On the Robustness of Clustering Algorithms to Adversarial Attacks

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Year 2018-2019

Graduand Antonio Emanuele Cinà
Supervisor Marcello Pelillo

Assistant supervisor Alessandro Torcinovich

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Marcello Pelillo, for having spent much time helping me during the development of my thesis, recommending papers which have been both useful and inspirational for expanding this work. His lessons have made him a great advisor and have taught me priceless notions on how to do research. Together with Alessandro Torcinovich, one of his Ph.D students, I feel we formed an excellent group in which we developed multiple curious and useful ideas both related to this work and also to future projects, which I will address during my Ph.D career.
I would also like to thank my friends for all the contributions that supported me while I was working on this challenging project.
Last but not least, I want to thank my parents, who have allowed me to pursue the best education I could ever have gotten and have given me the possibility to stay here in Venice during my academic career.
Parents and friends comforted me with much patience, encouragement and also kind smiles which I will never forget.

Abstract

Machine learning is becoming more and more used by businesses and private users as an additional tool for aiding in decision making and automation processes. However, over the past few years, there has been an increased interest in research related to the security or robustness of learning models in presence of adversarial examples. It has been discovered that wisely crafted adversarial perturbations, unaffecting human judgment, can significantly affect the performance of the learning models. Adversarial machine learning studies how learning algorithms can be fooled by crafted adversarial examples. In many ways it is a recent research area, mainly focused on the analysis of supervised models, and only few works have been done in unsupervised settings. The adversarial analysis of this learning paradigm has become imperative as in recent years unsupervised learning has been increasingly adopted in multiple security and data analysis applications. In this thesis, we are going to show how an attacker can craft poisoning perturbations on the input data for reaching target goals. In particular, we are going to analyze the robustness of two fundamental applications of unsupervised learning, feature-based data clustering and image segmentation. We are going to show how an attacker can craft poisoning perturbations against the two applications. We choose 3 very well known clustering algorithms (K-Means, Spectral and Dominant Sets clustering) and multiple datasets for analyzing the robustness provided by them against adversarial examples, crafted with our designed algorithms.

Keywords Adversarial Machine Learning, Unsupervised Learning, Clustering Algorithms, Machine Learning, Security, Robustness.

Contents

Chapter 1

Introduction

Nowadays we have enough knowledge about the ability of machine learning models to make good predictions in different domains, like image classification, speech recognition, market analysis and image segmentation. Due to their incredible results, learning systems are assuming a fundamental role in very sophisticated applications as tools for aiding in decision making. However, it has been observed that, despite their advanced capabilities, they are sensitive to adversarial perturbations in the input data, leading algorithms to make wrong predictions. A very significant observation is that sometimes these alterations are invisible to human eyes, leaving us some doubts about how these models manage data. The key problem is that these models have not been designed for working in scenarios where an attacker wants to subvert or compromise the results of the system. Essentially an attacker can carefully craft adversarial samples to inject inside data for subverting the normal behavior of the machine learning model. In [8, 22] the authors discussed the problems of adversarial perturbations for spam filtering systems. They show how linear classifiers can be easily fooled by simply injecting carefully crafted perturbations inside spam messages. Even other applications of machine learning models (like fraud detection, face recognition, video surveillance, traffic prediction, credit-risk assessment, etc.) could be subject to malicious activities. Due to the sensitivity of these domains, defense strategies and robustness analyses have been studied for limiting the vulnerability to adversarial examples [1, 2, 7, 10, 18]. Nevertheless, security can be seen as an arms race game between attacker and designer. The former wants to subvert the system, on the other hand, the latter makes the system available and wants to protect it developing opportune countermeasures. During the game each player evolves its strategy in contrast to the other. Indeed, the designer develops defensive countermeasures against threats and the attacker develops new attacks for breaking defensive strategies. The key problem of machine learning models is that they are completely data-driven and data contains noise by nature. Consider the scenario of a network of temperature sensors in a room and, according to the retrieved temperature, the system reacts with appropriate strategies. Data collected by sensors are commonly subject to noise, due to internal and/or external variables. However, sensors can also be manipulated by an attacker for obtaining certain results. From this observation, we can see how it is difficult to detect when data are subject to natural noise or when they are affected by malicious threats, named adversarial noise.
In the rest of this Chapter we are going to introduce the theory behind the adversarial machine learning field, limits of the current state of the art and how this thesis is organized.

1.1 Adversarial Machine Learning Theory

The sensitivity of machine learning models to adversarial noise has recently attracted a large interest in the computer science community. Different authors, such us Biggio and Roli, proposed a theoretical framework for studying the roles of attacker and designer in this arms race game. In their comprehensive review [2] they propose a framework for studying the security property of supervised models. In next works, Biggio et al. [4] extended the initial framework for the unsupervised paradigm. In particular, they provided a taxonomy about possible adversary's goals and knowledge.

1.1.1 Adversary's Goal

The adversary's goal defines the objective function that the attacker wants to maximize or minimize by injecting adversarial examples to the system. Essentially the attacker may be interested on affecting three security properties: integrity, availability, confidentiality.

Integrity Violation The attacker performs malicious activities that do not compromise significantly the system behavior. In the clustering scenario we refer to the violation of the resulting grouping. Indeed, with integrity violation the attackers aims to subvert the grouping for a portion of samples preserving the original grouping as much as possible for the rest of the sample. For example, given a dataset X containing two groups of users: `authorized` or `guest`. The attacker may inject well crafted adversarial perturbations for moving samples from `guest` towards `authorized`.

Availability Violation The attacker performs malicious activities for limiting functionalities or access to the system. In the clustering scenario the attackers wants to subvert completely the clustering process. Considering the `authorized` and `guest` groups, the attacker injects noise such that at the end the two clusters are reversed or they are merged together.

Confidentiality Violation The attacker performs malicious activities in order to extract private information about the composition of the clusters. Considering the `authorized` and `guest` clusters, the attacker may extract fundamental features from `authorized` users by applying strategies of reverse-engineer of the clustering process.

For all the 3 cases the attacker may also define a certain specificity, that could be targeted or indiscriminate. In the first case a targeted subset of samples is subject to adversarial perturbations. On the other hand, any sample can be manipulated without any distinction with respect to the others. For example, consider a dataset composed by three clusters: `authorized`, `guest` and `rejected`. If the attacker moves explicitly samples from rejected towards authorized, then we define the attacker's goal as targeted. Conversely, if the attacker moves samples without a target direction, then we define the attacker's goal as indiscriminate. In the proposed example, this would mean that moving samples from `guest` towards `authorized` gives the same payoff as moving samples from `guest` towards `rejected`.

1.1.2 Adversary's Knowledge

The adversary's knowledge defines the capacity of the attacker to inject malicious threats. The more knowledge the attacker has about the system, the greater the attacker's capacity is. It is reasonable to think that if the attacker knows perfectly the system, then it is easier is for him/her to craft appropriate threats. Conversely, if the attacker has no knowledge about the system, it is more difficult to craft threats against a black box system.
Biggio et al. provided in [4] a taxonomy of attacker's knowledge:

- Knowledge of data $\mathcal{D}$: the attacker knows exactly the composition of the data samples taken into consideration. It could be unnecessary the complete knowledge but a portion of the entire collection might suffice. $\mathcal{D}$ is drawn from an unknown probability distribution p.

- Knowledge of the feature space: the attacker knows exactly the features composition of data samples or how they are obtained.

- Knowledge of the algorithm: the attacker knows which algorithm is used for clustering data samples and how similarity between them is computed if necessary.

- Knowledge of the algorithm's parameters: the attacker knows the parameters provided to the clustering algorithm (ex: the number of clusters k).

The best scenario for the attacker, and consequently the worst case for the designer, is the one in which he/she has full knowledge (Perfect Knowledge) of the target system. Certainly, this is not always the case, bringing the attacker to a Limited knowledge, or worst, in a Zero knowledge. Even if the attacker has zero knowledge some strategies can be adopted for estimating the required components. For example, in absence of knowledge about $\mathcal{D}$ the attacker can collect a surrogate dataset $\hat{\mathcal{D}}$ with the hope that samples are retrieved from the same distribution p. For instance, if the target system uses clustering of malware samples, then the attacker can collect a surrogate dataset of malware $\hat{\mathcal{D}}$ and use them as an estimate of the true dataset $\mathcal{D}$. The attacker may also have no knowledge about the features representation used for projecting samples in the space. In certain conditions it could be easier to identify the right feature representation since machine learning is getting more and more standardized to provide unified services. For example, documents are likely represented in vectors of TF-IDF, or images are likely defined in matrices of intensities or RGB values.
A common argument in cyber-security is that in order to build a secure system, it should overestimate the attacker's capabilities rather than underestimating them.

1.2 Problem Description

Because of its strong implications, adversarial machine learning theory has been developed more and more in the latest years' research works. The greatest part of the research, including [3, 7, 14, 23], addresses the problem of adversarial examples against supervised learning. In these works authors try to give explanations to the adversarial phenomenon going deeper on how models work. Especially, Yin et al. give a first connection between adversarial learning and statistical learning theory,

with the usage of the Rademacher complexity. Papernot et al. introduce a key property of adversarial examples, which is their transferability between multiple models. Even defensive and adversary algorithms have been designed for crafting adversarial examples or for preventing the system against threats.
Conversely, from the best of our knowledge, only few works have been published against unsupervised learning paradigm applications. Biggio et al. develop some strategies for fooling single-linkage hierarchical clustering, and later on, a similar work [5] has been developed against complete-linkage hierarchical clustering. Despite this lack, the demanding techniques for cracking clustering models or for protecting them have recently turned to be fundamental. Clustering is finding a wide range of applications, due to the absence of labeled data, in very sensitive domains like image segmentation, face clustering, market or social analysis, information processing, etc.
The aim of this research is to investigate the robustness provided by some clustering algorithms in presence of very well-crafted adversarial examples. We provide three ways for crafting adversarial threats against clustering algorithms and we analyze how K-Means, Spectral and Dominant Sets clustering react against them.

1.3 Outline

For the remainder of this thesis, we organize our work as follows. In Chapter.2, we briefly introduce the background related to the unsupervised learning paradigm. More specifically we give a deeper introduction to the three clustering algorithm analyzed in the rest of this thesis (K-Means, Spectral and Dominant Sets clustering). We give knowledge about how these algorithms work, their formulation and properties. Following it in Chapter.3, we introduce the 3 designed algorithms against clustering. In the first part we introduce the concept and applications of image segmentation and how sensitive could be that field in presence of adversarial noise. We discuss the two designed algorithms that can be used by an attacker for fooling image classification. Conversely, in the second part of Chapter.3, we introduce the sensitivity of certain clustering applications in possible adversarial settings. We provide and explain how the three designed adversarial algorithms work in order to fool data clustering algorithms.
In Chapter.4 we show the experiments done during the development of this thesis and we analyze the robustness provided by the three algorithms. We show, using different visualization techniques, how the three algorithms react by changing the attacker's capacity.
Finally, in Chapter.5, we conclude this thesis with discussions, open issues and we highlight better our contribution thanks to this thesis. At last, we propose a list of open issues, possible future improvements and ideas, realized during the development of this work, in order to give future research directions on adversarial machine learning.

Chapter 2

Unsupervised Learning

Unsupervised learning is a paradigm of the machine learning field which is based on the training of knowledge without using a teacher. It includes a large set of techniques and algorithms used for learning from data without knowing the true classes. The main application of unsupervised learning consists on estimating how data are organized in the space, such that they can reconstruct the prior probability distribution of data. For doing that clustering algorithms are used with the goal of grouping a set of objects in such a way that objects in the same cluster are strongly similar (*internal criterion*) and objects from distinct clusters are strongly dissimilar (*external criterion*).
The classical clustering problem starts with a set of n objects and an $n \times n$ affinity matrix A of pairwise similarities that gives us an edge-weighted graph G. The goal of the clustering problem is to partition vertices of G into maximally homogeneous groups (clusters). Usually the graph G is an undirected graph, meaning that the affinity matrix A is symmetric.

Figure 2.1: "Classical" clustering problem.

In literature we can find different clustering algorithms that are strongly used, and each of them manages data in different ways. Some of the clustering algorithm, that we are going to test against adversarial noise in chapter 4, are: K-Means , Spectral and Dominant Sets clustering.

2.1 Images as Graphs

In some applications we can have that images correspond to our data for which we want to obtain groups partition. In this case the clustering algorithms could be used in order to reconstruct a simplified version of the input image, removing noisy information. For doing that the image is represented as an edge-weighted

undirected graph, where vertices correspond to individuals pixels and edge-weights reflect the similarity between pairs of vertices. Given an input image with $H \times W$ pixels we construct a similarity matrix A such that the similarity between the pixels i and j is measured by:

$$A(i,j) = \exp\left(\frac{-||F(i) - F(j)||_2^2}{\sigma^2}\right)$$

- $F(i)$, is the normalized intensity of pixel i (*intensity segmentation*).
- $F(i) = [v, vs\sin(h), vs\cos(h)](i)$ where h, s, v are the HSV values of pixel i (*color segmentation*).
- $F(i) = [|I * f_1|, \ldots, |I * f_k|](i)$ is a vector based on texture information at pixel i (*texture segmentation*).

The constant σ is introduced for obtaining a scaling effect on the affinity:

- Small σ: only nearby points are similar.
- Large σ: distant points tend to be similar.

An example of application of clustering algorithms for image segmentation is below provided:

Figure 2.2: Image of vegetables.

Figure 2.3: Clustering on pixels intensity.

Figure 2.4: Clustering on pixels color.

2.2 K-Means

K-Means is one of the simplest, famous and used iterative clustering algorithms. It aims to partition n objects into K maximal cohesive groups. The goal of the K-Means algorithm is to reach the following state: each observation belongs to the cluster with the nearest center. Its implementation can be shortly described in few lines:

- **Initialization:** Pick K random points as cluster centers (centroids).

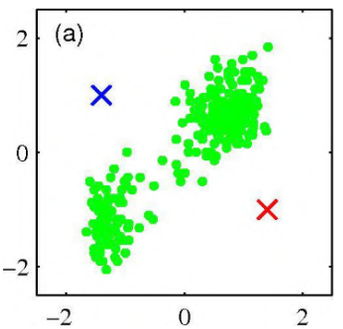


Figure 2.5: Initialization with $K = 2$.

- **Alternate:**
  1. Assign data points to closest cluster centroid.
  2. For each cluster C update the corresponding centroid to the average of points in C.

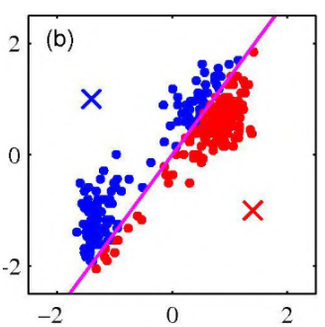


Figure 2.6: Iterative step 1.

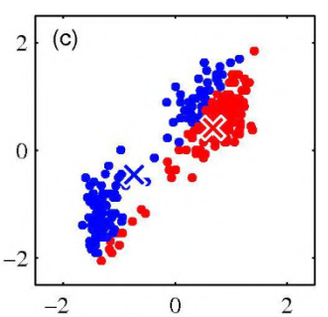


Figure 2.7: Iterative step 2.

- **Stop:** When no points' assignments change.

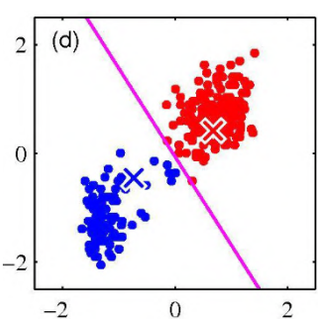


Figure 2.8: Repeat until convergence.

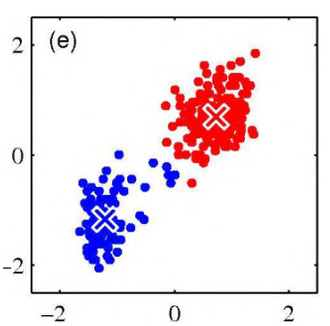


Figure 2.9: Final output.

Formally speaking we can define the K-Means algorithm over the set of points X in the following way:

1. Initialize cluster centroids $\mu_1, \ldots, \mu_k$,.
2. Repeat until all points remain unchanged (convergence):

   2.1. $\forall i \in X \quad c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2$

   2.2. $\forall j \in C \quad \mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}}$

Properties of K-Means. K-Means benefits of the following properties:

- It is guaranteed to converge in a finite number of steps.
- It minimizes an objective function, which represents the compactness of the retrieved K clusters:

$$\arg\min_{C} \sum_{i=0}^{K} \left\{ \sum_{j \in \text{elements of } C_i \text{ cluster}} ||x_j - \mu_i||^2 \right\}$$

  where μ_i is the centroid of cluster i.

- It is a polynomial algorithm: $O(Kn)$ for assigning each sample to the closest cluster and $O(n)$ for the update of the clusters center.

It is possible to say that K-Means is a very simple and efficient method but, on the other hand, it is strongly sensible to the initialization phase. If the initial centroids are not chosen correctly the algorithm converges to a local minimum of the error function. Another disadvantage of K-Means is that does not work well in presence non-convex shapes.

2.3 Eigenvector-based Clustering

The eigenvector-based clustering collects different techniques that use properties of eigenvalues and eigenvectors for solving the clustering problem. Let us represent a cluster using a vector x whose k-th entry captures the participation of node k in that cluster. If a node does not participate in cluster x, the corresponding entry is zero. We also impose the restriction that $x^T x = 1$. The goal of the clustering algorithm is to maximize:

$$\arg\max_{x} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j = x^T A x \tag{2.1}$$

which measures the cluster's cohesiveness and x_i, x_j represent the measure of centrality to the cluster and it is defined as:

$$x_i = \begin{cases} \neq 0 \text{ if } i \in C \\ = 0 \text{ if } i \notin C \end{cases}$$

.

Coming back to the notion of eigenvalues of a matrix we can say that λ is an eigenvalue of A and x_λ is the corresponding eigenvector if:

$$A x_\lambda = \lambda x_\lambda$$

From which we can derive that:

$$x_\lambda^T A x_\lambda = \lambda x_\lambda^T x_\lambda = \lambda$$

There are two important theorems that defines the nature of eigenvalues of a $n \times n$ matrix A:

1. If $A = A^T$ then A is symmetric and has only "*real*" eigenvalues. It means that we can sort them, from the smallest one to the largest one.

2. If A is symmetric then $\max_x x^T A x$ corresponds to the largest eigenvalue λ. Moreover, the corresponding eigenvector x_λ is the argument which maximizes the cohesiveness.

Taking advantage of the two theorems we can say that considering A as the affinity matrix, then clustering problem 2.1 corresponds to an **eigenvalue problem**, maximized by the eigenvector of A with the largest eigenvalue.

Clustering by Eigenvectors Algorithm

We can define the algorithm for extracting clusters from data points using the eigenvectors strategy with the following steps:

1. Construct the affinity matrix A from input G.
2. Compute the eigenvalues and eigenvectors of A.
3. Repeat
4. Take the largest unprocessed eigenvalue and the corresponding eigenvector.
5. Zero all the components corresponding to samples that have already been clustered.
6. Threshold the remaining components to detect which elements belong to this cluster.
7. If all elements have been accounted for, there are sufficient clusters.
8. Until there are sufficient clusters.

2.3.1 Clustering as Graph Partitioning

Let $G = (V, E, w)$ is an undirected weighted graph with $|V|$ nodes (samples) and $|E|$ edges. Note that it is undirected when the affinity matrix is symmetric. Given two graph partitions of vertices A and B, with $B = V \setminus A$, we define $\text{cut}(A, B)$ in the following way:

$$cut(A, B) = \sum_{i \in A} \sum_{j \in B} w(i, j)$$

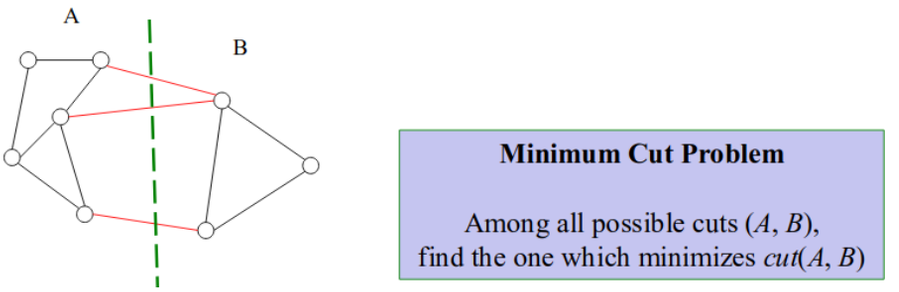


Figure 2.10: Minimum cut problem.

In the MinCut problem, we look for the partitioning that minimizes the cost of crossing from one A to B, which is the sum of weights of the edges which cross the cut. The fundamental idea is to consider the clustering problem as a graph partitioning. Indeed, the MinCut problem can be considered a good way of solving the clustering problem in graph data. The MinCut clustering is advantageous because it is solvable in polynomial time but, on the other hand, it favors highly unbalanced clusters (often with isolated vertices), indeed, it only measures what happens between the clusters and not what happens within the clusters.

