \tag{6.8}$$

Hence in order to compute the distances between points in the Hilbert complex space defined by the Euler kernel we don't need to compute the Euler kernel itself.

6.4 Euler K-Means

The Euler k-means is a implementation of Kernel K-means that uses the Euler kernel. It exploits the properties of the euler kernel in order to make the computation more efficient. First of all it doesn't create a gram matrix but it projects explicitely the objects in the complex space defined by the Euler kernel that has the same dimension of the original data and so it doesn't increase the computational complexity (that is comparable to the classic K-means). Second, getting rid of the kernel trick permits to explicitely compute the centroids:

$$\begin{aligned}\mathbf{c}_i &= \frac{1}{|P_i|}\sum_{\mathbf{x}\in P_i}\phi(\mathbf{x}) = \\ &= \frac{1}{\sqrt{2}|P_i|}\sum_{\mathbf{x}\in P_i}[\cos(\alpha\pi\mathbf{x}) + i\sin(\alpha\pi\mathbf{x})] = \\ &= \frac{1}{\sqrt{2}|P_i|}(\sum_{\mathbf{x}\in P_i}\cos(\alpha\pi\mathbf{x})) + i\sum_{\mathbf{x}\in P_i}\sin(\alpha\pi\mathbf{x}))\end{aligned}$$

By setting $\mathbf{a} = \frac{1}{|P_i|}\sum_{\mathbf{x}\in P_i}\cos(\alpha\pi\mathbf{x})$ and $\mathbf{b} = \frac{1}{|P_i|}\sum_{\mathbf{x}\in P_i}\sin(\alpha\pi\mathbf{x})$ we have

$$\mathbf{c}_i = \frac{1}{\sqrt{2}}(\mathbf{a} + i\mathbf{b}) \tag{6.9}$$

Consequently, even the computation of the squared distance between points and centroids becomes simpler respect the classic kernel k-means:

$$\begin{aligned}d(\phi(\mathbf{x}_j), \mathbf{c}_i)^2 &= ||\phi(\mathbf{x}_j) - \mathbf{c}_i||^2 = \\ &= ||\frac{\cos(\alpha\pi\mathbf{x}_j) - \mathbf{a}}{\sqrt{2}} + i\frac{\sin(\alpha\pi\mathbf{x}_j) - \mathbf{b}}{\sqrt{2}}||^2 = \\ &= ||\frac{\cos(\alpha\pi\mathbf{x}_j) - \mathbf{a}}{\sqrt{2}}||^2 + ||\frac{\sin(\alpha\pi\mathbf{x}_j) - \mathbf{b}}{\sqrt{2}}||^2 = \\ &= \frac{p}{2} + ||\mathbf{c}_i||^2 - \cos(\alpha\pi\mathbf{x}_j)^T\mathbf{a} - \sin(\alpha\pi\mathbf{x}_j)^T\mathbf{b}\end{aligned} \tag{6.10}$$

where p is the number of dimensions of the data space. The norms of the centroids are not equal to $p/2$ because they are not projections of data objects and so in this case the proposition 6.1 is not valid.
The Euler k-means algorithm works as the classic k-means, except that computes centroids and distances using respectively (4.5) and (4.6). At each iteration the algorithm monotonically decreases the objective function

$$f = \sum_{i=1}^{K}\sum_{\mathbf{x}_j\in P_i} d(\phi(\mathbf{x}_j), \mathbf{c}_i)^2 \tag{6.11}$$

after updating the centroids and reassigning labels for data points.

6.5 Euler Dominant-Set Clustering

We have seen that Euler k-means get rid of the kernel trick and projects explicitely the data in the complex space defined by the kernel. Dominant-Set clustering algorithm needs pairwise similarity matrix in order to work but it can't use directly the Euler kernels for two important reasons:

1. it is not able to manage a payoff matrix of complex numbers (the similarity measure has to assume only real values)

2. the kernel is not transactional invariant or normalized, so it is not a proper measure of similarity.

We can use the Euler kernel as start point in order to create a new kernels that that have these properties.

6.5.1 Euler-Gaussian translation invariant Kernel

The normalization of the Euler Kernel do not avoid complex values so it is useless if we want to use it to create a payoff matrix. By converse, the distances between points in the complex space assume real values and it is this measure the effective cosine based dissimilarity used in both Euler-PCA and Euler K-means. In Euler Dominant-Set we want to preservate this information but we have to transform it in a similarity measure. In our implementation we have incapsulate the Euler dissimilarity in a Gaussian Kernel:

$$k_{eg}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{d_e(\mathbf{x}_i, \mathbf{x}_j)}{2\sigma^2}) \tag{6.12}$$

This kernel is transational invariant and assumes only real values. Finally we show the algorithm used to compute the payoff matrix for both Dominant-Set and Graph Transduction algorithms:

Computation of Euler payoff matrix

Input: a dataset of n objects $x_i \in \mathbb{R}$, parameters $\alpha, \sigma > 0$

1. for each $\mathbf{x}_i$ compute is projection in the complex space $\mathbf{z}_i$ using (6.2).

2. split the rational and imaginary parts of each $\mathbf{z}_i$ and compute $k_r(\mathbf{x}_i, \mathbf{x}_j)$ and $k_i(\mathbf{x}_i, \mathbf{x}_j)$ using (6.6) and (6.7).

3. compute $d(\mathbf{z}_i, \mathbf{z}_j)^2$ using (6.8).

4. compute the payoff matrix $A = (k_{eg}(\mathbf{x}_i, \mathbf{x}_j))_{ij}$ using (6.12).

5. for each $i = 1...n$, $A_{ii} = 0$.

Chapter 7

Experiments

We show now some of the tests performed in order to evaluate the performances of the techniques introducted in the previous chapters. The tests are divided in three main categories:

1. Performance of the Dominant-Set clustering using graph transduction.

2. Performance of the Dominant-Set clustering using Euler-Gaussian Kernels.

3. Performance of the Dominant-Set clustering using both graph transduction and Euler-Gaussian Kernels.

There are many ways to evaluate the goodness of a clustering, in this chapter we use an indicator called Normal Mutual Information (NMI), that can be used when we know the true class labels of the data objects (see [5]). Let $n_i^{(j)}$ be the number of points in cluster i and class j, c and k respectively the number of classes and the number of clusters, the NMI is defined as follows:

$$NMI = \frac{2\sum_{l=1}^{k}\sum_{h=1}^{c}\frac{n_l^{(h)}}{n}\log\frac{n_l^{(h)}n}{\sum_{i=1}^{k}n_i^{(h)}\sum i{=}1^{c}n_l^{(i)}}}{H(\pi)+H(\zeta)} \tag{7.1}$$

where

$$H(\pi) = -\sum_{i=1}^{k}\frac{n_i}{n}\log\frac{n_i}{n} \tag{7.2}$$

$$H(\zeta) = -\sum_{j=1}^{c}\frac{n^{(j)}}{n}\log\frac{n^{(j)}}{n} \tag{7.3}$$

NMI can assume values from 0 to 1: the higher the value is better is the cluster quality.

7.1 Dominant Sets Clustering using graph transduction

7.1.1 Toy datasets

In this section we use dominant set clustering to extract two clusters from each dataset and then we connect the noise to the clusters using two different methods:

1. The points are connected to the class of the nearest labeled points (traditional method);
2. The information of membership is propagated from labeled points to unlabeled points using graph transduction.

The objects in the toy datasets are 2-dimensional vectors so they can be plotted. In order to create the similarity matrix for the dominant set clustering and graph transduction we use the RBF Gaussian Kernel (with different σ):

$$S_{ij} = k(i,j) = \exp(\frac{||x_i - x_j||^2}{2\sigma^2}) \tag{7.4}$$

We can see that while using distances the clusters continue to have globular shapes, the use of graph transduction permits to model better the real shapes of clusters. More the real clusters tend to be spherical, more the workload (measured by the number of point classified)of the dominant-set clustering is greater respect those of the graph transduction: indeed we can see that they are almost completely clustered in the first phase (i.e the inner cluster in Ring Dataset and upper one in the Moon Dataset). In the case of very complex-shaped real clusters (i.e. the two in the Spiral Dataset) in the first phase only few points are clustered and then the infomation is propagated to the others using GT.