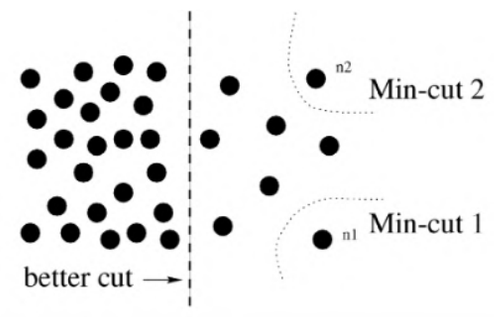


Figure 2.11: Minimum cut unbalance clusters.[19]

2.3.2 Normalized Cut

In order to overcome the problem of unbalanced clusters, a normalized version of the min cut problem, called **Normalized Cut**, is used and it is defined by:

$$Ncut(A,B) = \underbrace{cut(A,B)}_{\text{Between A and B}} \left(\underbrace{\frac{1}{vol(A)} + \frac{1}{vol(B)}}_{\text{Within A and B}} \right)$$

where $vol(A)$ is the volume of the set A given by $vol(A) = \sum_{i \in A} d_i,\ A \subseteq V$ and $d_i = \sum_j w_{i,j}$ is the degree of nodes (sum of weights).
The Normalized Cut has the advantage of taking into consideration what happens within clusters, indeed, considering $vol(A)$ and $vol(B)$ it takes into account what's going on within A and B.

2.3.3 Graph Laplacians

From an accurate analysis von Luxburg discovered that the main tools for spectral clustering are graph Laplacian matrices, defined in the spectral graph theory. In this section we are going to define different graph Laplacians and point out their most important properties since they will be later on used for solving the MinCut and NMinCut problem.

The Unnormalized Graph Laplacian. The **unnormalized graph Laplacian** matrix is defined as:

$$L = D - W$$

where:

- D is a diagonal matrix containing information about the degree of each node in G.
- W is the affinity matrix of G, containing $1s$ or $0s$ if nodes are adjacent. Diagonal elements are all set to 0.

In the following we provide an example of matrices D and W obtained considering the graph shown in Fig. 2.14.

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Figure 2.12: Degree matrix D.

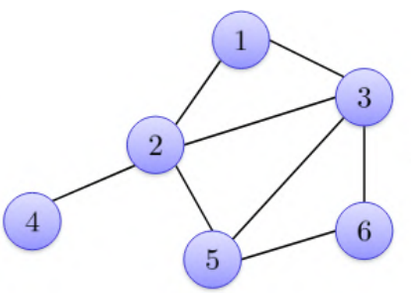
$$W = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2.13: Affinity matrix W.

The elements of L are given by:

$$L_{i,j} = \begin{cases} \mathrm{d}\,(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

where $d(v_i)$ is the degree of the vertex i.



$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 \\ -1 & -1 & 4 & 0 & -1 & -1 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}$$

Assume the weights of edges are 1

Figure 2.14: Laplacian matrix L associated to the graph in the left.

In [21] are reported the properties satisfied by the matrix L, that are:

1. For all vectors f in $\mathbb{R}^n$, we have:

$$f^\top L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(f_i - f_j\right)^2$$

This is proved by the definition of d_i:

$$\begin{aligned} f^\top L f = f^\top D f - f^\top W f &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij} \\ &= \frac{1}{2} \left(\sum_{i=1}^{n} \left(\sum_{j=1}^{n} w_{ij} \right) f_i^2 - 2 \sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} \left(\sum_{i=1}^{n} w_{ij} \right) f_j^2 \right) \\ &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(f_i - f_j\right)^2 \end{aligned}$$

2. L is symmetric (by assumption) and positive semi-definite. The symmetry of L follows directly from the symmetry of W and D. The positive semi-definiteness is a direct consequence of the first property, which shows that $f^T L f \geq 0$

3. The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant 1 vector.

4. L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$

First relation between spectrum and clusters:

- The multiplicity of eigenvalue $\lambda_1 = 0$ corresponds to the number of connected components of the graph.
- The eigenspace is spanned by the characteristic function of these components (so all eigenvectors are piecewise constant).

Normalized Graph Laplacians. In literature it is also defined the normalized form of a Laplacian graph. In particular, there exists two definitions that are closely related:

$$\begin{aligned} L_{\text{sym}} &= D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \\ L_{\text{rw}} &= D^{-1} L = I - D^{-1} W \end{aligned}$$

The first matrix L_{sym} is a symmetric matrix, and the second one L_{rw} as a normalized form of a Laplacian graph which is closely connected to a random walk [21].

Definition 2.3.1 (Properties for Laplacian matrices and normalized ones)**.** In relation to Laplacian matrices, it is possible to notice that, let L be the Laplacian of a graph $G = (V, E)$. Then, $L \geq 0$, indeed:
$\forall x = (x_1, \ldots, x_n)$,

$$\begin{aligned} x^T L x &= x^T \sum_{e \in E} L_e x \\ &= \sum_{e \in E} x^T L_e x \\ &= \sum_{i,j \in E} \left(x_i - x_j\right)^2 \geq 0 \end{aligned}$$

In relation instead to the normalized Laplacian Matrix we have that:

$$\forall x \in \mathbb{R}^n \quad x^T L x = \sum_{i,j} \left(\frac{x(i)}{\sqrt{d(i)}} - \frac{x(j)}{\sqrt{d(j)}} \right)^2 \geq 0$$

2.3.4 Solving Ncut

Any cut (A, B) can be represented by a binary indicator vector x:

$$x_i = \begin{cases} +1 \text{ if } i \in A \\ -1 \text{ if } i \in B \end{cases}$$

It can be shown that:

$$min_x \; Ncut(x) = min_y \underbrace{\frac{y'(D-W)y}{y'Dy}}_{\text{Rayleigh quotient}} \tag{2.2}$$

subject to the constraint that $y'D1 = \sum_i y_i d_i = 0$ (with $y_i \in \{1, -b\}$ (relaxation introducing also real values), indeed y is an indicator vector with 1 in the i-th position if the i-th feature point belongs to A, negative constant $(-b)$ otherwise).

Theorem 2.3.2 (Solving Ncut proof)**.**

$$\lambda_2 = \min_x \frac{x^T L x}{x^T x} = \min_x \frac{x^T D^{-1/2} L D^{-1/2} x}{x^T x} \qquad \textit{Remember } L_{sym} = D^{-1/2} L D^{-1/2}$$

Considering the change of variables obtained by setting $y = D^{-1/2}x$ *and* $x = D^{1/2}y$*:*

$$\lambda_2 = \min_y \frac{y^T L y}{(D^{1/2}y)^T (D^{1/2}y)} = \min_y \frac{y^T L y}{y^T D y}$$

Issues rise up because solving Problem 2.2 is not computationally efficient since it is an **NP-Hard** problem. The huge Ncut time complexity brings us to take into consideration an approximation of it. If we relax the constraint that y must be a discrete-valued vector and allow it to take on real values, then the original problem

$$\min_y \frac{y'(D-W)y}{y'Dy}$$

is equivalent to:

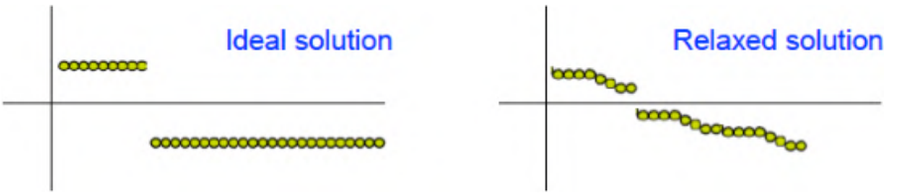
$$\min_y y'(D-W)y \quad \text{ subject to } y'Dy = 1$$

This amount to solve a *generalized* eigenvalue problem, but now the optimal solution is provided by the second smallest eigenvalue since we want to minimize the cut. Note that we pick the second smaller eigenvalues since we have seen the smallest one is always zero and corresponds to the trivial partitioning $A = V$ and $B = \emptyset$.

$$\underbrace{(D-W)}_{Laplacian} y = \lambda D y$$

We started from an NP-Hard problem and through relaxation we reached a feasible solution. However, we have not the warranty that the relaxed solution is in one to one correspondence.

The effect of relaxation. Through the relaxation we loose some precision in the final solution.



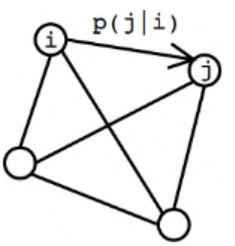
Note that the original problem returns binary values $(-1, 1)$, indicating the clustering membership. The relaxed version, on the right, returns continuous values of it can be the case that some points are not so clear to assign (close to the margin between the two). For that reason relaxed solution not always is in one-to-one correspondence with the original problem.

2.3.5 Random Walk Interpretation

The Ncut problem can be formalized also in terms of random walk, as highlighted in [21], since we want to find a cut that reduces the probability of jumping between nodes of different clusters. It can be defined by a Markov chain where each data point is a state, connected to all other states with some probability. With our affinity W and degree D, the stochastic matrix is:

$$P = D^{-1}W$$

which is the row-normalized version of W, so each entry $P(i, j)$ is a probability of "*walking*" to state j from state i[9].



Probability of a walk through states $(s_1, \ldots, s_m)$ is given by:

$$P(s_1, \ldots, s_2) = P(s_1) \prod_{i=2}^{m} P(s_i, s_{i-1})$$

Suppose we divide the states into two groups, and we want to minimize the probability of jumping between the two groups. We can formulate this as an eigenvector problem:

$$Py = \lambda y$$

where the component of vector y will give the segmentation.
We can precise also that:

- P is a stochastic matrix.
- The largest eigenvalue is 1, and its eigenvector is the all-one vector 1. Not very informative about segmentation.

- The second largest eigenvector is orthogonal to the first, and its components indicate the strongly connected sets of states.
- Meila and Shi (2001) showed that minimizing the probability of jumping between two groups in the Markov chain is equivalent to minimizing Ncut.

Theorem 2.3.3 (Random Walk Proposition)**.** *(λ, y) is a solution to $Py = \lambda y$ if and only if* [1]*:*

- *$1 - \lambda$ is an eigenvalue of $(D - W)y = \lambda Dy$*
- *y is an eigenvector of $(D - W)y = \lambda Dy$*

Proof:

$$\begin{aligned} Py = \lambda y &\Leftrightarrow -Py = -\lambda y \\ &\Leftrightarrow y - Py = y - \lambda y \\ &\Leftrightarrow (I - P)y = (1 - \lambda)Iy \\ &\Leftrightarrow \left(D^{-1}D - D^{-1}W\right) y = (1 - \lambda)D^{-1}Dy \\ &\Leftrightarrow D^{-1}(D - W)y = D^{-1}(1 - \lambda)Dy \\ &\Leftrightarrow (D - W)y = (1 - \lambda)Dy \end{aligned}$$

The problem is to find a cut (A, B) in a graph G such that a random walk does not have many opportunities to jump between the two clusters.
This is equivalent to the Ncut problem due to the following relation:

$$Ncut(A, B) = P(A|B) + P(B|A)$$

2.3.6 2-way Ncut clustering algorithm

In section 2.3.4 we have seen how to solve the Normalized Cut clustering problem, and here we want to discuss its implementation for extracting just two clusters:

1. Compute the affinity matrix W, compute the degree matrix D. D is diagonal and $D_{i,i} = \sum_{j \in V} W_{i,j}$
2. Solve the generalized eigenvalue problem $(D - W)y = \lambda Dy$
3. Use the eigenvector associated to the second smallest eigenvalue to bipartition the graph into two parts.

Sometimes there's not a clear threshold to split based on the second vector since it takes continuous values. In which way it is possible to choose the splitting point?

- Pick a constant value (0 or 0.5).
- Pick the median value as the splitting point.
- Look for the splitting point that has minimum Ncut value:
  1. Choose n possible splitting points.
  2. Compute Ncut value.
  3. Pick the minimum.

[1] Adapted from Y. Weiss

2.3.7 K-way Ncut clustering algorithm

In the case we want to extract more than 2 clusters we can adopt two possible strategies:

Approach # 1. It is a recursive two-way cuts:

1. Given a weighted graph $G = (V, E, w)$, summarize the information into matrices W and D.
2. Solve $(D - W)y = \lambda Dy$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph by finding the splitting point such that Ncut is minimized.
4. Decide if the current partition should be subdivided by checking the stability of the cut, and make sure Ncut is below the prespecified value.
5. Recursively repartition the segmented parts if necessary.

Note that the approach is computationally wasteful, only the second eigenvector is used, whereas the next few small eigenvectors also contain useful partitioning information.

Approach #2. Using the first k eigenvectors:

1. Construct a similarity graph and compute the unnormalized graph Laplacian L.
2. Compute the k smallest **generalized** eigenvectors $u_1, u_2, \ldots, u_k$ of the generalized eigenproblem $Lu = \lambda Du$.
3. Let $U = [u_1, u_2, \ldots, u_k] \in \mathbb{R}^{n \times k}$.
4. Let $y_i \in \mathbb{R}^k$ be the vector corresponding to the ith row of U.

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix} = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{bmatrix}$$

5. Thinking of y_i's as points in $\mathbb{R}^k$, cluster them with k-means algorithms.

2.3.8 Spectral Clustering vs K-Means

First of all, let us define the spectral clustering algorithm [12], its goal is to cluster objects that are connected but not necessarily compact or clustered within convex boundaries. The algorithm has in input the similarity matrix $S \in \mathbb{R}^{n \times n}$ and the k number of clusters to construct. It returns in output the k clusters. It follows these steps:

1. Construct a similarity graph and compute the normalized graph Laplacian L_{sym}.
2. Embed data points in a low-dimensional space (spectral embedding), in which the clusters are more obvious, computing the k smallest eigenvectors $v_1, \ldots, v_k$ of L_{sym}.
3. Let $V = [v_1, \ldots, v_k] \in \mathbb{R}^{n \times k}$.

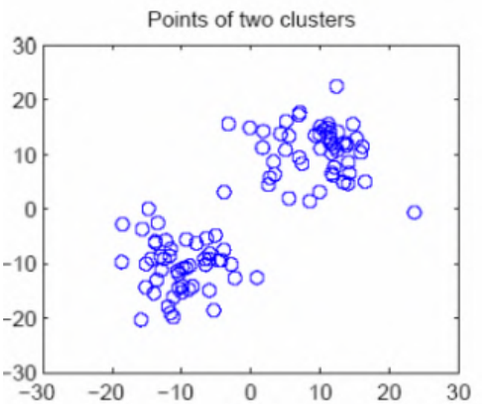
4. Form the matrix $U \in \mathbb{R}^{n \times k}$ from V by normalizing the row sums to have norm 1, that is:

$$u_{ij} = \frac{v_{ij}}{\left(\sum_k v_{ik}^2\right)^{1/2}}$$

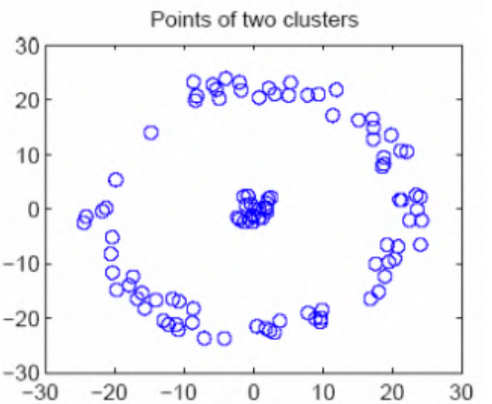
5. For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the ith row of U.

6. Cluster the points y_i with $i = 1, \ldots, n$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

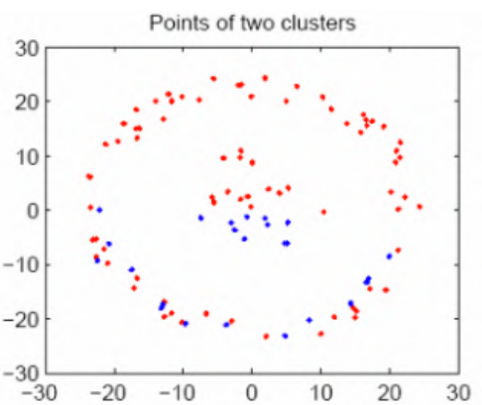
Applying k-means to Laplacian eigenvectors allows us to find cluster with non-convex boundaries.



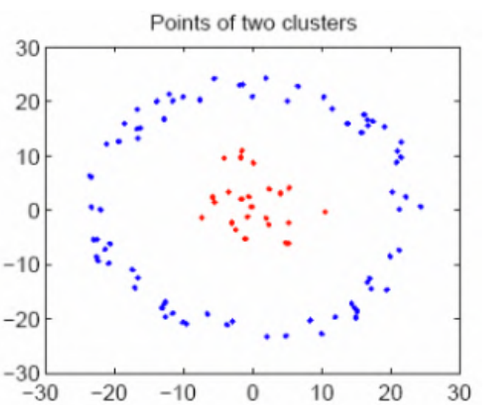
Both perform same



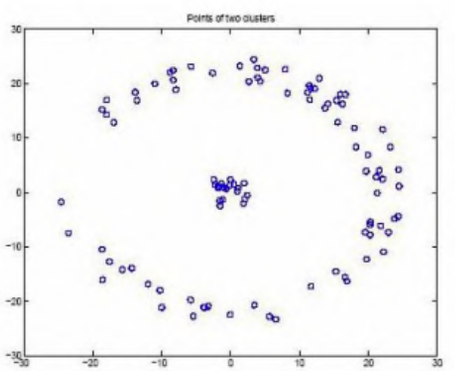
Spectral clustering is superior

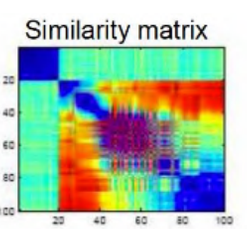


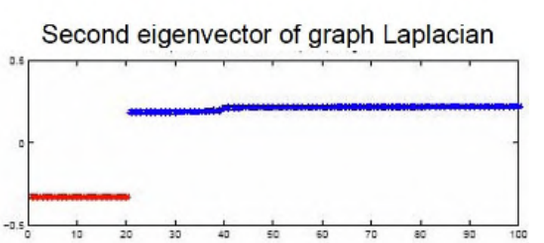
k-means output



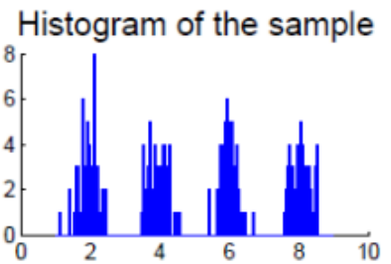
Spectral clustering output

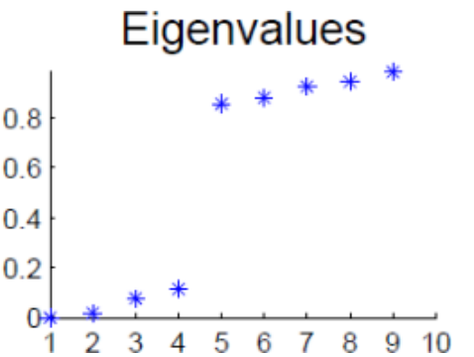






One of the possible problems that could appear on the usage of the Spectral Clustering algorithm consists on choosing the best k. We want to find a k such that all eigenvalues $\lambda_1, \ldots, \lambda_k$ are very small, but λ_{k+1} is relatively large. In this way, the choosing of k maximizes the eigengap (difference between consecutive eigenvalues) $\delta_k = |\lambda_k - \lambda_{k-1}|$.





2.4 Dominant Sets

In the previous sections we have seen that data can be represented using weighted graphs, also called similarity graphs, in which data are represented by nodes in the graph and the edges represent the similarity relation between nodes. This representation allows us to codify also very complex structured entities. In literature some authors argue to the fact that a cluster can be seen as a **maximal clique**[2] of a graph, indeed the concept of clique is related to the internal cluster criteria, instead maximal clique responds to the external criteria. But the standard definition of click does not consider weighted graphs. For this reason, dominant set is introduced by Pavan and Pelillo as an extension of the maximal clique problem. We are going to see that the notion of dominant set provides measures of cohesiveness of a cluster and vertex participation of different clusters.

2.4.1 Cluster in Graph Theory

Data to be clustered could be coded as an undirected weighted graph with no self-loops: $G = (V, E, \omega)$, where $V = \{1, \ldots, n\}$ is the vertex set, $E \subseteq V \times V$ is the edges set and $w : E \to \mathbb{R}^*_+$ is the positive weight function. Vertices represent data points, edges neighborhood relationships and edge-weights similarity relations. G is then represented with an adjacency matrix A, such that $a_{ij} = \omega(i, j)$. Since there are not self-loops we have that $\omega(i, i) = 0$ (main diagonal equal to 0).
One of the key problem of clustering is that there is not a unique and well define definition of cluster, but in literature researches agree that a cluster should satisfy two conditions:

- **High internal homogeneity**, also named *Internal criterion*. It means that all the objects inside a cluster should be highly similar(or low distance) to each other.
- **High external in-homogeneity**, also named *External criterion*. It means that objects coming from different clusters have low similarity (or high distance).

The idea of the criterion is that clusters are groups of objects which are strongly similar to each other if they become to the same cluster, otherwise they have a highly dissimilarity. Informally speaking a cluster is a set of entities which are alike, and entities from different clusters are not alike.
Let $S \subseteq V$ be a nonempty subset of vertices and $i \in S$. The average weighted degree of i with regard to S is defined as:

$$\text{awdeg}_S(i) = \frac{1}{|S|} \sum_{j \in S} a_{ij} \tag{2.3}$$

This quantity over here represents the average similarity between entity i and the rest of the entities in S. In other words how much similar i is in average with all the objects in S. It can be observed that $\text{awdeg}_{\{i\}}(i) = 0 \ \forall i \in V$, since we have no self-loops.
We now introduce a new quantity ϕ such that if $j \notin S$:

$$\phi_S(i, j) = a_{ij} - \text{awdeg}_S(i) \tag{2.4}$$

[2] A **clique** is a subset of mutually adjacent vertices.
A **maximal clique** is a clique that is not contained in a larger one.

Note that $\phi_{\{i\}}(i,j) = a_{ij}\ \forall i,j \in V$ with $i \neq j$. $\phi_S(i,j)$ measures the relative similarity between i and j with respect to the average similarity between i and its neighbors in S. This measures can be either positive or negative.

Definition 2.4.1 (Pavan and Pelillo, Node's weight)**.** Let $S \subseteq V$ be a nonempty subset of vertices and $i \in S$. The weight of i with regard to S is:

$$w_S(i) = \begin{cases} 1 & \text{if } |S| = 1 \\ \sum_{j \in S \setminus \{i\}} \phi_{S\setminus\{i\}}(j,i) w_{S\setminus\{i\}}(j) & \text{otherwise} \end{cases} \tag{2.5}$$

Further, the total weight of S is defined to be $W(S) = \sum_{i \in S} w_S(i)$.

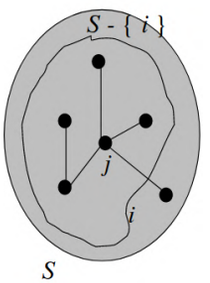


Figure 2.15: Weight of i respect to elements in S.

Note that $w_{\{i,j\}}(i) = w_{\{i,j\}}(j) = a_{ij}\ \forall i,j \in V \wedge i \neq j$. Then, $w_S(i)$ is calculated simply as a function of the weights on the edges of the sub-graph induced by S.

Intuitively, $w_S(i)$ gives a measure of the similarity between i and $S \setminus \{i\}$ with respect to the overall similarity among the vertices of $S \setminus \{i\}$. In other words, how similar (important) i is with respect to the entities in S. An important property of this definition is that it induces a sort of natural ranking among vertices of the graph.

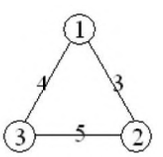


Figure 2.16: Similarity graph example.

Considering the graph proposed in Figure 2.16, we can derive a ranking between nodes:

$$w_{\{1,2,3\}}(1) < w_{\{1,2,3\}}(2) < w_{\{1,2,3\}}(3)$$

Definition 2.4.2 (Pavan and Pelillo, Dominant Set)**.** A nonempty subset of vertices $S \subset V$ such that $W(T) > 0$ for any nonempty $T \subseteq S$, is said to be a **dominant set** if:

- $w_S(i) > 0\ \forall i \in S$ (*internal homogeneity*)
- $w_{S\cup\{i\}}(i) < 0\ \forall i \notin S$ (*external homogeneity*)

These conditions correspond to cluster properties (**internal homogeneity** and **external in-homogeneity**). Informally we can say that the first condition requires that all the nodes in the cluster S are important (high weight, similar). The second one assumes that if we consider a new point in the cluster S, the cluster cohesiveness will be lower, meaning that the current cluster is already maximal.
By definition, dominant sets are expected to capture compact structures. Moreover, this definition is equivalent to the one of maximal clique problem when applied to unweighted graphs.

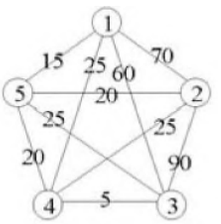


Figure 2.17: The set {1,2,3} is dominant.

2.4.2 Link to Standard Quadratic Optimization

Clusters are commonly represented as an n-dimensional vector expressing the participation of each node to a cluster. Large numbers denote a strong participation, while zero values no participation. In section 2.3 we have seen that the goal of clustering algorithm is to maximize the cohesiveness of the retrieved clusters. Formally speaking the goal can be expressed using the following optimization problem:

$$\begin{array}{ll} \text{maximize} & f(x) \\ \text{subject to} & x \in \Delta \end{array} \tag{2.6}$$

where A is a symmetric real-valued matrix with null diagonal and

$$\Delta = \{x \in \mathbb{R}^n : x \geq 0 \wedge e^\top x = 1\} \tag{2.7}$$

is the standard simple of $\mathbb{R}^n$. This yields the following standard quadratic problem in which local solution corresponds to a maximally cohesive cluster.
The entity x is a strict local solution of problem 2.6 if there exists a neighborhood $U \subset \Delta$ of x such that $f(x) > f(z)\ \forall z \in U \setminus \{x\}$. Then we define the support $\sigma(x)$ of $x \in \Delta$ as the index set of the positive components in x.

$$\sigma(x) = \{i \in V : x_i > 0\}$$

In other words $\sigma(x)$ is the set of vertices in V that belongs to the extracted cluster.

Definition 2.4.3 (Pavan and Pelillo, Characteristic vector)**.** A non-empty subset $C \subseteq V$ and C is a dominant set, admits a weighted **characteristic vector** $x^c \in \Delta$ if it has positive total weight $W(C)$, in which:

$$x_i^C = \begin{cases} \frac{W_c(i)}{W(C)} & \text{if } i \in C \\ 0 & \text{otherwise} \end{cases}$$

The important notion provided by Definition 2.4.3 is that also dominant set solutions belong to the standard simplex, as imposed in problem 2.6. The advantage is that, empirically, strict local maximizers of the dominant sets procedure work well in extracting clusters.

2.4.3 Link to Game Theory

Game theory is a theoretical framework used for examining and analyzing models of strategic interaction between competing rational actors. The clustering problem, as suggested by Pavan and Pelillo, can be formulated in terms of a game, also called *clustering game*, with the following properties:

- **Symmetric game**, the payoff of playing any strategy does not depend by the player but only by the strategy itself.
- **Complete knowledge**, players have complete knowledge about the game, they know what are the strategies that can be played and the corresponding payoffs.
- **Non-cooperative game**, players take independent decisions about the strategy to play, they don't make a priori alliance.
- Players play only **pure strategies**, meaning that they do not behave "rationally" but they take decisions in a pre-programmed pattern.

In the clustering game we have two players that want to extract the best structure of cluster from data samples. The pure strategies available to the players are the data points themselves in V and the similarity matrix A is used as the *payoff matrix* for the clustering game. The values A_{ij} and A_{ji} are the revenues obtained by player 1 and player 2 considering that they have player strategies $(i, j) \in V \times V$. Remember that the main diagonal of the similarity matrix is zero, meaning that $A_{ii} = 0$. A *mixed strategy* $x = (x_1, \ldots, x_n)^T \in \Delta$ is a probability distribution over the set of pure strategies, which models a stochastic playing strategy of a player. If player 1 and 2 play mixed strategies $(x_1, x_2) \in \Delta \times \Delta$, then the expected payoffs for the players are: $\mathbf{x_1^T A x_2}$ and $\mathbf{x_2^T A x_1}$ respectively. The goal of the two players of course is to maximize as much as possible their resulting revenue. During the game each player extract an object (i, j) and the resulting revenue is associated according to the payoff matrix A. Since we are considering A equal to the similarity matrix we can say that in order to maximize their revenue the two players would coordinate their strategies so that the extracted samples belong to the same cluster. In other words, only by selecting objects belonging to the same cluster, each player is able to maximize his expected payoff. The desired condition is that the two players reach a **symmetric Nash equilibrium**, that is state in which the two players agree about the cluster membership. A **Nash Equilibrium** is a mixed-strategy profile $(x_1, x_2) \in \Delta \times \Delta$ such that no player can improve the expected payoff by changing his playing strategy, given the opponent's strategy being fixed. This concept can be expressed with the following expression:

$$y_1^T A x_2 \leq x_1^T A x_2 \qquad y_2^T A x_1 \leq x_2^T A x_1 \qquad \forall (y_1, y_2) \in (V \times V).$$

A Nash equilibrium is **symmetric** if $x_1 = x_2$, meaning that considering a symmetric Nash Equilibrium $x \in \Delta$ the two conditions hold in a unique one:

$$y^T A x \leq x^T A x$$

The symmetric Nash equilibrium condition satisfies the internal homogeneity criterion required by the dominant set definition. However, it does not include any kind of constraint that guarantees the maximality condition. In order to satisfy

this condition it is necessary to look for a different type of Nash Equilibrium, known as **Evolutionary Stable Strategy (ESS)**.

Definition. A symmetric Nash equilibrium $x \in \Delta$ is an ESS if it satisfies also:

$$y^T Ax = x^T Ax \implies x^T Ay > y^T Ay \qquad \forall y \in \Delta \setminus \{x\}$$

$$y^T Ax = x^T Ax \implies x^T Ay < x^T Ax \qquad \forall y \in \Delta \setminus \{x\}$$

Even if the strategy y provides the same payoff of the strategy x, it is better to play x since the payoff against itself is greater than the one provided by y. The two strategies x and y represents two Nash Equilibrium, but only x is an ESS.
In conclusion we can say that the ESSs of the clustering game with affinity matrix A are in **correspondence** with dominant sets of the same clustering problem instance. However, we can also conclude that ESS's are in one-to-one correspondence to (strict) local solutions of StQP.
It is possible to say that ESS's satisfy the main characteristics of a cluster:

- **Internal coherency:** High support for all samples within the group.
- **External incoherency:** Low support for external samples.

2.4.4 Extracting Dominant Sets

One of the major advantages of using dominant sets is that it can be written with few lines of code, and moreover we can define different clustering approaches:

- Extracting a dominant set, done using the replicator dynamics procedure.
- Partitioning of the data points, obtained using the *peel-off* strategy, which means that at each iteration we extract a dominant set and the corresponding vertices are remove from the graphs. This is done until all vertices have been clustered (Partitioning-based clustering).
- Extracting overlapping clusters, obtained enumerating dominant sets.

In our applications we are going to deal with the second one, assuming that each entity belongs to a cluster. This assumption is required since the subject of this thesis is based on comparing three algorithms, but the first two (K-Means and Spectral clustering) are essentially partitioning-based algorithm.

The **Replicator Dynamics** are deterministic game dynamics that have been developed in evolutionary game theory. It considers an ideal scenario whereby individuals are repeatedly drawn at random from a large, ideally infinite, population to play a two-player game. Players are not supposed to behave rationally, but they act according to an inherited behavioral pattern (pure strategy). An evolutionary selection process operates over time on the distribution of behaviors [17].

Let $x_i(t)$ the population share playing pure strategy i at time t. The state of the population at time t is: $x(t) = (x_1(t), \ldots, x_n(t)) \in \Delta$.

We define an evolution equation, derived from Darwin's principle of nature selection:

$$\dot{x_i} = x_i \; g_i(x)$$

where g_i specifies the rate at which pure strategy i replicates, $\dot{x_i}$ is grow rate of strategy i.

$$\frac{\dot{x_i}}{x_i} \propto \text{payoff of pure strategy } i \text{ - average population payoff}$$

The most general continuous form is given by the following equation:

$$\dot{x_i} = x_i[(Ax)_i - x^T Ax]$$

where $(Ax)_i$ is the i-th component of the vector and $x^T Ax$ is the average payoff for the population. If we have a result better than the average strategy there's an improvement.

Theorem 2.4.4 (Nachbar,1990 TaylorandJonker,1978)**.** *A point $x \in \Delta$ is a Nash equilibrium if and only if x is the limit point of a replicator dynamics trajectory starting from the interior of Δ. Furthermore, if $x \in \Delta$ is an ESS, then it is an asymptotically stable equilibrium point for the replicator dynamics.*

Assuming that the payoff matrix A is symmetric ($A = A^T$) then the game is said to be doubly symmetric. Thanks to this assumption we can derive some conclusions:

- *Fundamental Theorem of Natural Selection (Losert and Akin,1983)*
  For any doubly symmetric game, the average population payoff $f(x) = x^T Ax$ is strictly increasing along any non-constant trajectory of replicator dynamics, meaning that $\frac{df(x(t))}{dt} \geq 0 \; \forall t \geq 0$, with equality if and only if $x(t)$ is a stationary point.

- *Characterization of ESS's (Hofbauer and Sigmund, 1988)*
  For any doubly symmetric game with payoff matrix A, the following statements are equivalent:

  - $x \in \Delta^{ESS}$
  - $x \in \Delta$ is a strict local maximizer of $f(x) = x^T Ax$ over the standard simplex Δ.
  - $x \in \Delta$ is asymptotically stable in the replicator dynamics.

A well-known discretization of replicator dynamics, which assumes non-overlapping generations, is the following (assuming a non-negative A):

$$x_i(t+1) = x_i(t)\frac{A(x(t))_i}{x(t)^T Ax(t)}$$

which inherits most of the dynamical properties of its continuous-time counterpart.

```
distance=inf;
while distance>epsilon
    old_x=x;
    x = x.*(A*x);
    x = x./sum(x);
    distance=pdist([x,old_x]');
end
```

Figure 2.18: MATLAB implementation of discrete-time replicator dynamics

The components of the converged vector give us a measure of the participation of the corresponding vertices in the cluster, while the value of the objective function provides of the cohesiveness of the cluster.

2.4.5 Dominant Sets Hierarchy

A useful extension of the Dominant Sets formulation is introduced in the optimization problem. The new formulation is now defined:

$$\begin{array}{ll} \text{maximize} & f_\alpha(x) = x'(A - \alpha I)x \\ \text{subject to} & x \in \Delta \end{array} \tag{2.8}$$

where $\alpha \geq 0$ is a parameter and I is the identity matrix.

The parameter α affects the number of clusters found by the algorithm. With an huge value of α each point defines a cluster since we require a strong cohesiveness. Instead decreasing the value of α the number of cluster increases.
The objective function f_α has now two kinds of solutions:

- solutions which correspond to dominant sets for original matrix A ($\alpha = 0$).
- solutions which don't correspond to any dominant set for the original matrix A, although they are dominant for the scaled matrix $A + \alpha(ee' - I)$. In other words, it allows to find subsets of points that are not sufficiently coherent to be dominant with respect to A, and hence they should be split.

This algorithm has the basic idea of starting with a sufficiently large α and adaptively decrease it during the clustering process following these steps:

1. Let α be a large positive value (ex: $\alpha > |V| - 1$).
2. Find a partition of the data into α-clusters.
3. For all the α-clusters that are not 0-clusters recursively repeat step 2 with decreasing α.

2.4.6 Properties

- **Well separation between structure and noise.** In such situations it is often more important to cluster a small subset of the data very well, rather than optimizing a clustering criterion over all the data points, particularly in application scenarios where a large amount of noisy data is encountered.
- **Overlapping clustering**. In some cases we can have that two distinct clusters share some points, but partitional approaches impose that each element cannot be long to more than one cluster.
- Dominant sets can be found by mining **local solutions**, so it is not necessary to look for global solutions.
- Deal very well in presence of noise.
- Strong connection with theoretical results.
- Makes no assumptions on the structure of the affinity matrix, being it able to work with asymmetric and even negative similarity functions.
- Does not require a priori knowledge on the number of clusters (since it extracts them sequentially).
- Leaves clutter elements unassigned (useful, e.g., in figure/ground separation or one-class clustering problems).
- Limiting the number of dynamic's iterations it is possible to detect quasi-click structures.

Chapter 3

Crafting Adversarial Threats

In the previous chapter we have discussed the history, real life implications and properties of adversarial machine learning. We have also defined different levels of attacker's knowledge and how an attacker can take advantage from it for fooling systems. From the best of our knowledge, the greatest part of the adversarial research is focused on studying supervised models. Indeed, only few works have been done for analyzing the robustness of unsupervised learning algorithms in presence of adversarial noise. In this chapter we are going to analyze two applications of clustering algorithms. The first one related to image segmentation and the second one to feature-based data clustering. In particular we are going to see that in both cases clustering algorithms can be fooled by using very well crafted adversarial perturbations. In this chapter we propose three algorithms useful for crafting adversarial examples against the two applications of clustering algorithms. The first and the second one are focused on crafting noise for fooling image segmentation systems. We will explain the reasons of using image segmentation in specific applications and what could be the implications of attacking those systems. The last one related to the capacity of fooling feature-based data clustering applications, and even in this case we will analyze sensitive application that requires security properties.

3.1 Genetic Algorithms

The three designed algorithms that we are going to discuss are based on the optimization of the adversarial noise. We will see that the resulting objective function is not derivable, for that reason genetic algorithms are used. Genetic algorithms (GAs) are a type of optimization algorithm based on principles of natural selection and genetics (Fraser, 1957; Bremermann, 1958; Holland, 1975). They are designed for finding optimal solutions to a specific problem that maximizes or minimizes a target objective function (or fitness function). They develop global search heuristic strategies based on biological processes of reproduction and natural selection. Each iteration of a genetic algorithm is also called generation and for each of them the fitness of every individual in the population is evaluated. Then the algorithm applies some stochastic operators (selection, crossover and mutation) in order to evolve the current generation to a new one. A generic algorithm does not guarantee convergence, indeed there are two stopping criteria. The first one is verified when the maximum number of generations has been reached. The second is verified when an acceptable fitness quality has been obtained for the current population.

Before going on it is necessary to introduce some extra notions:

- The fitness function is the function that we want to maximize or minimize. It is also called objective function.
- Individuals are candidate solutions.
- A population is a group of individuals that define a generation.
- Traits are the features of an individual.

In our work genetic algorithms are used for finding the best adversarial noise to inject inside the input data samples. Our individuals are real values, representing the perturbations to inject, and the fitness function is a measure that quantifies how the adversarial example is close to satisfy the attacker's goal.
In literature we can find multiple implementations of stochastic operators, and in general they work better when candidate solutions are encoded in vectors. In our experiments in order to evolve real values we decide to codify the integer part using the binary representation and the decimal part using strings.

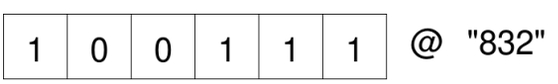


Figure 3.1: Encode for $X = 39.832$.

Then we have developed our custom crossover and mutation operators in order to work with this encoding.

Crossover Given two candidates X_1 and X_2 the crossover operator splits the two candidates and then combines different portions together in order to generate a new candidate. With our encoding we split the decimal and the integer part independently. The splitting point s is randomly chosen.

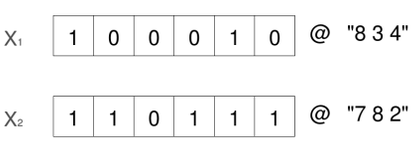


Figure 3.2: Candidate individuals X_1 and X_2 before crossover.

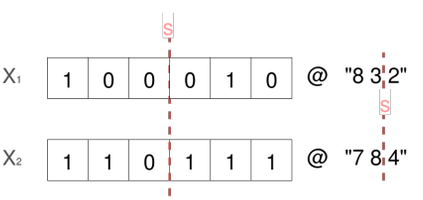


Figure 3.3: Candidate individuals X_1 and X_2 after crossover.

Mutation Mutation is performed only on the integer part, and it consists in flipping each variable value with probability p.

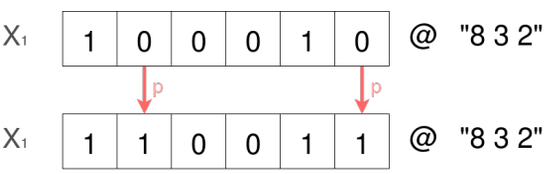


Figure 3.4: Mutation over candidate X_1 with $p = \frac{1}{3}$

The function `evolve` that we are going to see is the composition of these stochastic operators, designed for improving generation by generation the fitness function of the population.

3.2 Fooling Image Segmentation

Image segmentation is the arrangement of an image into different groups. Each pixel of an image is classified to a discrete region of the image. The resulting regions are strongly cohesive, meaning that pixels inside the same region have high similarity and preserve high contrast with respect to pixels of other regions. This definition is closely related to the one given for clustering algorithms provided in section 2. For that reason many research works in the area of image segmentation have been done on the usage of clustering algorithms for solving the segmentation problem. Note that clustering is not the unique way for solving the image segmentation problem, since we can also adopt other techniques which can be threshold based, edge-based, or neural network based.
Image segmentation deals with different kinds of datasets like images or video sequences. The most used application consists on segmenting images such a way that interesting areas can be extracted from the background. In many fields, like image processing, traffic analysis, pattern recognition and medicine image segmentation results to be extremely useful because at the same time it allows extracting essential information and removing noise and useless information.