Arcs Dataset

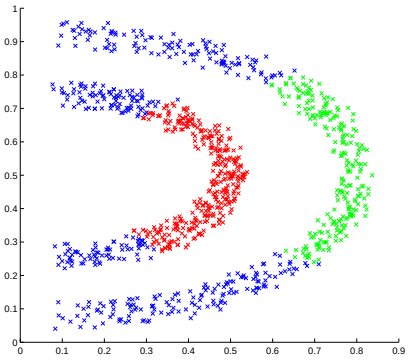


Figure 7.1: Dominant-Set clustering results using $\sigma = 2$.

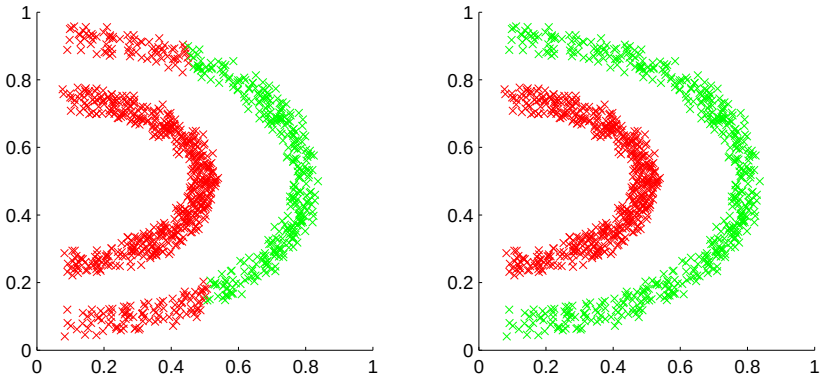


Figure 7.2: (a)Partition using distances; (b) Partition using GT ($\sigma = 0.01$)

In this dataset the two real clusters are arc-shaped, DS algorithm find a cluster in the "peak" of each arc, where the points are a bit more dense. Using distances to label the remaining points the inner cluster acquires even the "pillars" of the outer one. The graph transduction labels all the points correctly. The workload between DS-clustering and GT is well distributed.

Moon Dataset

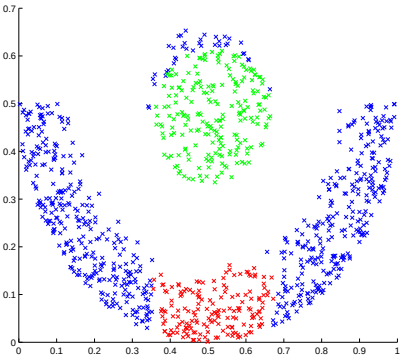


Figure 7.3: Dominant-Set clustering results using $\sigma = 1$.

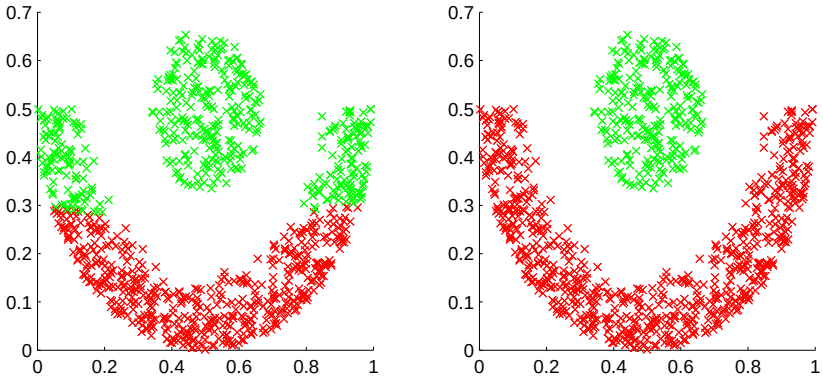


Figure 7.4: (a)Partition using distances; (b) Partition using GT ($\sigma = 0.01$)

In this dataset we have two real clusters with different shapes: one is spherical and the other is arc-shaped. Dominant-Set clustering extracts almost completely the first group and the "peak" of the second one. Even in this case, using distances, the inner cluster takes points belonging to the second real cluster. In order to label the spherical cluster the main part of work is done by DS clustering while for the arc-shaped one

Ring Dataset

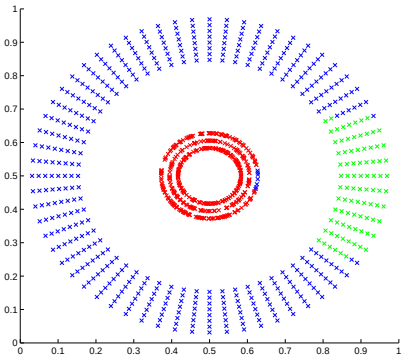


Figure 7.5: Dominant-Set clustering results using $\sigma = 1$.

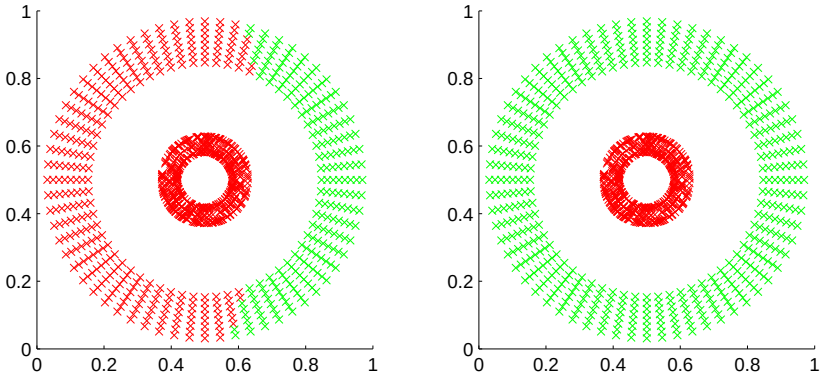


Figure 7.6: (a)Partition using distances; (b) Partition using GT ($\sigma = 0.01$)

We have now two ring-shaped sets of points: the first cluster extracted by Dominant-Set clustering cover almost all the inner ring; the second cluster takes only a part of the other ring (in this test we keep σ fixed but we can modify this parameter in this case we are able to correctly cluster all the dataset using only DS). The use of distance to label the other points makes the most of the objects of the outer ring parts of the "inner ring cluster".

Spiral Dataset

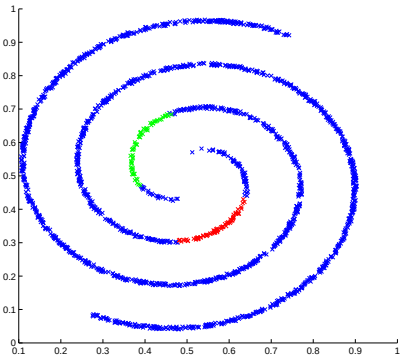


Figure 7.7: Dominant-Set clustering results using $\sigma = 0.5$.

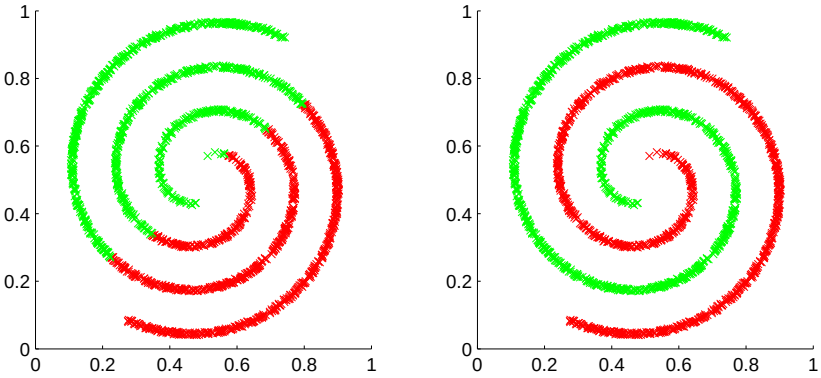


Figure 7.8: (a)Partition using distances; (b) Partition using GT ($\sigma = 0.01$)

Here the two subsets of points are shaped as concentric spirals: the DS-algorithm was able to extract only small clusters. The main work is done by using distances or graph transduction: in the first case the data space is "cutted" in two parts while in the second the information of the labels is propagated correctly.

7.1.2 Datasets from UCI machine learning repository

In this section we compare the clustering performances of DS clustering using graph transduction and distances in dataset from the UCI machine learning repository. Even in this case we create the similarity matrices using the Gaussian Kernel. For each dataset we measure the Normalized Mutual Information changing σ_1 (the parameter of the Gaussian Kernel used by DS clustering) and σ_2 (the parameter of the Gaussian Kernel used by the graph transduction algorithm). In order to compute kernels the objects chosen datasets contain only real-valued attributes. We give now some information about the Dataset used:

- **Iris Dataset:** this dataset contains objects representing physical features of 150 iris flower samples divided in three groups (the species of iris), each instance has 4 real-valued attributes.
- **Ionosphere Dataset:** radar data collected by a system in Goose Bay, Labrador. The dataset has 351 objects, each of them with 34 real and integer-valued attributes. Data are divided in two groups: "good" signals and "bad" signals.
- **Wine Dataset:** chemical analysis results of wines grown in the same region in Italy but derived from three different cultivars. The dataset is composed by 178 istances with 13 real/integer valued attributes. Objects are splitted in three parts (the cultivar).
- **Glass Identification Dataset:** chemical analysis results of glass samples: in this dataset where are 214 objects, each of them with 10 different real-valued features. Data are divided in six groups: the type of glass.
- **Lung Cancer Dataset:** an set of 32 instance describing 3 types of pathological lung cancer with 56 attributes (not defined).

Setup

The objects in the datasets are normalized before being clustered. The normalization consists in making the range of all the attributes to be between 0 and 1. For each set of objects DS-algorithm extracts c clusters, where c is the number of real groups of datapoints.

Iris Dataset

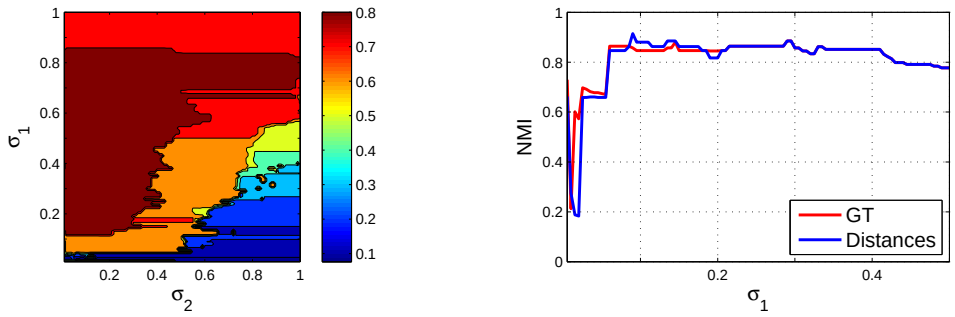


Figure 7.9: (a) NMI changing σ_1 and σ_2. (b) comparison between graph transduction and distances changing σ_1.

In the Iris Dataset we can increase σ_1 and so increase the number of objects clustered by DS-clustering without significative decreasing of the NMI: this fact may suggest that the real clusters have a spherical shape and so graph transduction does not improve the clustering.

Ionosphere Dataset

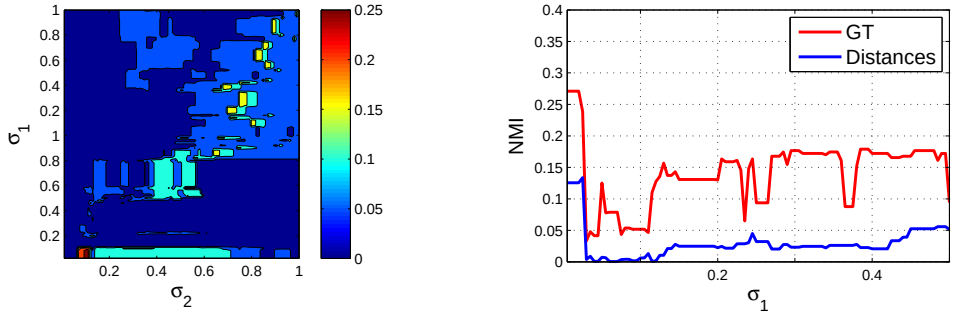


Figure 7.10: (a) NMI changing σ_1 and σ_2. (b) comparison between graph transduction and distances changing σ_1.

In the Ionosphere Dataset we have a completely inverse case: the NMI is (relatively) high only with very small σ_1: clearly the real clusters are not spherical shaped and so it is the graph tranduction that do the most of the work. GT-algorithm achieves a NMI that is about the two times those obtained by using distances.

Wine Dataset

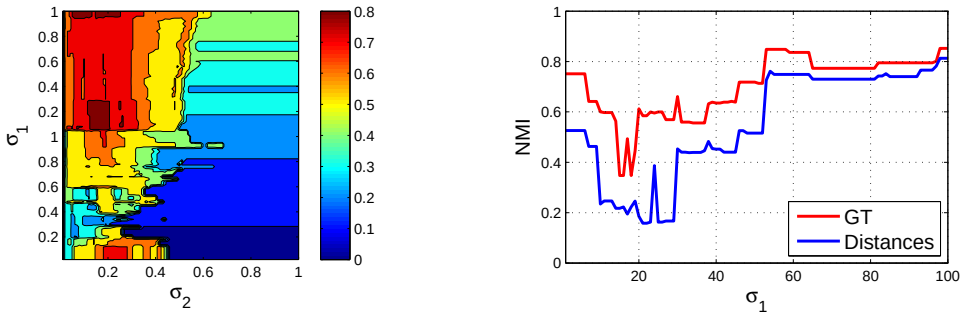


Figure 7.11: NMI using $\sigma_1 = 0.5$ and changing σ_2.

We have a sort of middleway between the two previous cases, hence it implies that the three real clusters have more regular shapes than those of Ionosphere set but not so regular as Iris set. The GT-algorithm obtains always a higher NMI than using distances even if the difference is less accentuated.

Glass Identification Dataset

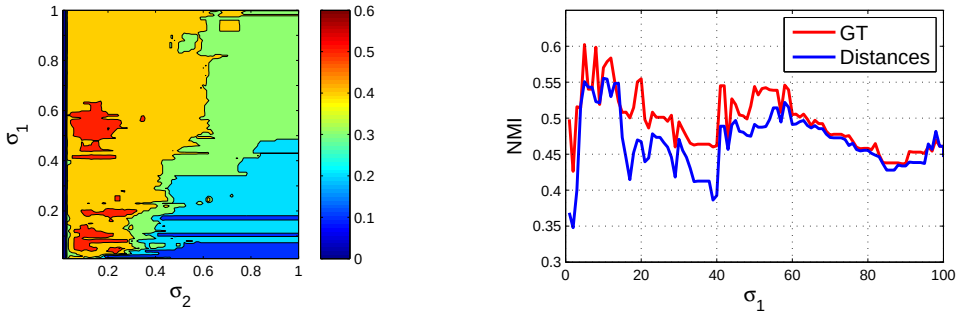


Figure 7.12: (a) NMI changing σ_1 and σ_2. (b) comparison between graph transduction and distances changing σ_1.

Here we are in a case between Ionosphere and Wine datasets: when NMI is higher it is the graph transduction that have the main workload. Note that in this dataset very small values of σ_2 drammatically decrease the performances of the clustering.

Lung Cancer Dataset

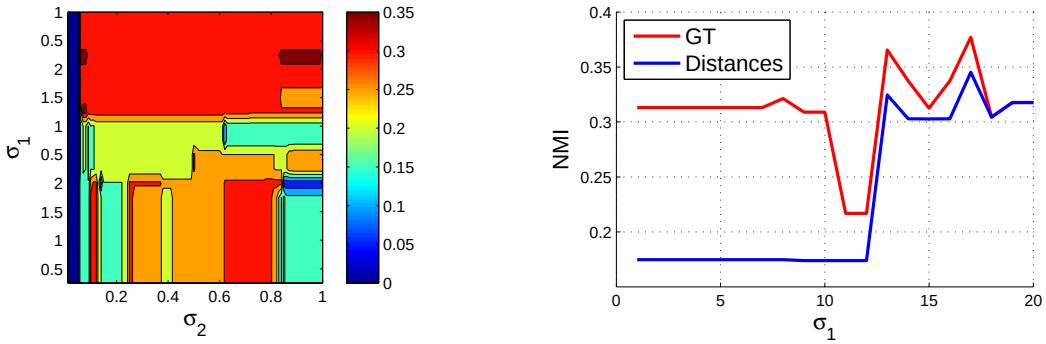


Figure 7.13: (a) NMI changing σ_1 and σ_2. (b) comparison between graph transduction and distances changing σ_1.

In this dataset the use of graph transduction increases the performances in a more relevant way when the clusters are small, even if using a higher σ_1 and so extracting bigger dominant sets we have the maximum NMI.