Figure 3.5: Image segmentation for Automated driving systems.

Fig. 3.5, taken from [6], shows one of the main applications of image segmentation. Nowadays, autonomous driving systems are being studied and used with great interest and care. In those systems we can commonly find cameras, that collect traffic scenes, and an AI system that takes opportune decisions. In general, instead of providing original frames as input to the AI system, a sort of pre-pocessing or simplification of the scenes is done with segmentation algorithms. This step is used for removing possible noisy patterns and extract essential information, so that the final algorithm can easily detect possible obstacles.
Now think about the possibility that an attacker wants to inject adversarial noise into the scenes, so that the resulting segmentation will remove or forge essential

information. What if the attacker is able to remove a fundamental element from the scene?

Figure 3.6: Segmentation obtained by injecting adversarial noise.

In Fig. 3.6 we can see a scenario similar to Fig. 3.5, but now, due to the presence of adversarial noise, one of the three bikers has been associated to the background. This example highlights how dangerous can be the usage of weak algorithms or the absence of defensive strategies.
In this section we are going to analyze two algorithms used for crafting adversarial noise against clustering algorithms for image segmentation. We will see that the two realized algorithms differ for the nature of the noisy pattern that they generate.

3.2.1 Row-Based Adversarial Noise

The row-based algorithm has been designed with the goal of evaluating the robustness of clustering algorithms in presence of clear and evident noisy patterns. The algorithm basically tries to inject noisy rows-pattern, even detectable by human eyes, for obtaining errors in the final segmentation. Given an input image X the algorithm crafts an adversarial examples X', which is obtained by manipulating the s most sensitive rows. In order to evaluate the robustness of clustering algorithms different parameters $(\Delta, \alpha, \beta, s)$ are introduced. This extension allows us to simulate different levels of attacker's capacity and evaluate how the clustering algorithms react.

Algorithm 1 Row-based Adversarial Noise Generator

Input Data: $X \in \mathbb{R}^{n \times m}$, $Y \in \mathbb{R}^{n \times m}$, `clst`, p, Δ, s, α, β, G, `global`.
Output: $X' \in \mathbb{R}^{n \times m}$

$\mathbf{t} \leftarrow$ `sensitive_pixels(`X, Y, s`)` ▷ List of sensible rows.
$X' \leftarrow$ X
for $i \in [0, \ldots, \frac{m}{p}]$ **do**
    $\mathbf{c} \leftarrow [ip, \ldots, (i+1)p - 1]$ ▷ Range of columns.
    $\varepsilon \leftarrow$ `rand(`Δ`)`
    if `global` **then**
        $\pi \leftarrow 1, \ldots, m$ ▷ All the columns.
    else
        $\pi \leftarrow \mathbf{c}$
    end if
    $\varepsilon^* \leftarrow$ `optimize(`X, π, $\mathbf{t}$, $\mathbf{c}$, ε, `clst`, Δ, α, β, G`)`
    $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + \mathbf{1}\varepsilon^*$ ▷ Inject Adversarial Noise.
end for
return X'

Algorithm 2 Adversarial Noise Optimizer

Input Data: $X \in \mathbb{R}^{n \times m}, \pi, \mathbf{t}, \mathbf{c}, \varepsilon$, `clst`, Δ, α, β, G.
Output: ε^*

$\hat{Y} \leftarrow$ `clst`$(X_{*,\pi})$
$X' \leftarrow X$
for $i \in [0, \ldots, G]$ **do**
    $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + \mathbf{1}\varepsilon^*$
    $Y' \leftarrow$ `clst`$(X'_{*,\pi})$
    $\sigma \leftarrow$ `eval(`$\hat{Y}, Y'$`)` ▷ Miss-segmentation score
    $\phi(\varepsilon, \sigma) \leftarrow \alpha\sigma + \beta||\Delta - \varepsilon||$ ▷ Noise score.
    $\varepsilon \leftarrow$ `evolve(`ε, ϕ`)`
end for
$\varepsilon^* \leftarrow \arg\max_\varepsilon \phi$
return ε^*

where:

- $\mathbf{t}$ is the vector of s most sensible rows to attack.
- $\mathbf{c}$ is the vector of consecutive p columns.
- π is the projection vector. It depends on the type of optimization (global or local).
- X is the input image for which the attacker wants to break the segmentation.
- $X_{\mathbf{t},\mathbf{c}}$ defines a portion of the input image X, obtained by picking rows in $\mathbf{t}$ and columns in $\mathbf{c}$.
- Y is the true segmentation. Y_{ij} specifies the cluster label in which the pixel (i, j) belongs.
- `clst` is the clustering algorithm used for segmenting the input image X that the attacker wants to fool.
- p number of consecutive columns to attack jointly. The greater it is, the lower the computational cost is.
- Δ defines the noise threshold. The greater it is, the greater the attacker's capacity is, meaning that a stronger adversarial noise can be injected.
- s is the number of rows to attack.
- G is the number of generations used for optimizing the noise perturbation.
- $\sigma = \texttt{eval}\,(\hat{Y}, Y')$ get a score about how much the initial predicted segmentation $\hat{Y}$ and the adversarial one Y' are dissimilar.
- $\phi(\varepsilon, \sigma)$ is the objective function that measures, through a weighted mean, the effectiveness of noise ε in X. The goal of the adversarial algorithm is to maximize this quantity.
- α is the importance of the dissimilarity between $\hat{Y}$ and Y'. A very strong attacker wants to use large values of α. In contrast, low values of α will bring the optimizer to leave the input image X as unchanged as possible.
- β is the importance of choosing a small ε. Force the optimizer to find the best adversarial noise but limit the number and strength of adversarial transformations as much as possible,
- `global` is a boolean variable that specifies if the optimization procedure has to consider the entire image or only a limited part. The results of the two optimization procedures are different. If `global` is equal to FALSE then the optimizer returns a spatially local optimal solution, otherwise it returns a spatially global optimal solution. Moreover, when `global=FALSE` then $\pi = \mathbf{c}$, meaning that the clustering has to be evaluated locally in the portion $X_{*,\mathbf{c}}$. Otherwise, if `global=TRUE` then $\pi = *$, meaning that the segmentation has to be evaluated over all the m columns.

- X' is the crafted adversarial example.

- ε^* defines the optimal adversarial perturbation for X.

- $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + \mathbf{1}\varepsilon^*$ injects noise ε^* in rows $\mathbf{t}$ and columns $\mathbf{c}$ of X.

From the pseudo-code proposed in Alg. 1 we can notice the usage of two fundamental functions: `sensitive_pixels` and `optimize`. The first one returns a list containing the most s sensitive rows in X. It is based on the usage of some heuristic, carefully crafted considering the similarity measure adopted and the input image. Considering a grayscale image with just two clusters, background (black) and foreground (white), a possible heuristic could be to get pixels in the foreground with the lowest intensity. The second one, `optimize`, is a genetic algorithm, described in Alg. 2, used for finding the best noise alteration ε^* to inject in X. Starting with a population of candidate solutions it applies, generation by generation, stochastic operators (selection, mutation and crossover) in order to evolve the initial solution ε to a better one ε^* which maximizes the objective function $\phi(\varepsilon, \sigma)$. It does not guarantee to find a global optimum, but at least a local one. The greater the number of generations G is, the greater the probability of finding the global one is. Note that, in order to obtain a score about the goodness of the adversarial permutation, it is necessary to compute the clustering result on the adversarial image X' at each iteration and compare, using the function `eval`, the difference between the original labeling and the adversarial one.
In chapter 4 will be discussed the effectiveness of each hyper-parameter and how much the obtained adversarial perturbations are sensible to them.

The most noticeable limit of Alg. 1 is that it is computationally costly, since we are trying to optimize a certain function that requires the application of the clustering algorithm on the input X at each generation. The greater G and X is, the greater the computational complexity is. For that reason the local optimization strategy (`global=FALSE`) has been developed, for reducing the size of the input X provided to the optimizer.

The last consideration of this section is related to the role of Y. In most of the clustering applications the true segmentation Y is unknown. For that reason the attacker can decide to estimate it using clustering algorithms or other learning models. However, the realized algorithm is able to work without that knowledge, since the latter is only used for the development of `sensitive_pixels` heuristic function. Other heuristics can be taken into consideration according to the attacker's knowledge, without affecting the rest of the designed adversarial algorithm.

Example of Application Let's take into consideration the image X showed in Fig. 3.7, taken randomly from the MNIST dataset. Now applying the spectral clustering algorithm over X we obtain the segmentation proposed in Fig. 3.8. We can see that the obtained segmentation $\hat{Y}$ is very good, the algorithm has correctly separated the foreground from the background.

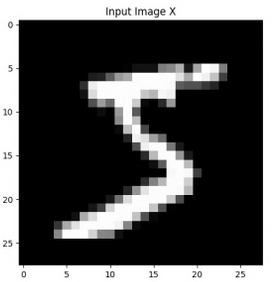


Figure 3.7: Digit 5 from MNIST dataset.

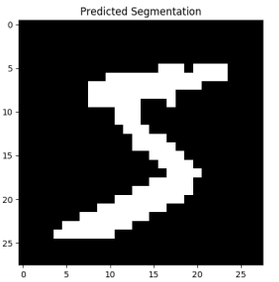


Figure 3.8: Segmentation using Spectral Clustering.

Now we consider the scenario in which the attacker wants to obtain errors in the final segmentation. For doing that the attacker uses Alg. 1 for injecting noisy rows in X. The corresponding adversarial example X' is shown in Fig. 3.9, and we can see that it is very similar to the original input image X. To the human eyes, the two images might represent the same digit, but something strange happens in the case above. Indeed, Fig. 3.10 shows the segmentation Y' obtained using the spectral clustering algorithm on X'. For some reasons the Y' and $\hat{Y}$ are strongly different, but X and X' are really similar for human eyes.

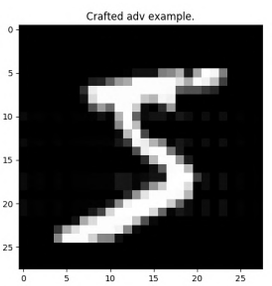


Figure 3.9: Adversarial Digit 5.

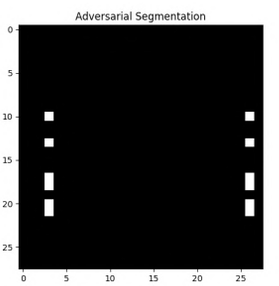


Figure 3.10: Segmentation using Spectral Clustering.

Just looking at Fig. 3.9 we are not able to detect where the injected noise is located, and how strong it is. After some experiments we discovered that changing the color map used for showing X', from grayscale to jet, it is easier to detect the noisy pattern. Indeed in Fig. 3.11 it is simpler to detect the adversarial noise. We can see that the injected noise is similar to the background and for that reason in the grayscale representation we are not able to detect it. With respect to Fig. 3.12 in fact we can notice that in Fig. 3.11 6 rows have been transformed by Alg. 1.

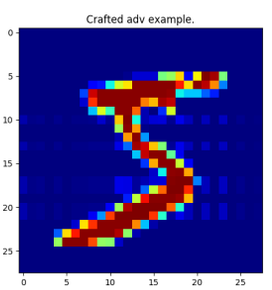


Figure 3.11: Digit 5.

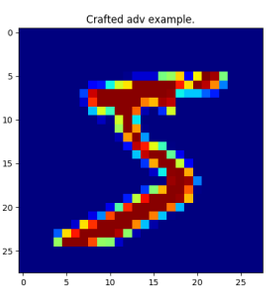


Figure 3.12: Adversarial Digit 5.

3.2.2 Pixel-Wise Adversarial Noise

The pixel-wise version is used for crafting adversarial examples in which no clear noisy patterns are injected. The pixel-wise algorithm has been designed for injecting noise in such a way that it seems to be random. Remember that even in real life we do not always have clear images. As a matter of fact, we can find different types of noise coming from multiple sources: camera quality, illumination, brightness. Indeed, in literature have been defined different types of noise that can be founded inside images, like gaussian or salt-and-pepper noise. The goal of pixel-wise adversarial algorithm is to craft an adversarial noise mask that is indistinguishable from a random noise pattern but which is able to fool clustering algorithms. With respect to a random noise mask, the adversarial noise is targeted and attacks only sensitive regions of the input data.

Algorithm 3 Pixel-Wise Adversarial Noise Generator

Input Data: $X \in \mathbb{R}^{n \times m}$, $Y \in \mathbb{R}^{n \times m}$, `clst`, p, Δ, s, α, β, G, `global`.
Output: $X' \in \mathbb{R}^{n \times m}, \mathbf{t}$

1: $X' \leftarrow$ X
2: $\mathbf{t}_{in} \leftarrow$ `inner_sensitive_pixels(`X, Y, s`)`
3: $\mathbf{t}_{out} \leftarrow$ `outer_sensitive_pixels(`X, Y, s`)`
4: $X' \leftarrow$ `craft_adv_pixels(`$X, \mathbf{t}_{in}$`)`
5: $X' \leftarrow$ `craft_adv_pixels(`X'`,` $\mathbf{t}_{out}$`)`
6: $\mathbf{t} \leftarrow [\mathbf{t}_{in}, \mathbf{t}_{out}]$
7: **return** $X', \mathbf{t}$

Algorithm 4 Craft Adversarial Pixels

Input Data: $X \in \mathbb{R}^{n \times m}, \mathbf{t}, \varepsilon$, `clst`, Δ, α, β, G.
Output: $X' \in \mathbb{R}^{n \times m}$

1: s $\leftarrow |\mathbf{t}|$ ▷ Counts the number of pixels to attack.
2: $X' \leftarrow$ X ▷ Adv. Example
3: **for** $i \in [0, \ldots, \frac{s}{p}]$ **do**
4: $\quad \mathbf{c} \leftarrow$ `columnsOf(`$\mathbf{t}[ip], \ldots, \mathbf{t}[(i+1)p-1]$`)` ▷ Get p-continuous pixels.
5: $\quad \varepsilon \leftarrow$ `rand(`Δ`)`
6: $\quad$ **if** `global` **then**
7: $\quad\quad \pi \leftarrow 1, \ldots, m$ ▷ All columns.
8: $\quad$ **else**
9: $\quad\quad \pi \leftarrow \mathbf{c}$
10: $\quad$ **end if**
11: $\quad \varepsilon^* \leftarrow$ `optimize(`X, π, $\mathbf{t}$, $\mathbf{c}$, ε`,` `clst`, Δ, α, β, G`)` ▷ See Alg. 2
12: $\quad X'_{t,\mathbf{c}} \leftarrow X'_{t,\mathbf{c}} + \mathbf{1}\varepsilon^*$
13: $\quad$ **return** X'
14: **end for**

where:

- s is the number of inner and outer pixels to attack.
- p is the number of consecutive pixels to attack jointly.

- `columnsOf(S)` gets a list of column over S.

  $S = \{(i,j)|i \in 1,\ldots,n \wedge j \in 1,\ldots,m\}$ `columnsOf`$(S) = \{j|(i,j) \in S\}$

  `columnsOf`$(\{[1,2],[1,3],[5,4]\})$=$\{2,3,4\}$

Note that the time complexity of the entire algorithm also depends on s and p. In line 4 it is possible to see that the optimizer executes s/p iterations. The other parameters have the same meaning described for Alg. 1 in section 3.2.1.
We can immediately see that Alg. 3 uses exactly the same optimizer described in Alg. 2. In the pixel-wise version three new functions are used: `craft_adv_pixels`, `inner_sensitive_pixels` and `outer_sensitive_pixels`. The first one is strongly similar to the code showed in Alg. 1 from line 16 to 26, but now it is defined inside a named function, since it is used twice. The only difference with respect to the previous algorithm is that it manipulates only the s pixels and not the entire rows. The second one returns the most sensitive pixels that belong to a targeted cluster (for instance inside the foreground). Conversely, the last one returns the most sensitive pixels that belongs to another cluster but are close to the targeted one. These two functions implement possible attacker's heuristics for finding sensitive pixels. In section 4 we will see how they have been developed for our experiments. Note that the list of targeted pixels T is exactly the combination of inner sensitive pixels T_{in} and outer sensible pixels T_{out}.

Example of Application In order to better explain how the last two functions can be implemented we are going to consider a classic problem in image segmentation: background/foreground separation for images in MNIST dataset.

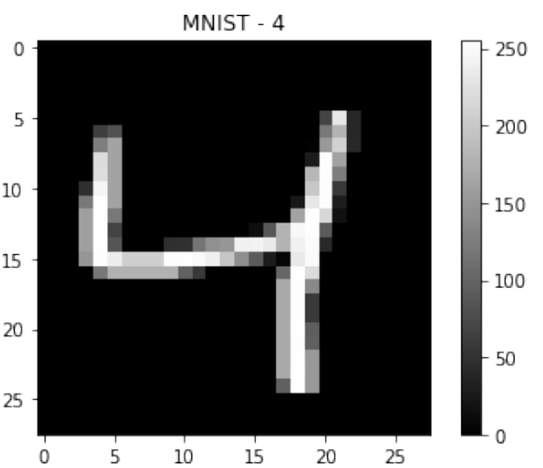


Figure 3.13: MNIST example.

The first function `inner_sensitive_pixels` returns pixels inside the foreground (white) and conversely the `outer_sensitive_pixels` returns pixels in the background that are spatially close to the foreground.

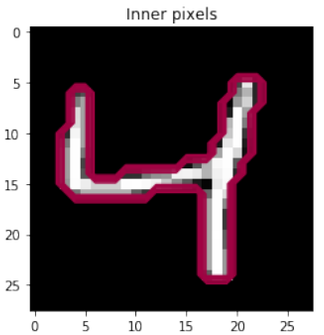


Figure 3.14: Inner pixels mask.

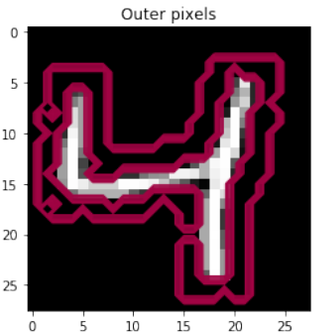


Figure 3.15: Outer pixels mask.

The red region highlighted in Fig. 3.14 (that corresponds to the border of digit 4) shows the part from which are extracted inner sensible pixels. Instead, the red mask highlighted in Fig. 3.15 is external to the foreground and from that portion of the image are extracted the outer pixels.

3.3 Fooling Data Clustering

Clustering has been used for different applications, like pattern recognition, spatial data analysis, image processing, market analysis, etc. Furthermore, nowadays clustering algorithms have been increasingly adopted in security for detecting possible dangerous or illicit activities, in particular for malware categorization. Malwares are categorized by their intent and this allows to respond to a threat quickly, since the removal techniques are similar inside the same category. In signal processing data clustering algorithms are used for compression. All of these applications are sensitive and malicious attacks can end up being dangerous. Let's take into consideration a message compression scenario: the attacker can introduce some noisy words into the secret text in order to obtain a certain target compression, but then the decompression would decode it to a different message, possibly with a completely different meaning.

Algorithm 5 Adversarial Target Clustering

Input Data: $X \in \mathbb{R}^{n \times m}$, $Y \in \mathbb{R}^n$, `clst`, p, Δ, s, α, β, G, `global`.
Output: $X' \in \mathbb{R}^{n \times m}$, T

$c_0 \leftarrow \texttt{centroid}(X_{Y=0,*})$
$c_1 \leftarrow \texttt{centroid}(X_{Y=1,*})$
$d \leftarrow \texttt{sign}\,(c_0 - c_1)$ ▷ Get orientation vector.

$\mathbf{t} \leftarrow \texttt{sensitive\_points}\,(X, Y, s)$ ▷ List of sensible sample.
$X' \leftarrow$ X
for $i \in [0, \ldots, \frac{m}{p}]$ **do**
    $\mathbf{c} \leftarrow [ip, \ldots, (i+1)p - 1]$ ▷ Range of features.
    $\varepsilon \leftarrow \texttt{rand}\,(\Delta)$
    if `global` **then**
        $\pi \leftarrow 1, \ldots, m$ ▷ All the features.
    else
        $\pi \leftarrow \mathbf{c}$
    end if
    $\varepsilon^* \leftarrow \texttt{optimize}\,(X,\ \pi,\ \mathbf{t},\ \mathbf{c},\ d_\pi,\ \varepsilon,\ \texttt{clst},\ \Delta,\ \alpha,\ \beta,\ G)$
    $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + d_{\mathbf{c}}\varepsilon^*$
end for
return $X', \mathbf{t}$

where:

- $\mathbf{t}$ is the vector of s most sensitive samples to attack.
- $\mathbf{c}$ is the vector of consecutive p columns.
- π is the projection vector. It depends on the type of optimization (global or local).
- X is the input collection for which we want to break the clustering result.
- $X_{\mathbf{t},\mathbf{c}}$ portion of the input dataset X, obtained by picking rows in $\mathbf{t}$ and features in $\mathbf{c}$.