Final considerations

The clustering results obtained using graph transduction ovecome those achieved using distances except in Iris Dataset, probably because the real clusters have a spherical shape and so the graph transduction is not very useful. When the real clusters are very irregular, as in the Ionosphere Dataset GT improves more the clustering.
In the following table we resume the results of the tests, best results for each dataset are in bold:

Dataset	graph transduction	Distances
Iris	0.89	**0.91**
Ionosphere	**0.27**	0.13
Wine	**0.85**	0.81
Glass Id.	**0.60**	0.55
L. Cancer	**0.38**	0.34

Table 6.1: NMI using graph transduction and distances.

7.2 Dominant Set Clustering using Euler-Gaussian Kernels

7.2.1 Datasets from UCI machine learning repository

We now compare the clustering performances of Dominant Set Clustering using the Euler-Gaussian Kernels and Gaussian Kernels to compute similarity matrix in the same datasets from UCI Machine Learning Repository used to test the graph transduction.

For each dataset we compute the NMI and the percentage of labeled objects using Euler-Gauss Dominant Set changing α (the parameter of the Euler Distance) and σ (parameter of the Gaussian Kernel) and the same measures using Gaussian Kernel changing σ. Finally we compare the maximum Normalized Mutual Information of the two approaches changing the number of clustered objects.

Setup

Even in this tests the objects are normalized and DS-algorithm extracts as many clusters as the real groups of objects for each dataset. The iteratively extracted clusters are pelled off from the dataset, so there is no overlapping.

Iris Dataset

Euler-Gaussian Kernel

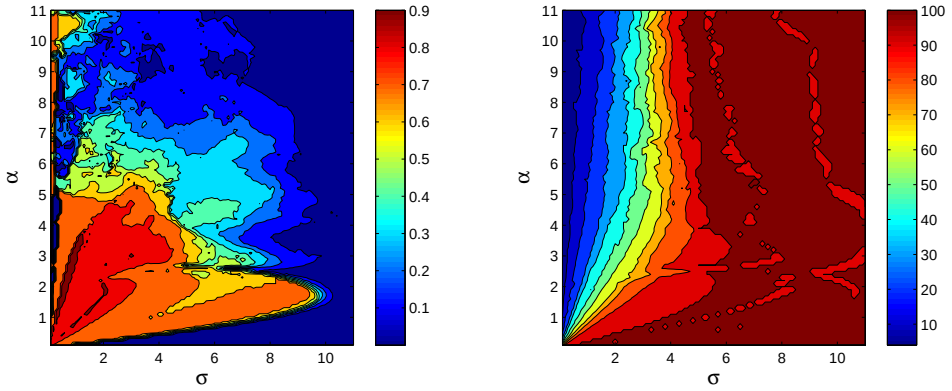


Figure 7.14: (a)NMI changing α and σ. (b) Number of labeled objects (%) changing α and σ.

Gaussian Kernel

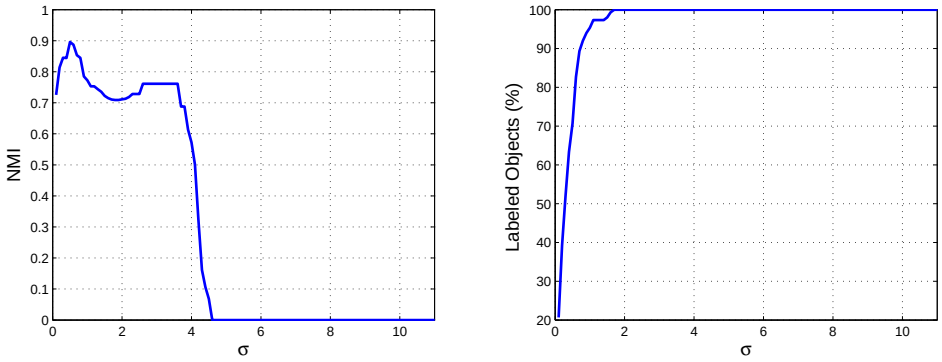


Figure 7.15: (a)NMI changing σ. (b) Number of labeled objects (%) changing σ.

Comparison

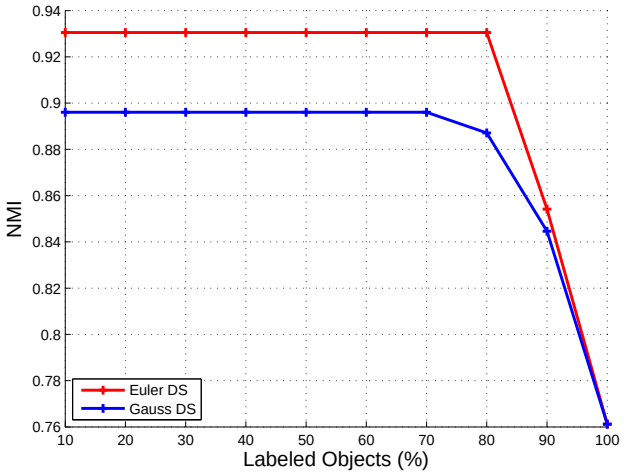


Figure 7.16: NMI changing the % of the labeled objects.

In Iris Dataset the Euler-Gaussian Kernel tends to extract purest clusters when α is small and so the behavior of the kernel is similar to the Gaussian one. Performances of EG-kernel are always better than Gauss Kernel, but they are equivalent when the clusters extracted cover the entire dataset.

Ionosphere Dataset

Euler-Gaussian Kernel

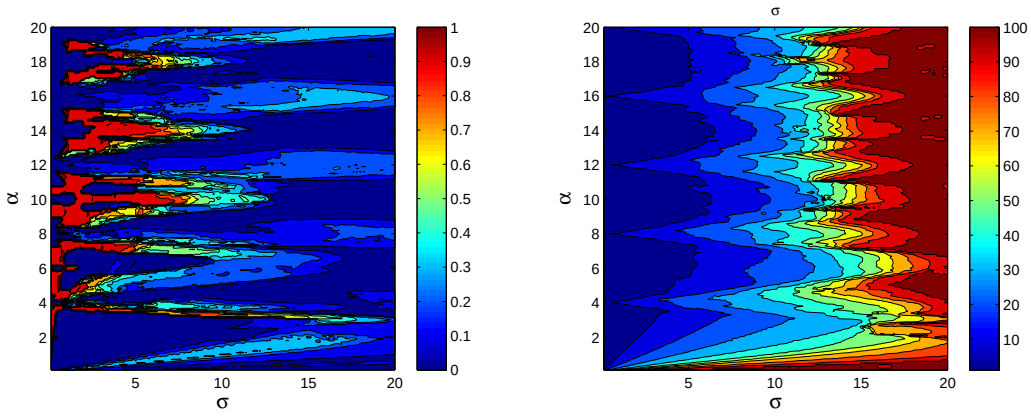


Figure 7.17: (a)NMI changing α and σ. (b) Number of labeled objects (%) changing α and σ.

Gaussian Kernel

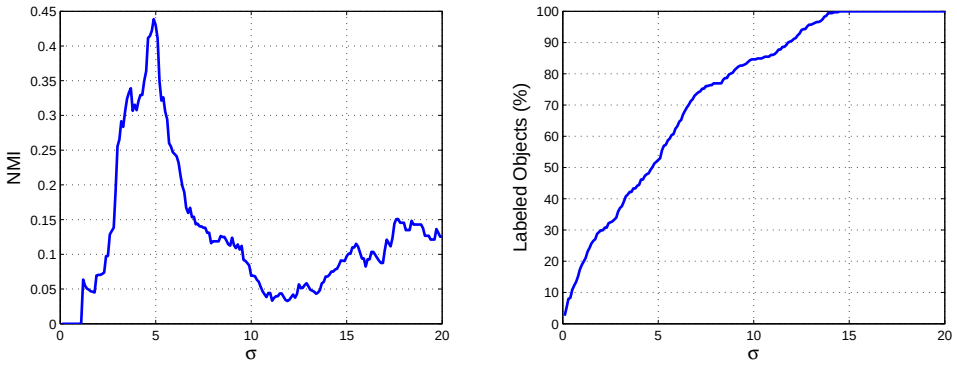


Figure 7.18: (a)NMI changing σ. (b) Number of labeled objects (%) changing σ.

Comparison

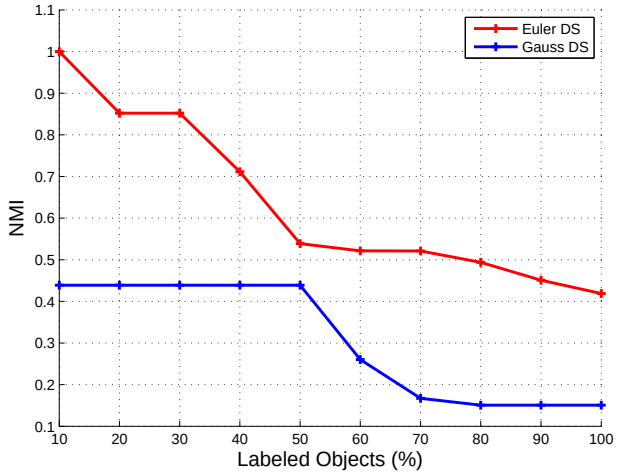


Figure 7.19: NMI changing the % of the labeled objects.

The real clusters of objects in Ionosphere dataset are less "regular" than those of the previous one: the clusters extracted using Euler-Gaussian Kernel are purer using a higher α: When the clustering covers the 10% of the data the NMI is equal to 1, two times more than using Gaussian Kernels. At 100% the NMI of EG-kernel is almost three time higher.

Glass Dataset

Euler-Gaussian Kernel

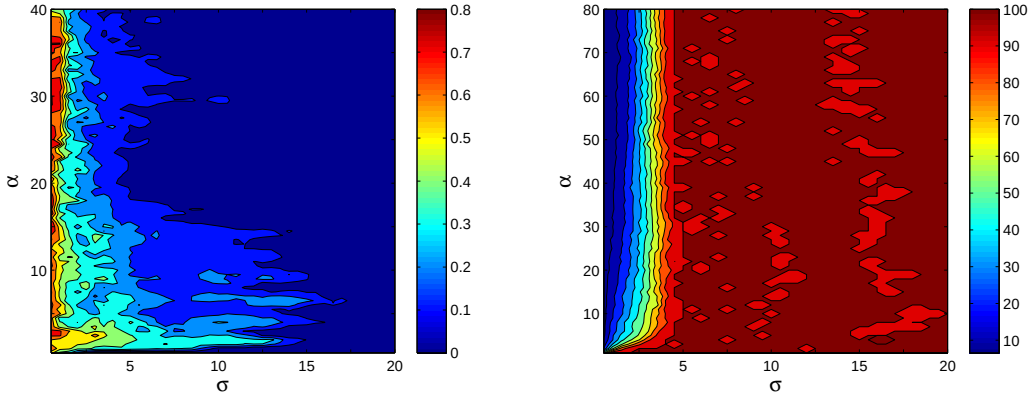


Figure 7.20: (a)NMI changing α and σ. (b) Number of labeled objects (%) changing α and σ.

Gaussian Kernel

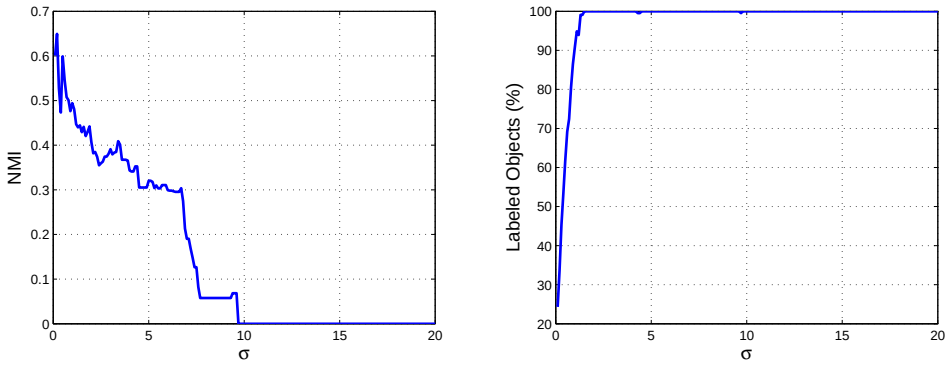


Figure 7.21: (a)NMI changing σ. (b) Number of labeled objects (%) changing σ.

Comparison

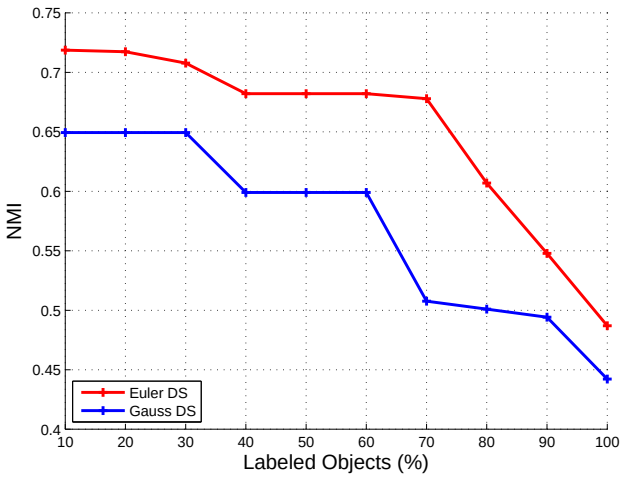


Figure 7.22: NMI changing the % of the labeled objects.

Wine Dataset

Euler-Gaussian Kernel

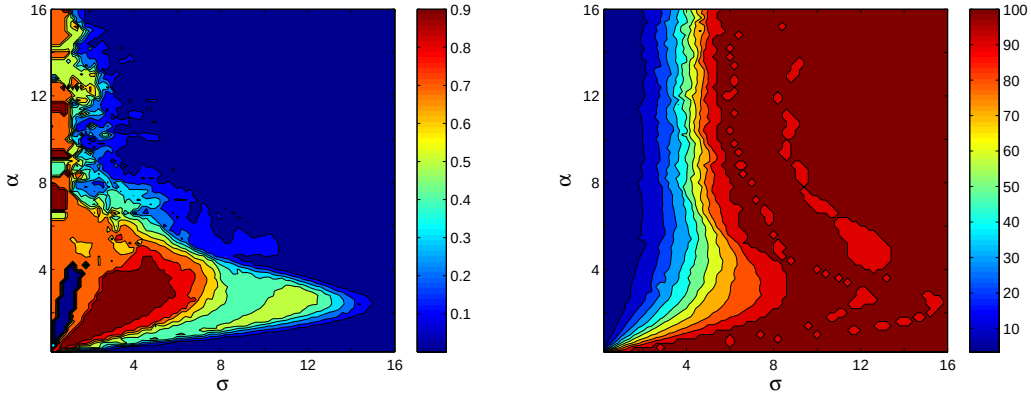


Figure 7.23: (a)NMI changing α and σ. (b) Number of labeled objects (%) changing α and σ.

Gaussian Kernel

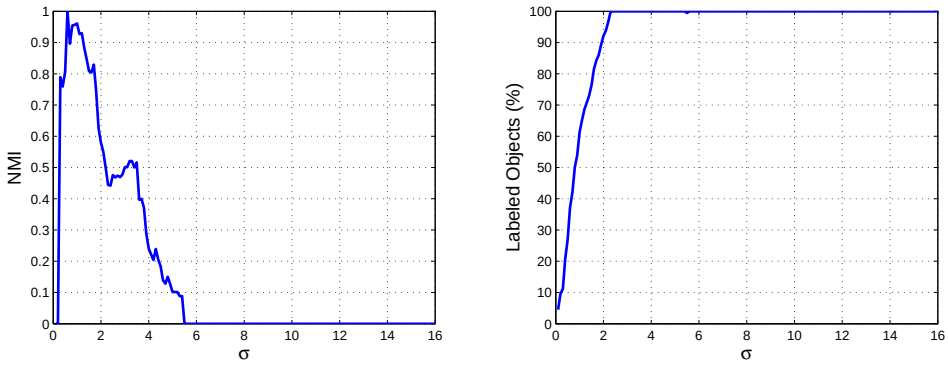


Figure 7.24: (a)NMI changing σ. (b) Number of labeled objects (%) changing σ.