Algorithm 6 Adversarial Target Noise Optimizer

Input Data: $X \in \mathbb{R}^{n \times m}, \pi, \mathbf{t}, \mathbf{c}, d, \varepsilon$, `clst`, Δ, α, β, G.
Output: ε^*

$\hat{Y} \leftarrow \texttt{clst}(X_{*,\pi})$
$X' \leftarrow X$
for $i \in [0, \ldots, G]$ **do**
    $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + \varepsilon d$
    $Y' \leftarrow \texttt{clst}(X'_{*,\pi})$
    $\sigma \leftarrow \texttt{eval}\,(\hat{Y}, Y')$ ▷ Get miss-clustering score
    $\phi(\varepsilon, \sigma) \leftarrow \alpha\sigma + \beta||\Delta - \varepsilon||$
    $\varepsilon \leftarrow \texttt{evolve}\,(\varepsilon, \phi)$
end for
$\varepsilon^* \leftarrow \arg\max_\varepsilon \phi$
return ε^*

- Y is the true clustering vector. Y_i specifies the cluster label in which sample i belongs.
- `clst` is the clustering algorithm that the attacker wants to fool.
- p is the number of consecutive features to attack jointly. The greater it is, the lower the computational cost is, but from an attacker point of view it would be better to optimize each pixel independently.
- Δ noise threshold. The greater it is, the greater the attacker's capacity of injecting adversarial noise is.
- s number of samples to manipulate.
- G number of generations used for optimizing the noise perturbation.
- $\sigma = \texttt{eval}\,(\hat{Y}, Y')$ get a score about how much the initial predicted labels $\hat{Y}$ and the adversarial one Y' are dissimilar.
- $\phi(\varepsilon, \sigma)$ objective function that measures, through a weighted mean, the effectiveness of noise ε in X. The goal of the adversarial algorithm is to maximize this quantity.
- α is the importance of the dissimilarity between $\hat{Y}$ and Y'. A very strong attacker wants to use large values of α. In contrast, low values of α will bring the optimizer to leave the input image X as unchanged as possible.
- β, is the importance of choosing a small ε. Forces the optimizer to find the best adversarial noise, but limiting as possible the number and strongness of adversarial transformations.
- `global`, it is a boolean variable that specifies if the optimization procedure has to consider the entire set of features or only a limited part. The results of the two optimization procedure are different, if `global` is equal to FALSE then the optimized returns a spatially local optimal solution, otherwise it

returns a spatially global optimal solution. Moreover, when `global=FALSE` then $\pi = \mathbf{c}$, meaning that the clustering has to been evaluated locally in the portion $X_{*,\mathbf{c}}$. Otherwise, if `global=TRUE` then $\pi = *$ means that the clustering labeling has to be evaluated over all the m features.

- X' crafted adversarial dataset, in which samples $\mathbf{t}$ have been attacked.
- ε^* optimal adversarial perturbation for X.
- $X'_{\mathbf{t},\mathbf{c}} \leftarrow X_{\mathbf{t},\mathbf{c}} + d_{\mathbf{c}}\varepsilon^*$ injects noise ε^*, with direction $d_{\mathbf{c}}$, in samples $\mathbf{t}$ and features $\mathbf{c}$ of X.
- $c_k \leftarrow$ `centroid(`$X_{Y=k,*}$`)` is the centroid vector of the elements in cluster k. In this implementation we consider just two clusters but the algorithm can be easily extended with more than two clusters.
- `sensitive_points` returns a list of targeted samples that satisfy a certain criterion or heuristic. For more detailed information see Section 4.3.
- d is the orientation vector, for which occurs:

$$d_i = sign(c_{0_i} - c_{1_i}) = \begin{cases} -1, & c_{0_i} < c_{1_i} \\ 0, & c_{0_i} = c_{1_i} \\ 1, & \text{otherwise} \end{cases}$$

which means that if d_i is $+1/-1$ the optimizer should find a positive/negative noise perturbation to inject in feature i for moving sensible entities from c_0 to c_1. If d_i is 0 it means that the entities are in the right position for the feature i.
In order to better understand the general behavior of the last algorithm we propose some pictures of a toy example. Let's take into consideration the following dataset X:

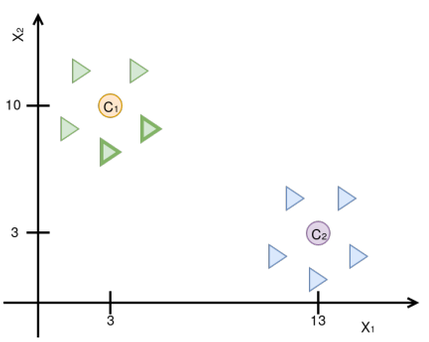


Figure 3.16: Data points X with features x_1 and x_2.

We can see from Fig. 3.16 that points in X are easily clustered in two distinct clusters with two centroids C_1 and C_2. The goal of the attacker algorithm is to move sensible points (highlighted) from the green cluster towards the blue one for fooling the clustering algorithm.

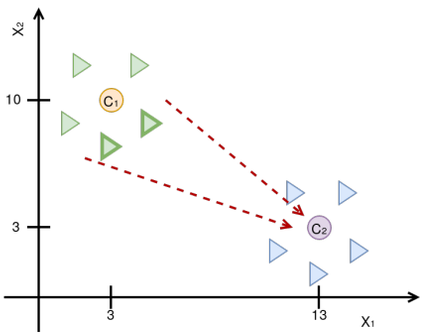


Figure 3.17: Trajectories from C_1 to C_2.

In Fig. 3.17 two red dashed lines define the optimal direction for moving green points towards the blue cluster. The vector d basically represents these two lines, gives features direction. We can notice that in this example $d = sign(c_1 - c2) = (+1, -1)$ meaning that it is necessary to reduce increase the x_1 component and decrease the x_2 one for moving a green point towards the blue cluster.

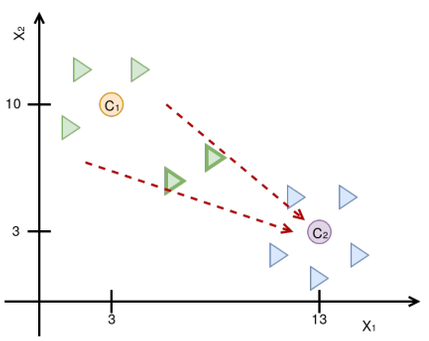


Figure 3.18: Adversarial perturbation on green points.

Fig. 3.18 shows the application and the final result obtained by the adversarial algorithm. We will see that the greater the attacker's capacity is, the closer green points are to the blue cluster.
Below we propose an example of using Alg. 5 using the MNIST dataset with digits 0 and 6. The algorithm was executed with the goal of moving images from the 6-digit cluster to the 0-digit one.

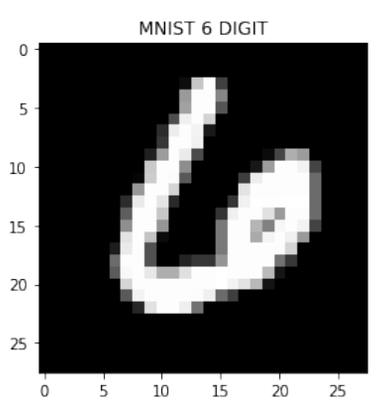


Figure 3.19: MNIST Adversarial Digit 6

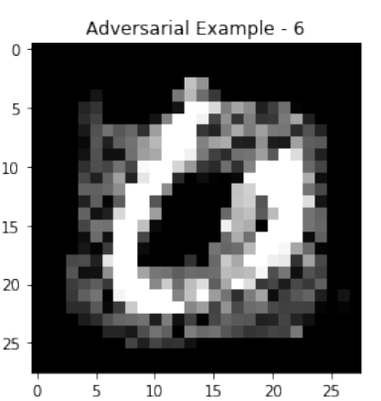


Figure 3.20: MNIST Adversarial Digit 6

Fig. 3.19 shows the original data image, which is labeled with 6. Instead, in Fig. 3.20 we can find the adversarial examples obtained by executing Alg. 5 to the previous image. We can see that now the shape of digit 6 is surrounded by other pixels that tend to assume the shape of a 0-digit.

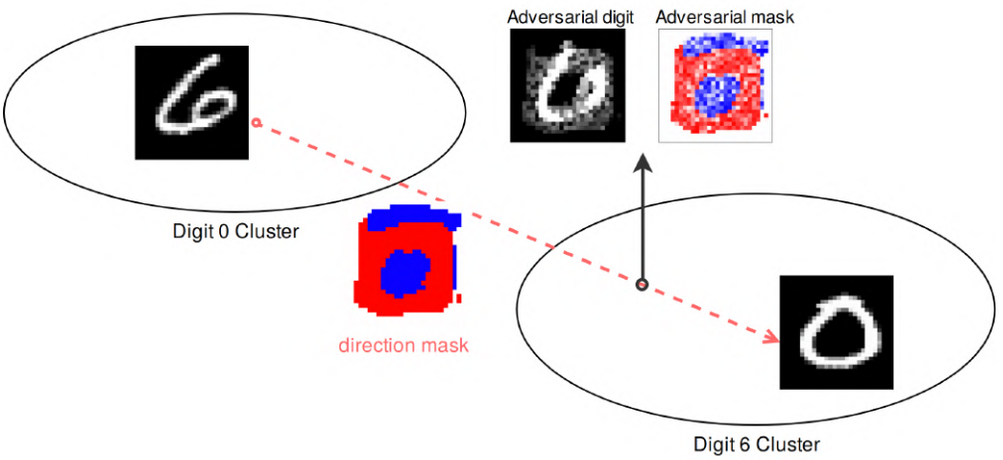


Figure 3.21: MNIST Adversarial Representation

Fig. 3.21 shows, from a geometrical point of view, how the adversarial algorithm works with the MNIST dataset. The algorithm is executed with the goal of moving a digit from cluster "0" to cluster "6". The first step consists in computing the direction mask, showed in Fig. 3.21 and in Fig. 3.22, then using the optimizer the starting image of digit-6 Fig. 3.19 is perturbed for obtaining a new adversarial example Fig. 3.20. The adversarial mask obtained, showed in Fig. 3.23, is the difference between X and X' and we can see that it is similar to the direction mask, meaning that the optimizer is working correctly.

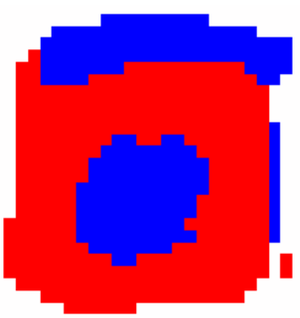


Figure 3.22: Direction Mask.

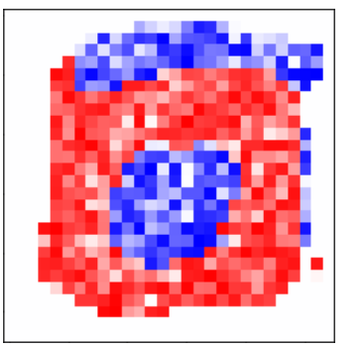


Figure 3.23: Adversarial Noise Mask.

Another key point that guarantees the empirical correctness of the algorithm is given by Fig. 3.23. The mask M is composed by red pixels and blue pixels. If M_{ij} is red it means that pixel in position (i, j) needs to increase its intensity. Otherwise if M_{ij} is blue it means that the corresponding pixel needs to decrease the intensity value. We can also notice that the composition of red pixels follows exactly the shape of a "0" digit, instead the composition of the blue ones represents the parts to remove from digit "6" for obtaining a "0".

Chapter 4

Experiments and Analysis

In this chapter we are going to analyze how the three designed algorithms, discussed in chapter 3, have been developed. Moreover, we are going to discuss the results obtained using the three algorithms in different contexts, analyzing the resulting robustness provided by the three algorithms: K-Means, Spectral and Dominant Sets clustering. In the first part of this chapter we will take into consideration the first two algorithms (Alg. 1, 3), used for fooling image segmentation, and in the last part we will address the third algorithm (Alg. 5), used for fooling data clustering. We will analyze the robustness provided by the three algorithms for all of the scenarios with the goal of establishing a sort of hierarchy of robustness between them.

4.1 Row-Based Adversarial Algorithm

The first algorithm, described in Section 3.2.1, allows the attacker to inject noisy patterns in the input image X. In particular, a key point of this algorithm is to detect whether some rows are sensitive to adversarial noise perturbations, basing on some heuristic strategies. In this section we are going to analyze how target rows are identified, how adversarial noise is optimized and we will analyze the results obtained for the MNIST dataset.

4.1.1 Choice of Target

In our experiments the s most sensitive rows are the ones which contains pixels in the background with the highest intensity value. Formally speaking, given an $n \times m$ image, let us define $S = \{(i, p_i)\}$ as the set of all the tuples whose first element is the row number and the second is the intensity of the background pixel in the i-th row which has the highest intensity. Then, we say that the s most sensitive rows are the s rows belonging to S which have the highest intensity value p. Our goal is to increase the intensity of the background pixels so that they will be clustered together with the foreground pixels. The reason for choosing pixels with the highest intensity comes from the idea that in general the greater their intensity is, the closer are with respect to the foreground intensity pixels. We assume that the attacker wants to fool the segmentation algorithms in such a way they will expand the foreground or, in the worst case, they will detect a different foreground shape with respect to the original one. Let's take into consideration an input digit X from the MNIST dataset:

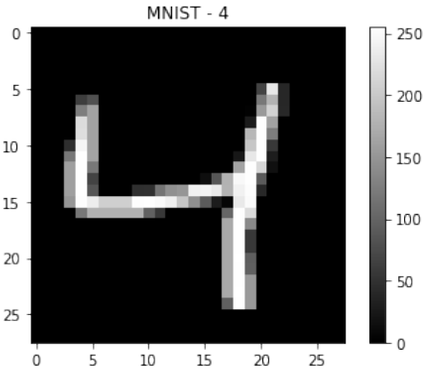


Figure 4.1: Digit "4" from MNIST.

Which is commonly encoded with an $n \times m$ matrix of real values:

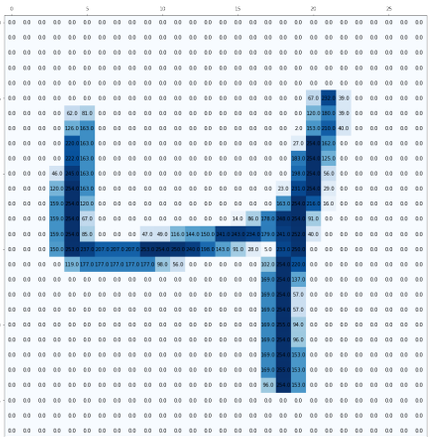


Figure 4.2: Encode of digit "4".

Fig. 4.2 shows the common encoding usually adopted for representing gray scale images in digital systems. Then the most sensitive rows are below highlighted:

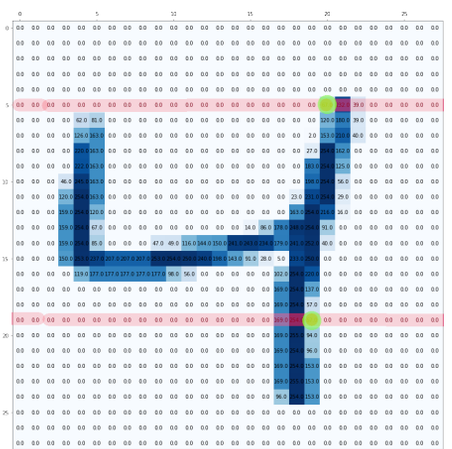


Figure 4.3: 2 Most sensitive rows for input X.

The two green points in Fig. 4.3 identifies the two pixels, grouped together in the background by the clustering algorithm, with the highest intensity. The red rows are the ones which will be perturbed by the algorithm, since they contain the 2 most sensitive green pixels. The resulting perturbation, as we said in Section 3.2.1,

depends on different parameters (Δ, s, `clst`, `full`, ...). For example we could obtain something like this:

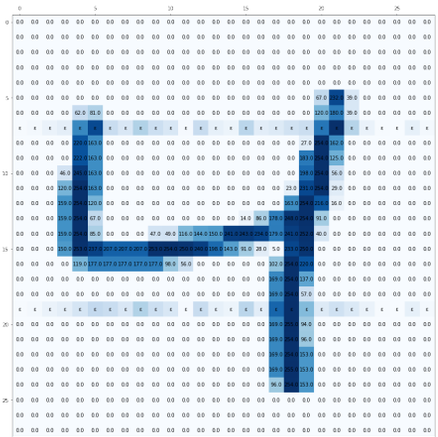


Figure 4.4: Adversarial example X'.

The two most sensitive rows have been influenced by the corresponding optimal adversarial noise ε^*, giving origin to the adversarial example X'. Note that in this example the adversarial noise injected is not so strong, but choosing accurately the two parameters Δ and α the attacker could be able to inject a stronger perturbation. The greater these two quantities are, the greater the attacker's capacity is, meaning that a stronger perturbation can be injected.

4.1.2 Local and Global optimization

In Section 3.2.1 we discussed the role of the `global` parameter for Alg. 1. In this section we are going to see how the crafted adversarial examples change according to this parameter.

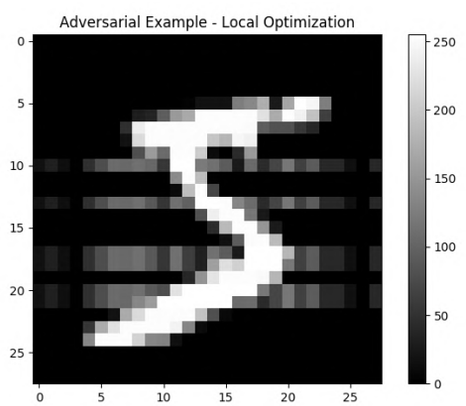


Figure 4.5: Local optimization adversarial example.

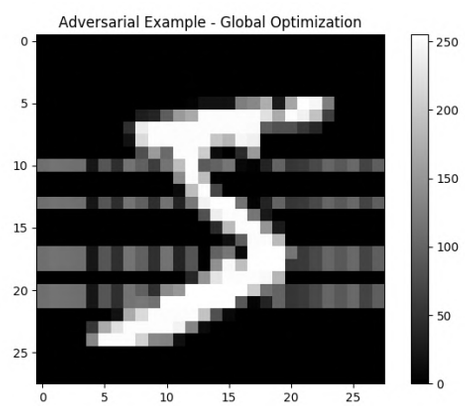


Figure 4.6: Global optimization adversarial Example.

The role of the `global` parameter is not only related to the time complexity, but we can see from the two previous images that also the resulting perturbation is different. When `global` is equal to `FALSE` we are in the case of local optimization, meaning that the optimal noise is optimal only for the p consecutive columns, regardless of the rest of the image. Otherwise, when `global=TRUE`,

the optimizer has to find an optimal perturbation for those p consecutive columns with respect to the entire image. From a practical point of view if the p columns contain only small intensities then a soft perturbation can be sufficient for breaking the local segmentation. Indeed we can see in Fig. 4.5 that the major level of perturbation is concentrated around the foreground, leaving the external columns quite unchanged. Conversely, in Fig. 4.5 even the external columns have been strongly transformed. Considering the attacker point of view, the adversarial example shown in Fig. 4.5 seems to be stronger for fooling the segmentation than the other one. On the other hand, if the attacker also wants to hide threats as much as possible then the local optimization seems to be more appropriate because it limits the perturbations on parts that are not important (like the external columns). This second scenario allows also the attacker to craft adversarial examples for which could be more difficult to detect adversarial perturbations. An example was previously described with Fig. 3.11. In that case the optimization was done in a local way, indeed it is very difficult for human eyes to detect where threats are located.

4.1.3 MNIST results

The evaluation of robustness provided by the three clustering algorithms (Dominant Sets, K-Means and Spectral Clustering) has been done using a random sample of the MNIST dataset. MNIST contains images of handwritten digits, from 0 to 9. Each sample in MNIST is a 28×28px gray scale image where the background is completely dark and the foreground has high intensity. The similarity measure used for the three algorithms is based on the distance between pixels intensities.

$$A(i,j) = \exp\left(\frac{-||I(i) - I(j)||_2^2}{\sigma^2}\right)$$

where:

- $I(i) \in [0, 255]$, intensity of pixel i in X.
- σ, scaling factor fixed to the standard deviation of the input X.

We decided to use clustering algorithms for segmenting digits with the goal of correctly splitting the background from the foreground. We noticed that in the natural setting the three algorithms work quite well. In order to evaluate the robustness in presence of adversarial noise we tested the three algorithms with multiple tests, each of which characterized by different maximum level of power noise Δ and the number of rows s. The other parameters were fixed according to the following description:

- $p = 1$, no joint optimization.
- `global = False`, local optimization.
- $\alpha = \beta = 0.5$, no privilege to a specific objective component for ϕ.
- $G = 20$

Each test has been repeated multiple times using different initial random seeds. Mean values and standard deviation over the multiple runs are considered in the following plots.

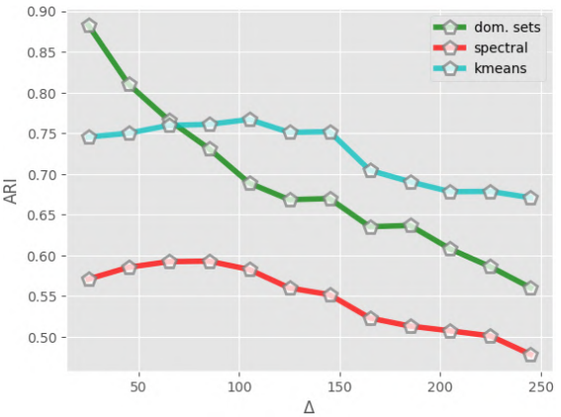


Figure 4.7: ARI over power noise Δ.

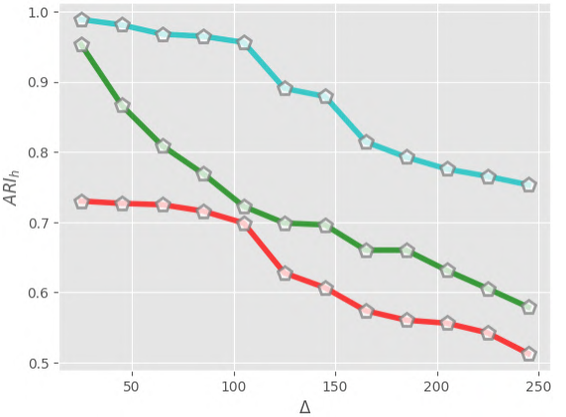


Figure 4.8: ARI_h over power noise Δ.

Fig. 4.7 and Fig. 4.8 show how the 3 clustering algorithms react in presence of adversarial noise. The x-axis represents the maximum level of perturbation Δ and the y-axis instead represents a similarity measure between two clustering labeling. In Fig. 4.7 we use the ARI measure, proposed in [11], instead in Fig. 4.8 we use the ARI_h measure. ARI (Adjusted Rand Index) considers all pairs of samples and counts the number of pairs that are assigned to the same or different clusters in the predicted and true clustering labeling. It has been introduced as the adjusted version of the RI measure. Given a set of n samples in X and two clustering labeling to compare Y and Y', we define the Rand Index (RI) in the following way:

$$RI = \frac{a+b}{\binom{n}{2}}$$

where:

- a is the number of pairs of samples that are in the same cluster in Y and in the same cluster in Y'.
- b is the number of pairs of samples that are in different groups in Y and in different ground in Y'.
- $\binom{n}{2}$ is the total number of pairs.

The numerator $a+b$ can be considered as the number of pairs-agreements between Y and Y'.

$$b = |S_B| \qquad \text{where} \quad S_B = \{(o_i, o_j) : o_i \in Y_k, o_j \in Y_l \ \wedge \ o_i \in Y'_k, o_j \in Y'_l\}$$

$$a = |S_A| \qquad \text{where} \quad S_A = \{(o_i, o_j) : o_i, o_j \in Y_k \ \wedge \ o_i, o_j \in Y'_k\}$$

where Y_k refers to a set of cohesive entities in the k-cluster in Y. The Adjusted Rand Index is the adjusted version of the RI, meaning that it is compared with respect to a random clustering labeling. In our experiments we consider as the true segmentation labeling Y all the pixels with intensity greater than 0. Then we consider the segmentation obtained on the adversarial example for the Y' labeling. The smaller the $ARI(Y, Y')$ measure is, the closer Y' is to a random labeling,

otherwise if they agree then ARI is exactly 1. In other words, the smaller it is the farthest are the two predictions, meaning that the adversarial noise has strongly affected the resulting segmentation. On the opposite, if ARI is equal to 1 it means that the adversarial perturbation is not suitable for fooling the segmentation.
A key property of the ARI measure is that it is not sensitive to labeling alteration, meaning that $ARI([0,0,1,1],[1,1,0,0]) = 1$. This is a strong advantage since we are interested on splitting the image in two clusters/regions, without taking in consideration the resulting labeling. Indeed if we use another measure like the $F-$measure we obtain $F1([0,0,1,1],[1,1,0,0]) = 0$, even if the separation results to be correct.
ARI_h is instead defined equally to the ARI measure but it considers as true labeling Y the predicted clustering $\hat{Y}$ obtained by the three algorithms in absence of adversarial noise. This new measure has been introduced for studying how the internal robustness change in presence of adversarial noise. In other words, how decrease the performance of the algorithm with respect to the initial prediction in presence of adversarial perturbations.

Coming back to Fig. 4.7 and Fig. 4.8 we can see that the Dominant Sets algorithm works better than the others in absence of adversarial noise. Then if we increase Δ the three clustering algorithms get worse, only K-Means is quite constant at the beginning. If we compare these results with Fig. 4.8 we can notice that with respect to the initial prediction K-Means is more robust than the others, with the disadvantage of obtaining lower ARI in absence of adversarial noise.

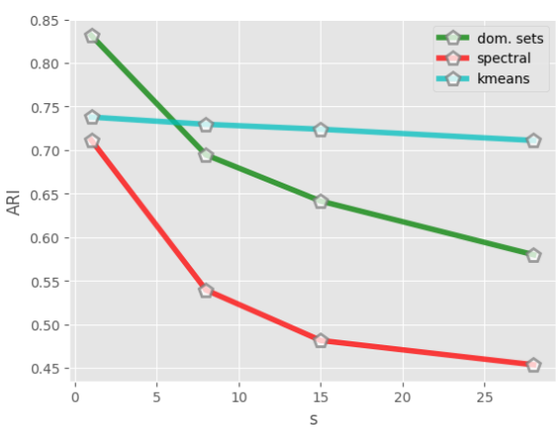


Figure 4.9: ARI over num. of perturbed rows s.

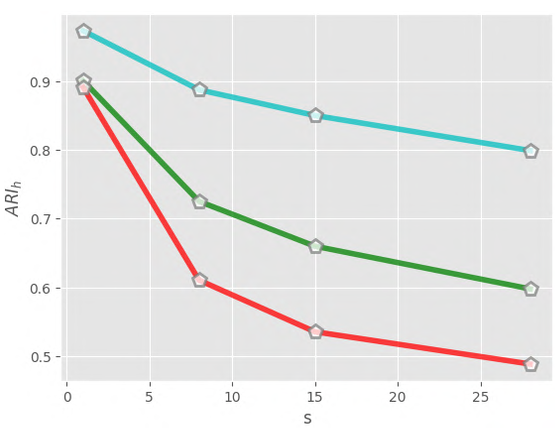


Figure 4.10: ARI_h over num. of perturbed rows s.

Another parameter used by the adversarial algorithm is the number of rows s. The two figures 4.8 and 4.10 highlight how the three algorithms react increasing this parameter. We can notice that K-Means is not strongly affected by this algorithm, the unique factor of interest result to be Δ. Indeed we can manipulate multiple rows, but if we use Δ too small then it will be ineffective, since we are using a similarity matrix based on distanced between pixel intensities.
In the opposite spectral results to be more sensitive on the number of perturbed rows. It means that even small values of Δ can affect the results of the final segmentation if the consider an opportune s.

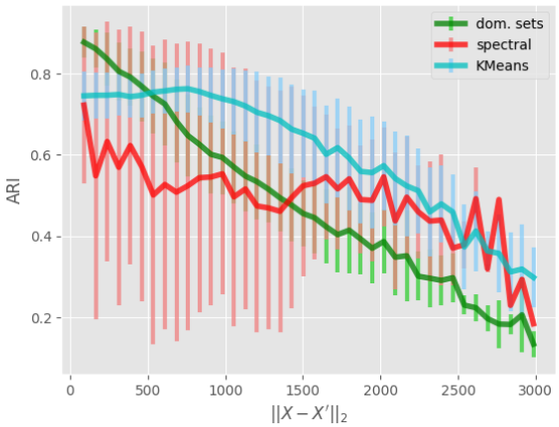


Figure 4.11: Errorplot ARI over the attacker's capacity.

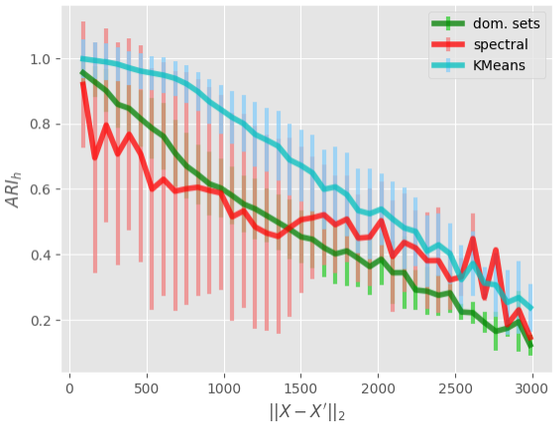


Figure 4.12: Errorplot ARI_h over the attacker's capacity.

Another way of reading the attacker's capacity, as suggested in [4], is to evaluate the distance between the original input X and the adversarial one X'. The euclidean norm $||X - X'||_2$ gives a measure of distance between the two entities. In particular, the greater is the distance the stronger is the attacker. From this analysis two plots have been designed and shown in Fig. 4.11 and Fig. 4.12. In the x-axis we consider the euclidean norm $||X - X'||_2$ and in the y-axis we consider again ARI and ARI_h. For the sake of readability, we report the average ARI and ARI_h with the corresponding standard deviation (shown with error bars). The first impression that comes from these two plots is that Spectral clustering has a very large standard deviation, meaning that it works well for some digits but does not provide good results on others. The other two clustering algorithms, Dominant Sets and K-Means , instead preserve a smaller variance. These two plots resume the analysis previously reported, that is, in the natural setting Dominant Sets works better with respect to the ground truth. Instead, with the increase of the attacker's capacity Dominant Sets is not able to maintain the clustering performances more than the other two algorithms.

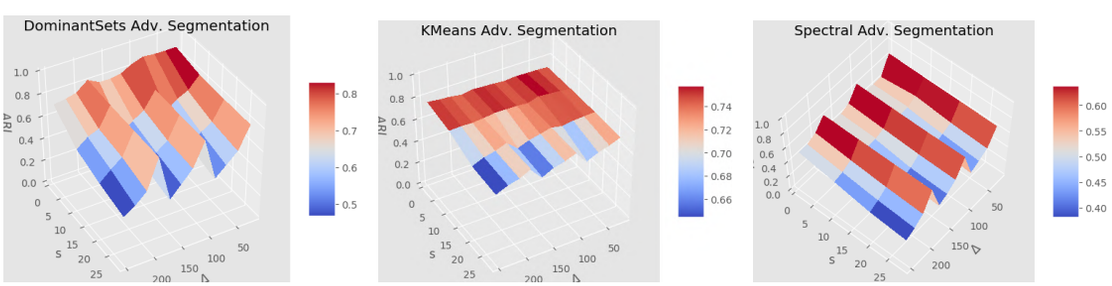


Figure 4.13: ARI over Δ and num. of perturbed rows s.

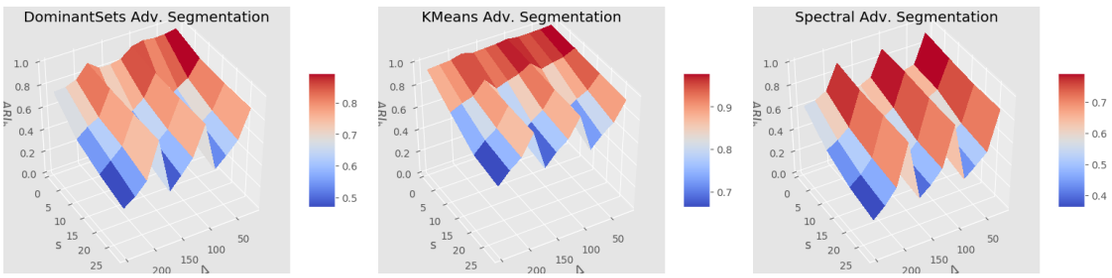


Figure 4.14: ARI_h over Δ and num. of perturbed rows s.

Figures 4.13 and 4.14 shows the 3D surface, obtained with linear interpolation, generated by changing Δ and s. In all the three algorithms the minimum ARI and ARI_h are reached when we use high Δ and s, meaning that also the resulting capacity $||X - X'||_2$ ie higher. In other words, this 3D representation confirm what we said previously for Fig. 4.11 and 4.12, meaning that the Euclidean norm is a good proxy for collapsing the two parameters (Δ, s) into a unique single measure.

Note that these analyses have been done considering only the MNIST dataset, but in the future works for sure we will extend this analysis with other datasets in order to check it these results are a general or they can be found also in other settings. Another key point is that in these experiments we used the distance between intensities for constructing the similarity matrix. Probably for that reason K-Means seems to be the strongest algorithm, but we don't have any guarantee about its robustness when other similarity measures are used.

4.1.4 Crafted Adversarial Examples

In this section we are going to show some adversarial examples crafted using Alg. 1 for the three clustering algorithm.

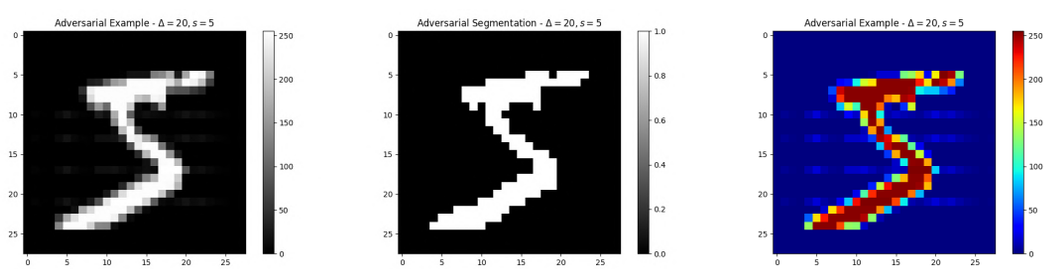


Figure 4.15: Adversarial example (left) crafted against Dominant Sets clustering with $\Delta = 20$ and $s = 5$. The resulting segmentation (middle) results to be quite good, insensible to the adversarial noise. The image on the right shows in another color scale the crafted adversarial example.

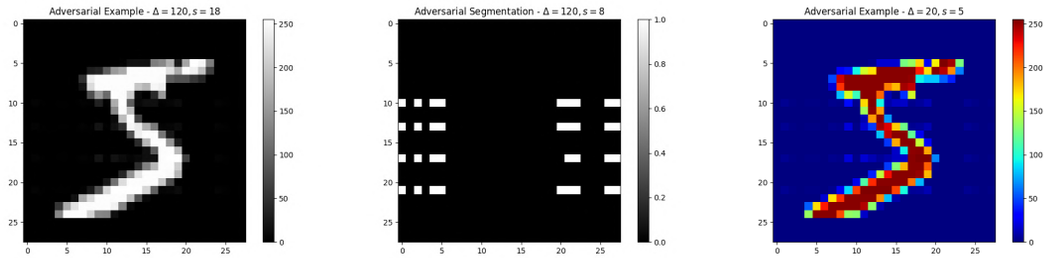


Figure 4.16: Adversarial example (left) crafted against Spectral clustering with $\Delta = 20$ and $s = 5$. The resulting segmentation (middle) results to be completely destroyed. A targeted and small perturbation has fooled completely the spectral segmentation for this example. The image on the right shows in another color scale the crafted adversarial example, an we can notice that the injected noise is close to zero.

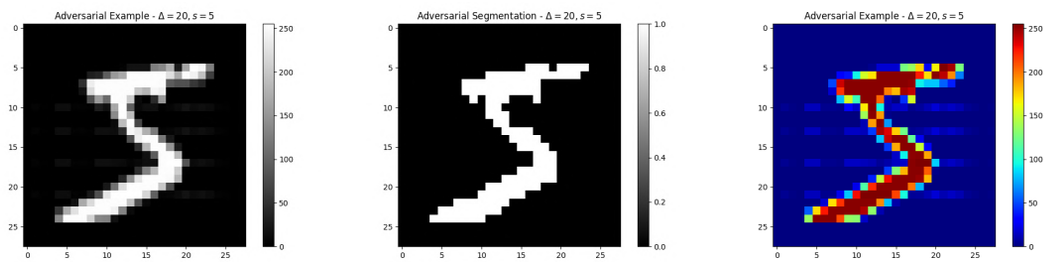


Figure 4.17: Adversarial Example (left) crafted against K-Means clustering with $\Delta = 20$ and $s = 5$. The resulting segmentation (middle) results to be quite good, insensible to the adversarial noise, since the maximum level of noise Δ is small. The adversarial noise injected is easier to see in the figure on the right, which basically projects the image using another color scale.

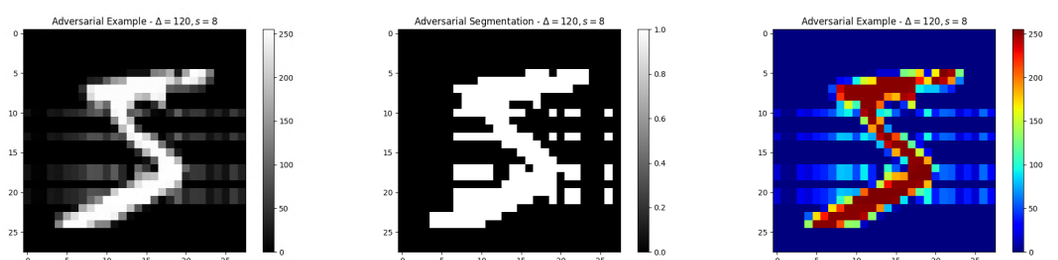


Figure 4.18: Adversarial example (left) crafted against Dominant Sets clustering with $\Delta = 120$ and $s = 8$. The resulting segmentation (middle) results to be strongly affected by noisy patterns. Even if for humans the only important thing is the digit 5, the noisy pattern is grouped together with the foreground. The image on the right highlights better how strong is the noise injected.

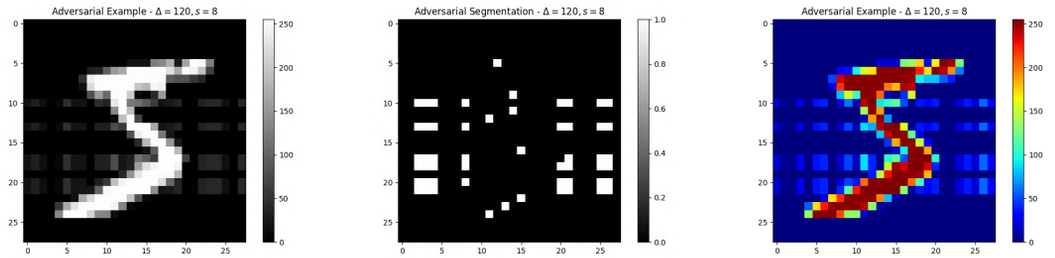


Figure 4.19: Adversarial example (left) crafted against Dominant Sets clustering with $\Delta = 120$ and $s = 8$. The resulting segmentation (middle) results to be completely destroyed by the noisy patterns. Even if the generated adversarial example is quite similar to the initial input image, the adversarial segmentation does not totally preserve the "5" digit shape. From the image on the right we can notice that the injected noise has a low value of intensity.

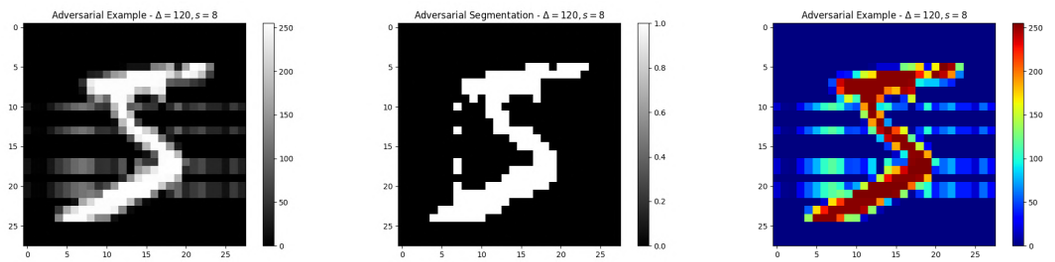


Figure 4.20: Adversarial example (left) crafted against K-Means clustering with $\Delta = 120$ and $s = 8$. The resulting segmentation (middle) results to be the best one with respect to the other algorithms. Only few points have been grouped inside the foreground, even if from the image on the right we can notice that the injected noise is very strong.

4.2 Pixelwise-Based Adversarial Algorithm

The next designed algorithm, described in Section 3, allows the attacker to inject noisy perturbations only in target pixels of X. It means that there is not a clear noisy pattern, but each target pixel is optimized independently. The choice of targets may be done considering opportune heuristics, designed taking advantage of the attacker's capacity. During this section we are going to analyze how targeted pixels are identified by our heuristics, how adversarial noise is optimized and we will analyze the results obtained for the MNIST dataset.

4.2.1 Choice of Target

The key point of this algorithm is that it is capable to detect the most s sensitive pixels that belong to the background and to the foreground. Then it optimizes for each of them an opportune adversarial noise. The choice of these most sensitive pixels is done by the two functions `inner_sensitive_pixels` and `outer_sensitive_pixels`.