Comparison

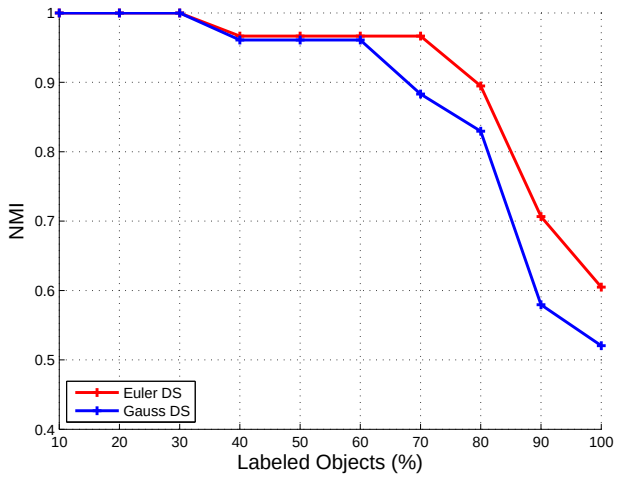


Figure 7.25: NMI changing the % of the labeled objects.

Lung Cancer Dataset

Euler-Gaussian Kernel

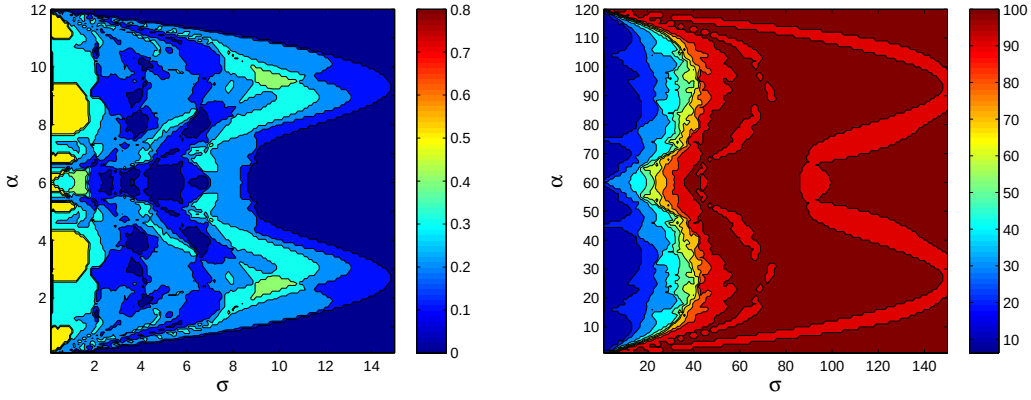


Figure 7.26: (a)NMI changing α and σ. (b) Number of labeled objects (%) changing α and σ.

Gaussian Kernel

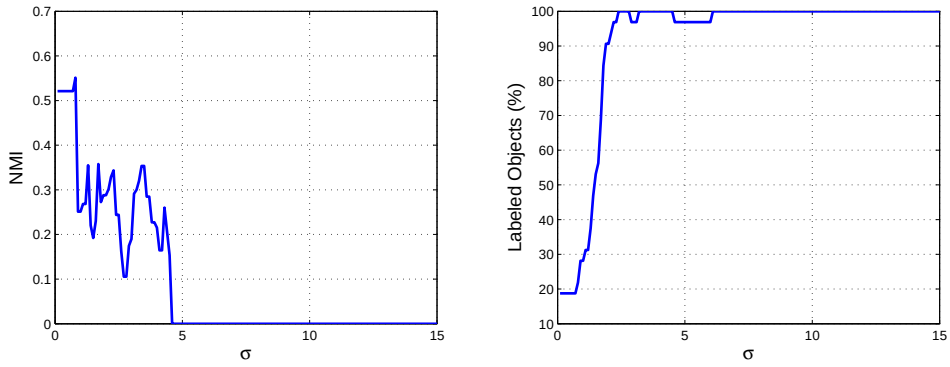


Figure 7.27: (a)NMI changing σ. (b) Number of labeled objects (%) changing σ.

Comparison

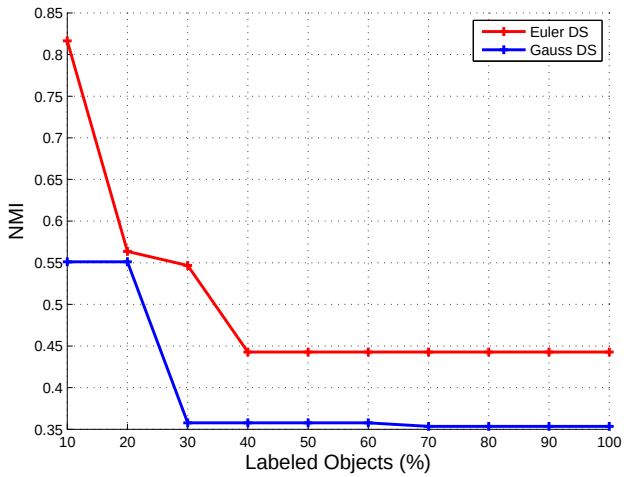


Figure 7.28: NMI changing the % of the labeled objects.

Final considerations

The use of Euler-Gaussian Kernels as measure of similarity achieves better performances than using Gaussian Kernels, in particular when the real clusters are very irregular. The robustness of the kernel permits, with fixed NMI, to create bigger clusters than using the Gaussian one.

In the next table we list the values of the normalized mutual information when the Dominant-Set algorithm labels all the objects in each dataset using the two kernels.

Dataset	Euler-Gauss kernel	Gauss kernel
Iris	**0.76**	**0.76**
Ionosphere	**0.42**	0.15
Wine	**0.60**	0.52
Glass Id.	**0.49**	0.44
L. Cancer	**0.44**	0.35

Table 6.2: NMI using Euler-Gaussian Kernel and Gauss kernel.

7.3 Dominant Set Clustering combining graph transduction and Euler-Gaussian Kernels

We now compare the clustering performances of Euler-DS clustering using graph transduction and distances in some image dataset. We create the similarity matrices using the Euler-Gaussian Kernel with $\alpha = 700$. We measure the Normalized Mutual Information (NMI) changing σ_1 (the parameter of the Gaussian Kernel used by Euler-DS clustering) and σ_2 (the parameter of the Gaussian Kernel used by the graph transduction algorithm). In these datasets the graph transduction considerably improves the performance of the clustering.

The image datasets are created as follows:

1. for each image we extract the Scale Invariant Feature Transform (SIFT) descriptors (see [21]), such objects are 128-dimensional vectors that describe image features invariant to image scaling, translation and rotation, and partially invariant to illumination changes and 3D projection.

2. We employ the Bag-of-Feature (see [19]) to get 512 prototypes of SIFT descriptors: this process consists in a K-means clustering over the SIFT keys extracted from all the images (or a subset if they are too many), each descriptor is associated to the nearest centroid.

3. We apply a three-level spatial pyramid modeling (see [17]): it consists in subdividing each image in three different levels of resolution: the first is the entire image, in the second the image is splitted in 4 parts and in the third in 16. For each level, we count how many SIFT descriptors fall in each spatial bin for each group of keys.

4. Finally the $512 \times (1 + 4 + 16) = 10752$-dimensional vector obtained for each image is normalized in order to have the sum of all the vector components equal to 1.

We give now a brief description of the image-datasets used:

- **Event Dataset:** it is composed by 1579 images regarding 8 different sport events: the size of the groups varies from 137 to 250.

- **13 Natural Scenes:** a set of 3859 pictures subdivided, as the name may suggest, in 13 natural scene categories. The cardinality of the subsets varies from 210 to 410 objects.

- **Caltech101 Dataset:** a set of 8677 images of objects subdivided in 101 categories with cardinality varying from 40 to 800.

SCENE13 Dataset

Euler-Gauss kernel

Euler-Gauss DS clustering with GT using $\sigma_1 = 29.9$ and $\sigma_2 = 4$ achieves $NMI \approx 0.59$. Without graph transaction it only obtains $NMI \approx 0.39$.

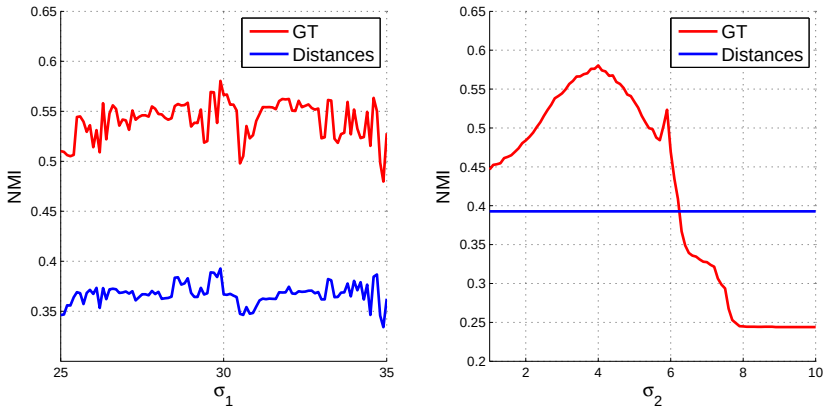


Figure 7.29: (a)NMI using $\sigma_2 = 4$ and changing σ_1; (b)NMI using $\sigma_1 = 29$ and changing σ_2.

Gauss kernel

Gauss-DS clustering with GT obtains $NMI \approx 0.31$ using $\sigma_1 = 25.2$ and $\sigma_2 = 4.2$. The maximum NMI using distances is ≈ 0.28.

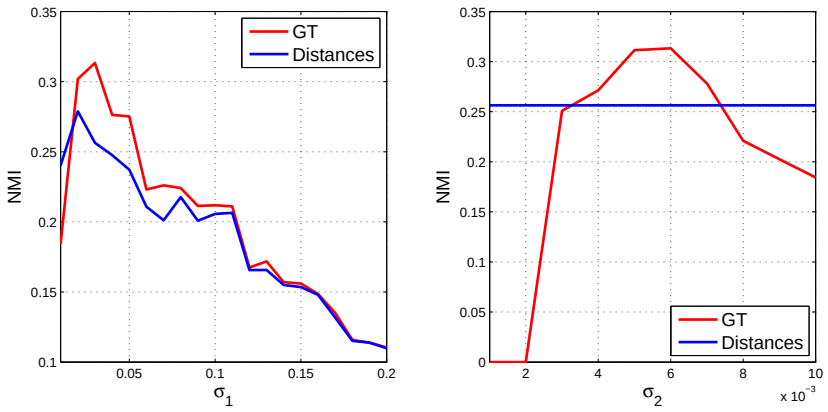


Figure 7.30: (a)NMI using $\sigma_2 = 0.006$ and changing σ_1; (b)NMI using $\sigma_1 = 0.03$ and changing σ_2.

EVENT Dataset

Euler-Gauss kernel

Euler-Gauss DS clustering with GT using $\sigma_1 = 25.2$ and $\sigma_2 = 4.2$ performs $NMI \approx 0.42$. Using distances the maximum NMI achieved is only 0.34.

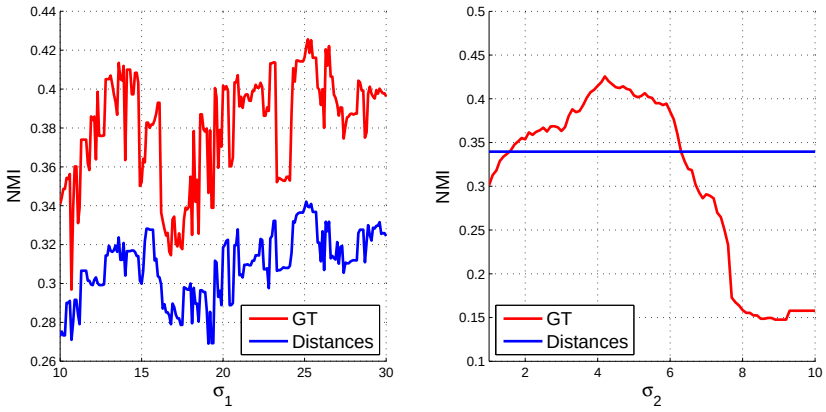


Figure 7.31: (a)NMI using $\sigma_2 = 4.2$ and changing σ_1; (b)NMI using $\sigma_1 = 25.2$ and changing σ_2.

Gauss kernel

Gauss-DS clustering with GT obtains an NMI slighty higher than 0.22 using $\sigma_1 = 0.03$ and $\sigma_2 = 0.006$.

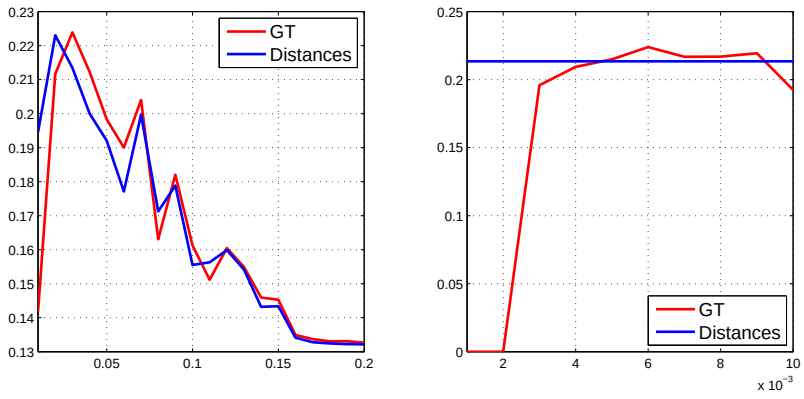


Figure 7.32: (a)NMI using $\sigma_2 = 0.006$ and changing σ_1; (b)NMI using $\sigma_1 = 0.03$ and changing σ_2.

Caltech101 Dataset

Euler-Gauss kernel

Euler-Gauss DS clustering with GT using $\sigma_1 = 9$ and $\sigma_2 = 4.1$ performs $NMI \approx 0.46$. The use of distances achieves only $NMI \approx 0.37$.

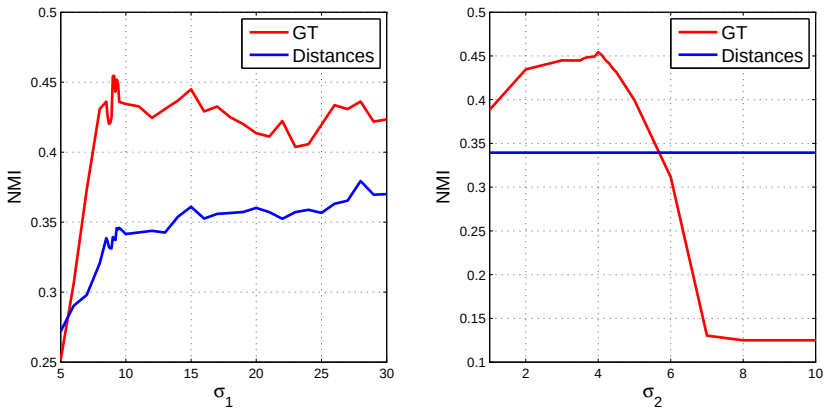


Figure 7.33: NMI using $\sigma_1 = 9$ and changing σ_2.

Gauss kernel

Gauss-DS clustering with GT obtains $NMI \approx 0.35$ using $\sigma_1 = 25.2$ and $\sigma_2 = 4.2$. The maximum NMI using distances is ≈ 0.34.

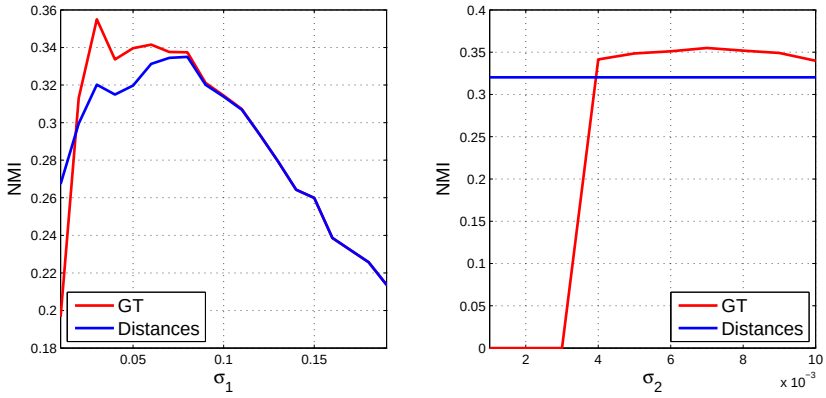


Figure 7.34: (a)NMI using $\sigma_2 = 0.007$ and changing σ_1; (b)NMI using $\sigma_1 = 0.03$ and changing σ_2.

Final considerations

In the following table we give a summary of the results of the performed tests:

Dataset	Event	Scene13	Caltech101
GT-EulerDS	**0.43**	**0.58**	**0.46**
EulerDS	0.34	0.38	0.37
GT-GaussDS	0.22	0.31	0.35
GaussDS	0.22	0.28	0.34

Table 6.3: NMI using EG kernel and Gauss kernel with and without GT.