The first function is designed for finding pixels belonging in the foreground that are sensitive to be moved towards the background cluster. Given an $n \times m$ input image X, partitioned with foreground pixels F and background pixels B, the function `inner_sensitive_pixels` returns a set T containing s indices corresponding to the pixels with the smallest intensity in F.

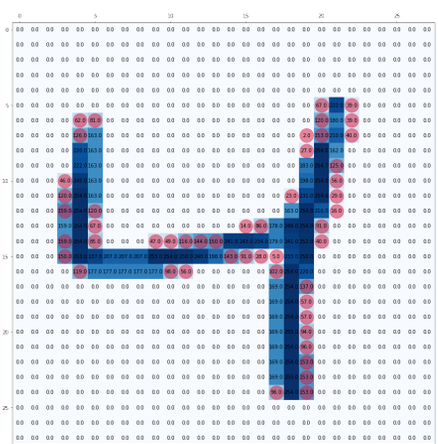


Figure 4.21: Top 50 inner sensible pixel for digit 4 in MNIST.

Pixels highlighted in red in Fig. 4.21 are the ones with the smallest intensity in the foreground (blue). These pixels define the target pixels in T that are sensitive to be perturbed. The goal of the adversarial algorithm is to move them towards the background cluster.

The second function `outer_sensitive_pixels` returns a set of s pixels in the background that satisfies a certain criteria or heuristic. In our experiments we decided to consider the locality of pixels for choosing the most sensitive pixels, with the assumption that the attacker wants to hide threats as much as possible for fooling also human eyes. The `outer_sensitive_pixels` can be briefly described in the following way:

- Given an $n \times m$ input image X with the associating ground truth segmentation Y.

$$Y_i = \begin{cases} 1, & X_i \in F \\ 0, & \text{otherwise} \end{cases}$$

- Craft an outer mask Y^d applying a *dilation* of Y using kernel size e.
- Get border mask Y^b by subtracting to Y^d the ground truth Y ($Y^b = Y^d - Y$).
- $\sigma(Y_b) = \{i \in \{1, \ldots, n \times m\} : Y_i^b > 0\}$
- Pick s random pixels in $\sigma(Y_b)$ as targets.

The dilating function is a morphological operator especially used in computer vision for different applications: removing noise, find holes or isolate individual elements. It consists in convolving an input image X with a square kernel E of size e. Then for each convolution, centered in pixel i, compute the maximal intensity value v inside the convolution window and sets intensity of pixel i to v. The kernel size e defines how much we want to dilate the image X.

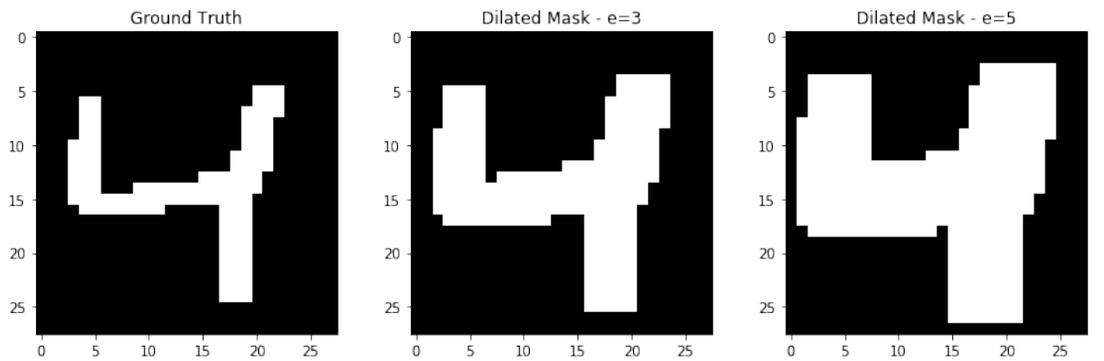


Figure 4.22: Ground truth segmentation on the left and the corresponding results obtained by dilate function. In the middle the kernel size $e = 3$. The figure in the right shows the result using kernel size $e = 5$. The greater is the kernel size the bigger will be the white shape.

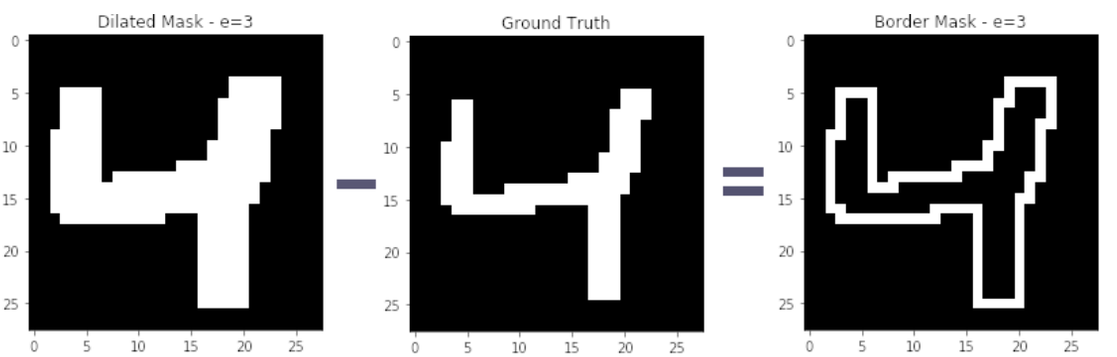


Figure 4.23: The border mask, on the right, Y^b obtained by subtracting from the dilated mask Y^d, on the middle, the ground truth segmentation Y. Kernel size $e = 3$.

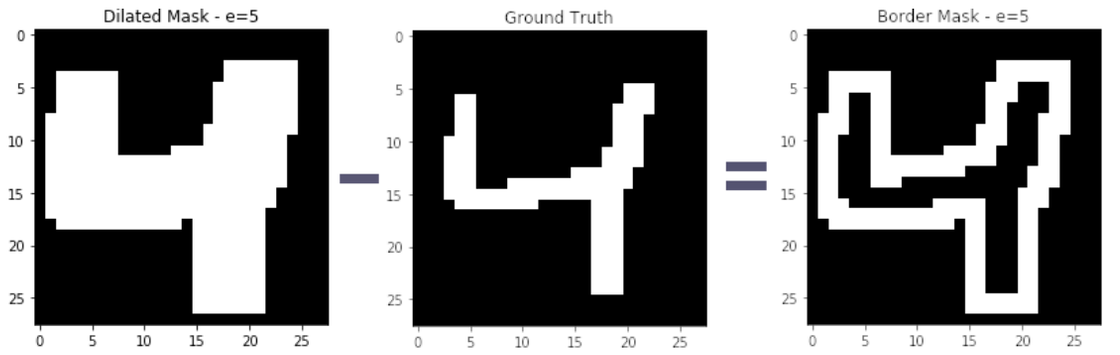


Figure 4.24: The border mask, on the right, Y^b obtained by subtracting from the dilated mask Y^d, on the middle, the ground truth segmentation Y. Kernel size $e = 5$.

Sensitive pixels are randomly extracted from the border mask. Note that the smaller the kernel size e is, the closer sensitive pixels are to the border. We may see this strategy as a way for hiding threats when e is small, since otherwise threats in pixels too far from the foreground are easier to be detected even by human eyes since they are strongly isolated.

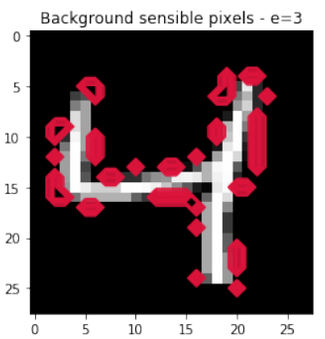


Figure 4.25: Outer target pixels with kernel size $e = 3$.

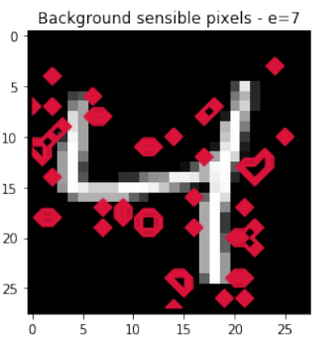


Figure 4.26: Outer target pixels with kernel size $e = 7$.

Note the differences between the two previous images. In Fig. 4.25 target pixels, highlighted in red, are very close to the border of the foreground. On the other hand, targeted pixels in Fig. 4.26 results to be farther from the foreground, meaning that strong perturbations may be more easily identified with respect to the ones closer to the foreground border.
Coming back to Alg. 3 remember that we can perturb and optimize each target pixel independently or jointly with others p consecutive.

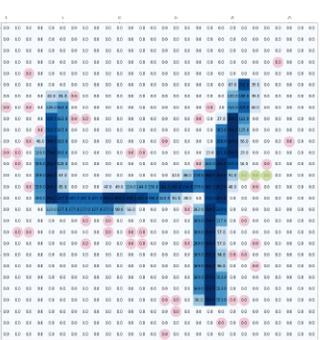


Figure 4.27: Outer sensible pixels for digit 4.

Fig. 4.27 shows in red targeted pixels that belong to the background. Consider the case in which $p = 3$, then the optimization will consider 3 consecutive pixels (green points) at a time generating a unique ε^* for them.

4.2.2 Local and Global optimization

In Section 4.1.2 we discussed the role of the `global` parameter and its implications on the resulting adversarial example. In the pixelwise algorithm it basically assumes the same role. Considering the example provided in Fig. 4.27, if `global=FALSE` then the optimizer performs the clustering evaluation only on the 3 consecutive columns where the 3 green pixels belong. Otherwise, with `global=TRUE`, the optimal adversarial noise is detected by injecting noise on the 3 green pixels and then the corresponding segmentation is evaluated on the entire image.

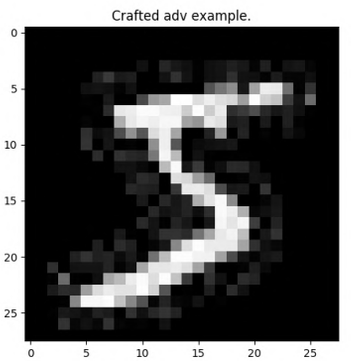


Figure 4.28: Local Optimization Adversarial Example.

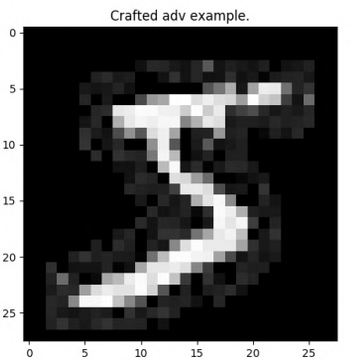


Figure 4.29: Global Optimization Adversarial Example.

Fig. 4.28 and 4.29 provide the visualization of two adversarial examples crafted against Dominant Sets with parameters $\Delta = 120, s = 200, e = 5$. The first on the left was crafted considering local optimization, while the second one was crafted using global optimization. The two seem to be very similar, differences can only be noticed in few pixels.

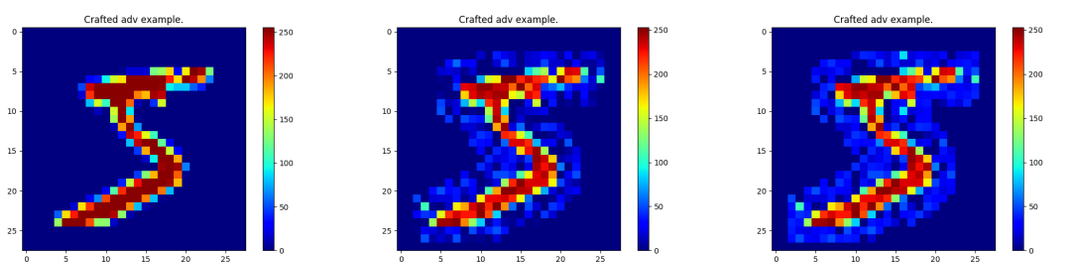


Figure 4.30: Visualization of the original input image X (left), adversarial example with local optimization (middle) and with global optimization (right).

By the color scale we can notice that the two adversarial examples are quite similar. The similarity between these two results comes from the observation that targeted pixels are strongly close to the foreground shape, meaning that optimizing locally

or respect to the entire image is more or less similar. In other words, while the global optimization considers all the pixels intensities, the local one considers only pixel intensities in a certain portion. If that portion has pixels strongly similar to others in the entire image then the two optimization strategies will look for similar solutions.

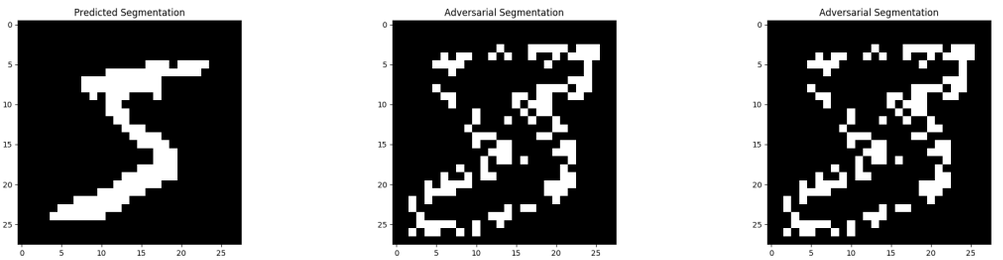


Figure 4.31: Clustering segmentation obtained for the input X (left), adversarial example with local optimization (middle) and with global optimization (right).

Equally the results that we obtain performing the clustering segmentation are similar. Both the two adversarial examples are able to break the clustering algorithm, obtaining a segmentation very far from the original predicted one. This observation gives more emphasis on the similarity between the two optimization strategies. In the optimization strategy a strong role is played also by the dilation kernel size e, since it defines how much targeted outer pixels should be close to the foreground. The closer the outer mask is to the foreground shape, the lower the effect of moving from global optimization to the local one is. Note what happens when we increase the dilation kernel size from $e = 5$ to $e = 7$.

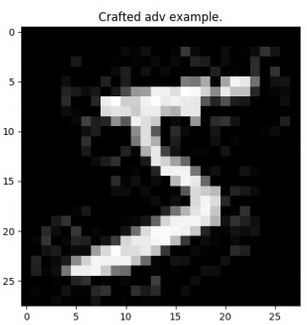


Figure 4.32: Local Optimization Adversarial Example.

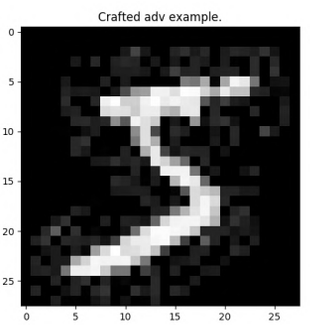


Figure 4.33: Global Optimization Adversarial Example.

Differences between the two adversarial examples are more evident with respect to the previous case. Indeed, we can notice that the perturbations injected for the first one in Fig. 4.32 are less evident than the ones shown in Fig. 4.33.
In conclusion the advantages obtained using the local optimization in the opposite of the global optimization can be shortly summarized:

- With small values of e we obtain similar adversarial examples crafted with global optimization but with a consistently lower computational time.
- With high values of e the outer perturbations are more sparse and the local optimization tries to hide threats better than the global one.

4.2.3 MNIST results

In this section we are going to analyze the robustness provided by the three clustering algorithms in presence of adversarial pixelwise noise. Dataset and measures used are exactly equal to the ones described in Section 4.1.3.
The first analysis takes into consideration the maximum power noise Δ. The two following plots show how the three algorithms react when Δ increase regardless of the other parameters.

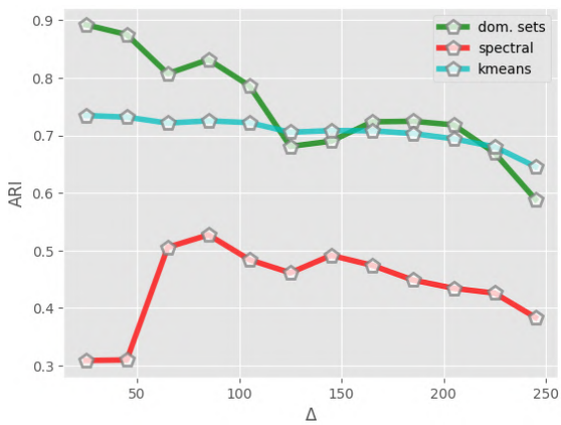


Figure 4.34: *ARI* over power noise Δ.

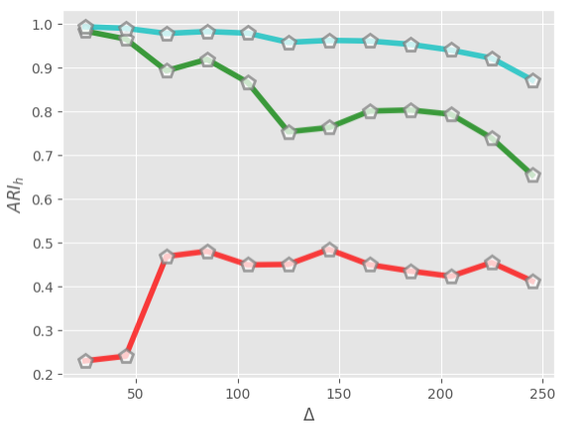


Figure 4.35: ARI_h over power noise Δ.

We can immediately confirm the same pattern discussed in Section 4.1.3, that is, Dominant Sets clustering works better than the others in the natural setting, but in a certain point, when Δ becomes bigger, it is not capable to maintain this quality anymore. Indeed, we can notice from 4.35 that the more robust algorithm seems to be K-Means, but looking at Fig. 4.34 we can notice that it does not work well with respect to the ground truth in absence of adversarial noise. Something strange happens with Spectral clustering, in fact it seems that very small perturbations strongly affect the results of the segmentation. From this observation we decided to evaluate how the three algorithms react by changing the number of internal and external pixels to attack.

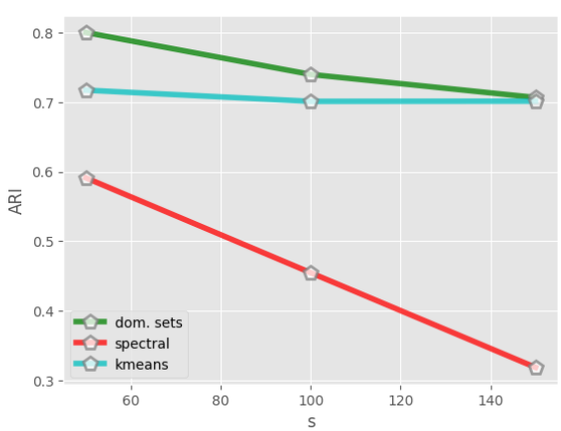


Figure 4.36: *ARI* over num. of perturbed pixels s.

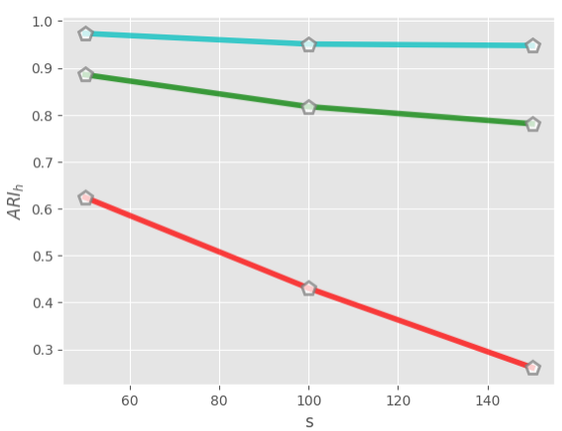


Figure 4.37: ARI_h over num. of perturbed pixels s.

Fig. 4.35 and Fig. 4.37 show that the most sensitive algorithm for this parameter is Spectral, meaning that, regardless of the maximum power noise Δ, by picking an accurate number of pixels to perturb the algorithm can be strongly fooled. On the other hand, the other two clustering algorithms requires greater values of Δ for changing the initial prediction.

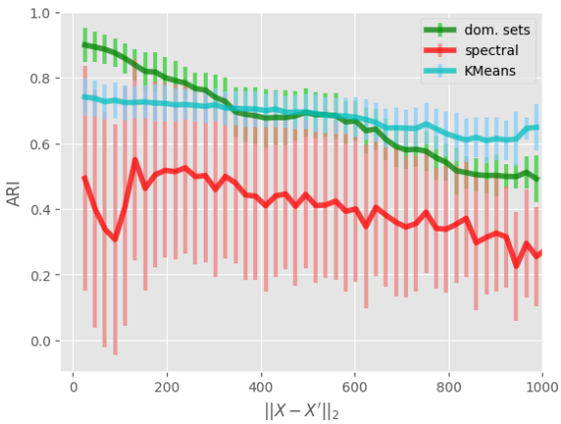


Figure 4.38: Errorplot ARI over the attacker's capacity.