As we can see, combining Euler-Gauss kernel with graph transduction improves in a very significant way the clustering of Dominant-Set algorithm. Not only the Euler-Gauss Kernel alone always overcomes the performances of the Gauss kernel, even with GT, but the contribution of the graph transduction is higher when Euler-Gauss Kernel is used: the average difference between NMI using distances or GT indeed is ≈ 0.01 with Gauss kernel (in Event dataset there are no improvements) while with EG kernel is ≈ 0.13 (in Scene13 dataset the improvement is 0.20). We schemes the improvements in table 6.4:

Dataset	Event	Scene13	Caltech101
Base	0.22	0.28	0.34
Using EG kernel	+0.12	+0.10	+0.03
Using GT	+0.0	+0.03	+0.01
Using both	+0.21	+0.30	+0.12

Table 6.4: improvement of NMI using EG kernel and graph transduction.

The increase of NMI due to the simultaneous use of the two methods is higher than the sum of the increases using the Euler-Gauss Kernel and the graph transduction separately. This fact means that, at least in such kind of datasets, the combination of the two methods creates a sort of synergy that permits better performances.

Comparison with other Clustering Methods

In this final section we compare the performances of Euler-Gauss Dominant set combined with graph transduction with other clustering techniques evaluated in [34]:

- classic K-means;
- kernel K-means with Gaussian Kernel (G-K-Kmeans)
- kernel K-means with Spatial Pyramid Kernel (P-K-Kmeans);
- Normalized Cut Algorithm with Gaussian Kernel (G-N-Cut);
- Normalized Cut Algorithm with Spatial Pyramid Kernel (P-N-Cut).
- Euler K-means with $\alpha = 700$ (as Euler-Gauss DS).

The Spatial Pyramid Kernel (see [17]) is a kernel specifically designed for image datasets created in the way explained in section 7.3.
The average results using for each method its optimal parameters are resumed in the following table:

Dataset	Event	Scene13	Caltech101
K-means	0.13	0.26	0.32
G-K-K-means	0.23	0.31	0.33
P-K-K-means	0.36	0.51	0.45
G-N-Cut	0.14	0.27	0.35
P-N-Cut	0.31	0.39	0.45
Euler K-means	0.38	**0.58**	**0.49**
GT-EulerDS	**0.43**	**0.58**	0.46

Table 6.5: NMI evaluated using different clustering algorithms.

The algorithms that use Euler Kernels have always better performances than the others. In Event dataset the Euler-Gauss Dominant-Set clustering overcomes the Euler K-means with a difference between NMI of 0.05. Scene13 is the dataset in which the improvement by using Euler Kernels is higher: in this case Euler K-means and Euler DS with GT achieves the same NMI. In Caltech101 dataset is the Euler K-means the method that performs slighty better, with a difference between NMI of 0.03.

Chapter 8

Conclusions

In this last chapter we draw some conclusions about the work done. To sum up we have implemented a Dominant-Set clustering algorithm that uses Euler-Gaussian Kernels to compute the pairwise similarities between points given a dataset with objects represented as real valued vectors. Moreover, in order to create a partition of the data, we have used the output of the DS-algorithm as a partially labeled training set for the game-theoretic graph transduction algorithm that uses the same Euler-Gaussian Kernels to propagate the label information among the objects. We have tested the two improvements individually and we have compared their performances with the classic counterparts:

- The use of the Euler-Gaussian Kernel instead of a traditional Gaussian Kernel to create the payoff matrix for the Dominant-Set clustering improves, at least in the datasets tested, the extraction of purer clusters from the data.
- The use of the graph transduction instead of distances to label the unclustered objects permits a for the majority of the cases a better partitional clustering of the data.

The combined use of Euler-Gaussian Kernel and graph transduction achieves better performances than the sum of the improvements achieved using Dominant-Set algorithm with one improvement at once. Both the methods increase in a more significant way the performances of the DS-clustering when the real clusters have irregular shapes.

The performances of the DS-clustering on image datasets using Euler-Gaussian Kernels and graph tranduction overcome those of native partitional clustering algorithms, moreover the use of Euler-Gaussian Kernel achieves better results than using kernels specifically designed for such type of data.

Appendix A

Mathematical Notation

Lowercase letters are mostly used for real/complex numbers and functions (e.g.: a, b, c).
Lowercase bold letters are used for real/complex vectors (e.g.: $\mathbf{a}, \mathbf{b}, \mathbf{c}$). The value in position i of vector $\mathbf{v}$ is denoted by v_i or $\mathbf{v}(i)$.
Uppercase bold letters are used for real/complex matrices (e.g.: $\mathbf{A}, \mathbf{B}, \mathbf{C}$). In case of bidimensional matrix the value in position i, j of matrix $\mathbf{M}$ is denoted by M_{ij}, m_{ij} or $\mathbf{M}(i, j)$.
The trasposition of a matrix or vector is denoted by a T (e.g.: $\mathbf{A}^T$, $\mathbf{a}^T$), the complex conjugate is denoted by the dagger symbol (e.g.: $\mathbf{A}^\dagger$, $\mathbf{a}^\dagger$).

A.1 special vectors

With the symbol $\mathbf{e}^k$ we denote an unit vector in which all the components are equal to zero except the one in position k, equal to 1. The symbol $\mathbf{1}$ represents a vector with all the components equal to 1.

Bibliography

[1] M. Ankerst, M. M. Breunig, H. peter Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. pages 49–60. ACM Press, 1999.

[2] A. S. B. Scholkopf and K.-R. Muller. Kernel methods in machine learning. *The Annals of Statistics, 36(3):1171-1220*, 2008.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. 2006.

[5] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 551–556, New York, NY, USA, 2004. ACM.

[6] A. Erdem and M. Pelillo. Graph transduction as a noncooperative game. *Neural Comput.*, 24(3):700–723, Mar. 2012.

[7] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[8] B. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. A Hodder Arnold Publication. Wiley, 2001.

[9] R. Fisher. *The Genetical Theory of Natural Selection*. Clarendon Press, Oxford, 1930.

[10] A. J. Fitch, A. Kadyrov, W. J. Christmas, and J. Kittler. Fast robust correlation. *Trans. Img. Proc.*, 14(8):1063–1073, Aug. 2005.

[11] R. M. Haralick, L. S. Davis, A. Rosenfeld, and D. L. Milgram. Reduction operations for constraint satisfaction. *Inf. Sci.*, 14(3):199–219, 1978.

[12] J. Hofbauer, P. Schuster, and K. Sigmund. A note on evolutionary stable strategies and game dynamics. Levine's Working Paper Archive 441, David K. Levine, Dec. 2010.

[13] J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, May 1998.

[14] T. Hofmann and J. M. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(1):1–14, Jan. 1997.

[15] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(3):267–287, Mar. 1983.

[16] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[17] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2169–2178, Washington, DC, USA, 2006. IEEE Computer Society.

[18] F.-F. Li and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 524–531, Washington, DC, USA, 2005. IEEE Computer Society.

[19] L.-J. Li and L. Fei-Fei. What, where and who? classifying event by scene and object recognition. In *Proc. of IEEE Intern. Conf. in Computer Vision (ICCV).*, 2007.

[20] S. Liwicki, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. Euler principal component analysis. *Int. J. Comput. Vision*, 101(3):498–518, Feb. 2013.

[21] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[22] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[23] J. Maynard Smith and G. R. Price. The logic of animal conflict. *Nature*, 246(5427):15–18, 1973.

[24] N.Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society, 68:337-404*, 1950.

[25] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[26] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(1):167–172, Jan. 2007.

[27] M. Pelillo. The dynamics of nonlinear relaxation labeling processes. *J. Math. Imaging Vis.*, 7(4):309–323, Oct. 1997.

[28] B. Schölkopf. The kernel trick for distances. *Advances in Neural Information Processing Systems, 301-307*, 2000.

[29] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319, July 1998.

[30] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[31] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[32] P. Taylor and L. Jonker. Evolutionary stable strategies and game dynamics. *Mathematical Biosciences*, 40:145–156, 1978.

[33] J. Weibull. *Evolutionary Game Theory*. MIT Press, 1997.

[34] J.-S. Wu, W.-S. Zheng, and J.-H. Lai. Euler clustering. In F. Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[35] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328. MIT Press, 2004.

[36] X. Zhu. Semi-supervised learning literature survey, 2006.