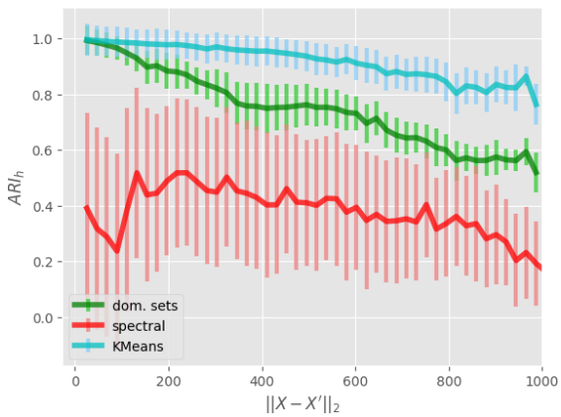


Figure 4.39: Errorplot ARI_h over the attacker's capacity.

We decided to analyze the robustness trend of the three algorithms considering the attacker's capacity, as suggested by Biggio et al., expressed in form $||X - X'||_2$. This representation takes into consideration both Δ and s at the same time, since the greater are these two parameters the greater is the Euclidean distance between the original sample X and the adversarial one X'. We can immediately notice the large standard deviation provided by the spectral clustering algorithm, that highlights how it works quite good for some examples in MNIST and for others it provides bad results.
The two more robust clustering algorithms seem to be K-Means and Dominant Sets, with greater performance of the last one in absence of adversarial noise or for small perturbations. K-Means again results to be more robust than the others, but this is probably related to the fact that we are using similarities based on the distances. In future works, it would be interesting to expand this analysis by introducing other similarity measures.

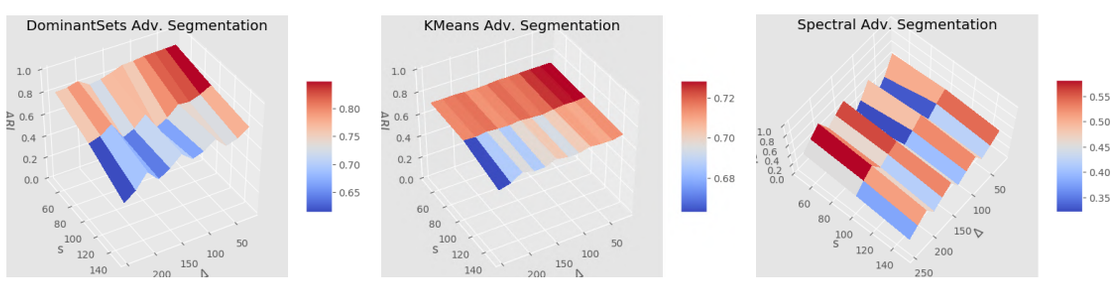


Figure 4.40: ARI over Δ and num. of perturbed pixels `s`.

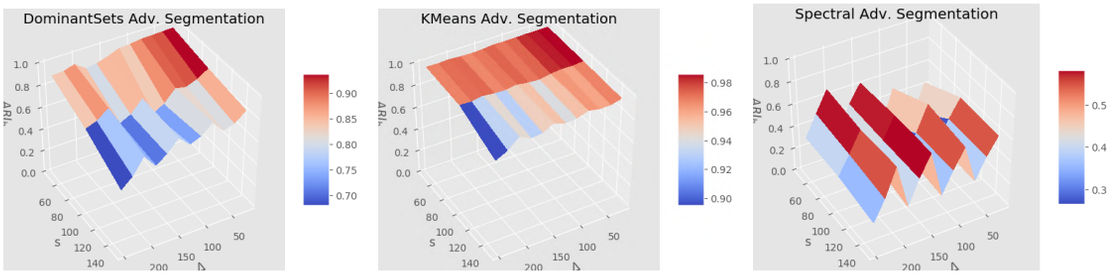


Figure 4.41: ARI_h over Δ and num. of perturbed pixels s.

Figures 4.40 and 4.41 shows the 3D surface, obtained with linear interpolation, generated by changing Δ and s. In both of the Dominant Sets and K-Means algorithm the minimum ARI and ARI_h is reached when we use high Δ and s, meaning that also the resulting capacity $||X - X'||_2$ is higher. On the other hand Spectral reaches the minimum in presence of small perturbations, and even from this representation we can notice a high variance provided by the Spectral algorithm results.
In other words, this 3D representation confirms what we said previously for Fig. 4.38 and 4.39, meaning that the Euclidean norm is a good proxy for collapsing the two parameters (Δ, s) into a unique single measure.

4.2.4 Crafted Adversarial Examples

In this section we are going to show some adversarial examples crafted against the three clustering algorithms:

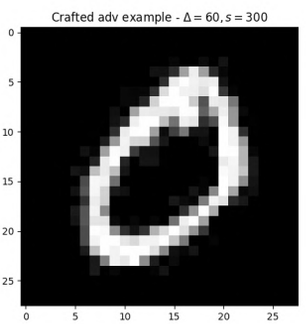


Figure 4.42: Sample digit "0" X to attack.

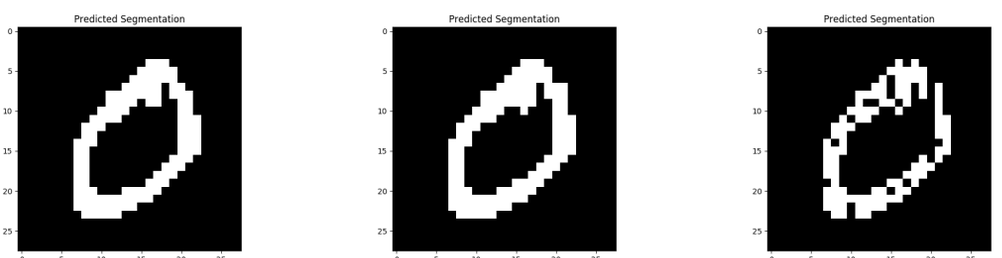


Figure 4.43: Clustering segmentation obtained by the three algorithms, correspondingly Dominant Sets, Spectral and K-Means, for digit "0".

We can immediately notice how in absence of adversarial noise K-Means and Dominant Sets work well, keeping correctly the foreground. On the other hand, Spectral leaves some holes inside the foreground, reducing the quality of the final result.

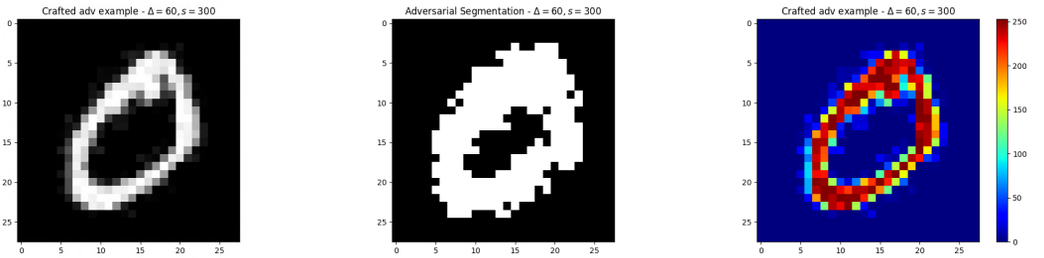


Figure 4.44: Adversarial Example (left) crafted against Dominant Sets clustering with $\Delta = 60$, $s = 300$ and $e = 3$. It is strongly similar to the original input image. The resulting segmentation (middle) seems to preserve the digit 0 shape but even adversarial noise has been clustered in the foreground. The image on the right shows in another color scale the crafted adversarial example, and we can notice where the noise is located.

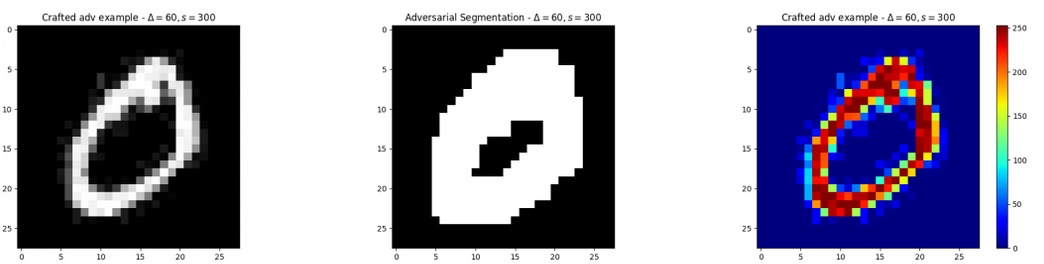


Figure 4.45: Adversarial Example (left) crafted against Spectral clustering with $\Delta = 60$, $s = 300$ and $e = 3$. The results are similar to the ones obtained for Dominant Sets but also for noisy pixels are grouped in the foreground.

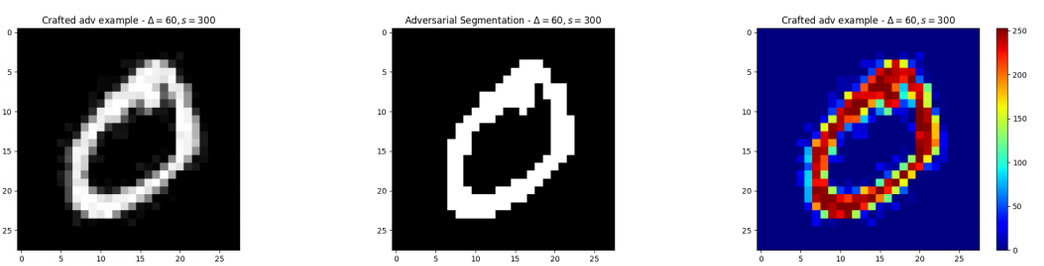


Figure 4.46: Adversarial Example (left) crafted against K-Means clustering with $\Delta = 60$, $s = 300$ and $e = 3$. The segmentation obtained against the adversarial example (middle) is quite good, meaning that the injected attacks are not able to fool K-Means since Δ is too small. The figure on the right shows, with a different color scale, where adversarial perturbations are located and how strong they are.

4.3 Targeted Clustering Adversarial Algorithm

Clustering algorithms are used in different applications, due to the unavailability of having labeled data in certain settings. Different applications of clustering algorithms are widely used and studied for analyzing data both in the literature and in the industry, for example:

- Crime analysis, used for predicting patterns and trends in crime. Given a location of interest we can use clustering algorithms for analyzing crime prone areas.
- Call Detail Record (CDR), used for studying customers. Clustering algorithms are used for grouping customers with similar needs and then specific or custom offers are proposed to the different groups.
- Cyber profiling, used for studying activities or Internet users through log activity analysis. Users could be clustered according to similar behaviors analyzing daily activities, and then a summarized profile of them can be reconstructed. An interesting application of this setting is used for identifying possible malicious users.
- Local classification, instead of training a classifier over all the data in some application it seems to be better to split data into groups and then train a model for each group. For example, in patient analysis we could group patients with similar characteristics and then use custom classifiers for predicting the risk of having dangerous diseases.

All of these applications adopt clustering algorithms for their purposes but they have to deal with an unexpected problem: clustering algorithms have not been originally designed to work in adversarial settings. This may allow the attacker to inject carefully-crafted attacks into collected samples in order to compromise the clustering results.
In section 3.3 we propose an algorithm for crafting targeted adversarial examples for fooling data clustering algorithms. The goal of the attacker is to obtain a certain clustering labeling for some samples, targeted or generic. In our application we decided to leave the algorithm the role of choosing the most sensitive samples to move towards a desirable cluster. Then, we tested the clustering robustness over multiple datasets considering always the same similarity measure, based on distance between entities.

4.3.1 Choice of Target

The first step of the algorithm, as highlighted in Alg. 5, is to pick the most sensitive points or entities to attack. Nothing excludes the possibility for the attacker to decide by himself/herself target entities. In our experiments we decided to leave the algorithm the role of picking the most sensitive entities through the application of a certain heuristic. In particular, given two clusters C_1 and C_2 we assume that the attacker wants to move samples from C_1 towards C_2. Then, the algorithm picks the closest samples to C_2 that belong to C_1. Distances from a point to a cluster can be defined in different ways:

- Single link (p, C_2): gets pairwise distances from p and all the samples in C_2 and the smallest one is considered.

- Complete link (p, C_2): gets pairwise distances from p and all the samples in C_2 and the smallest one is considered.

- Average link (p, C_2): gets the centroid c_2 of cluster C_2 (average point between samples in C_2) and then distance between c_2 and p is considered.

In our experiments we decided to use the last one, so at each iteration the adversarial algorithm looks for optimal noisy perturbations such that target samples are moved towards the centroid of a certain target cluster.
For the sake of clarity we are going to propose an example of how the adversarial algorithm works on a synthetic dataset X created using the `make_blobs` function of the `sklearn` library[1]. X is organized in two evident blobs, each of which contains 250 samples.

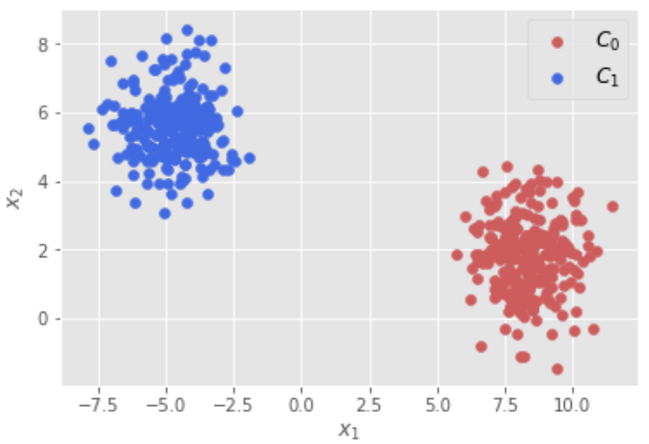


Figure 4.47: Synthetic dataset X.

From Fig.4.47 we can notice that the two clusters are strongly cohesive and separate, in fact the three algorithms work very well on detecting the two structures.
The first step of the adversarial algorithm is to detect sensitive targets for moving points from C_1 to C_2. For that reason the s closest points to c_0 (centroid of C_0) belonging to C_1 are considered.

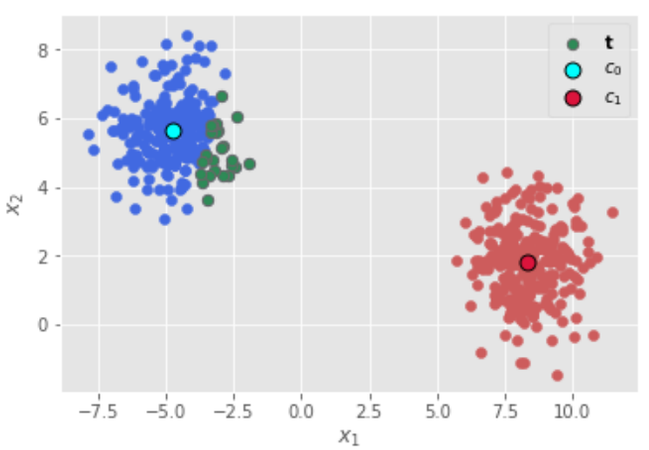


Figure 4.48: Target sensitive points $s = 25$ and cluster centroids.

[1]Sklearn Library - make_blobs function references

Fig. 4.49 provides a graphical visualization of the scenario. Note that the green points $\mathbf{t}$ are the ones that the algorithm tries to move. Of course we could take into consideration other heuristics, indeed another heuristic could be to take the farthest points instead of the closest one.
The last part of the adversarial optimization consists in finding the best perturbation to be applied to green points $\mathbf{t}$.

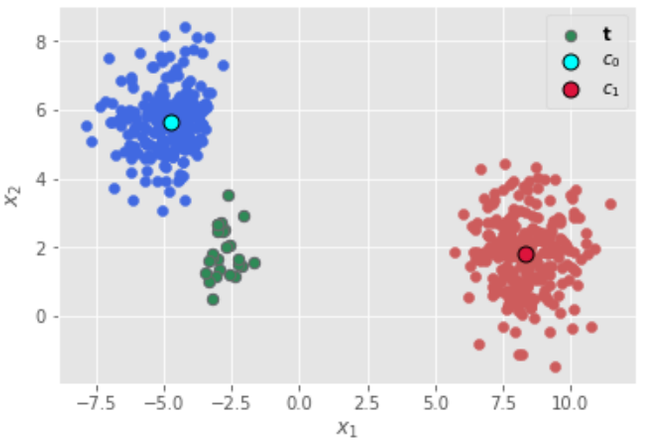


Figure 4.49: Adversarial synthetic dataset, $\Delta = 2, s = 25$.

As expected the adversarial algorithm moved 25 points (0.1% of C_0) towards C_1, simply reducing the x_2 and increasing the x_1 features.

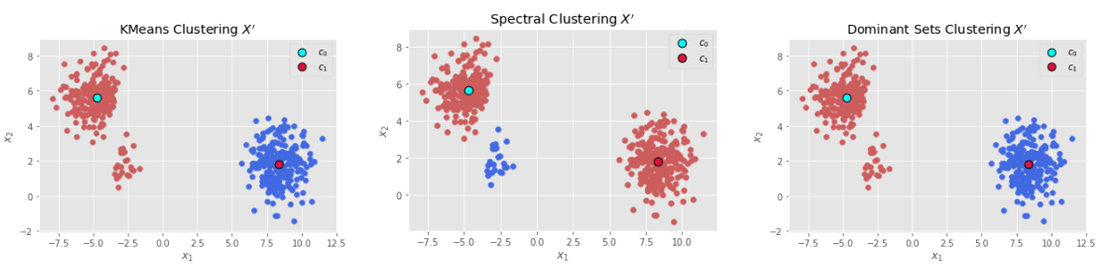


Figure 4.50: Data clustering obtained for X' using K-Means (left), Spectral (middle) and Dominant Sets (right) clustering.

From Fig. 4.50 we can observe two interesting things:

- Spectral clustering seems to be extremely sensitive to small perturbations.
- This example stresses again the importance of the introduction of the *ARI* measure. We can notice that, in the opposite on what shown in 4.49, the clustering labeling proposed by Dominant Sets and K-Means are reversed. The separation obtained is basically the same, the only difference is the cluster name, which is meaningless for our application. Indeed the *ARI* measure is not affected by this change and provides and quality equal to 1.

An interesting observation for Spectral clustering regards the role of k, which is the number of extracted clusters. Let k be the initial number of clusters retrieved over X. Now we define X' as an adversarial perturbation of X. Then, it seems

that iterating the Spectral clustering algorithm over X' forcing it to find exactly k clusters does not provide satisfying results. Conversely, if we execute it in order to find $k+1$ clusters, it seems to be more efficient and able to detect the adversarial points in a separate cluster. An example is shown below:

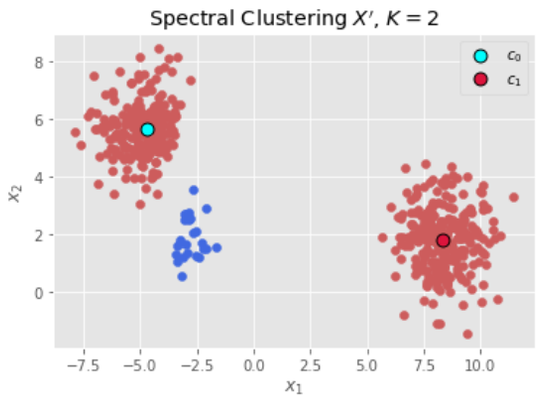


Figure 4.51: Spectral clustering for $K = 2$.

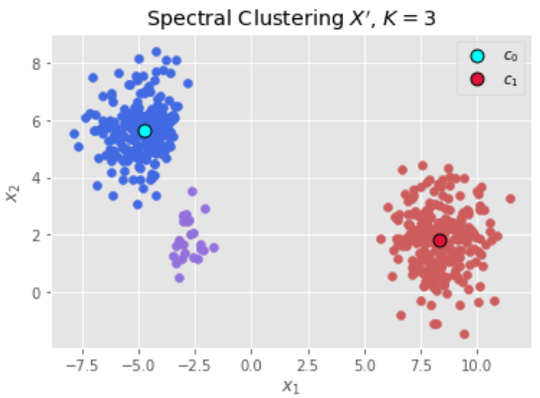


Figure 4.52: Spectral clustering for $K = 3$.

We can notice how the clustering obtained considering $K = 2$ over X' returns a strange labeling. The two blobs in Fig. 4.51 are clustered together (red) and target points are seen as a separate cluster (blue). On the other hand, if we consider one more cluster the Spectral clustering algorithm, shown in Fig. 4.52, is able to group adversarial samples in a unique cluster.

4.3.2 Local and Global optimization

In previous sections we analyzed the implications of using local or global optimization through the usage of the `global` parameter. Given $X \in \mathbb{R}^{n \times m}$, with $G = 20$ and $p = 2$, then the global optimization performs the clustering evaluation G times over $X' \in \mathbb{R}^{n \times m}$. Conversely, with the local optimization the adversarial algorithm iterates clustering evaluation G times over $X' \in \mathbb{R}^{n \times p}$, with $p \ll m$.
The global optimization may result to be computationally heavy to execute for certain datasets. For instance, in computer vision applications it is quite common to analyze large images, meaning that m is strongly high. Let's consider the MNIST dataset, containing images of 28×28 pixels. Then, we have that $m = 28 \times 28 = 784$ features to optimize and over with we need to execute the clustering algorithm G times. We can conclude that simply considering a small dataset like MNIST the computational advantage of using local optimization is extremely high.

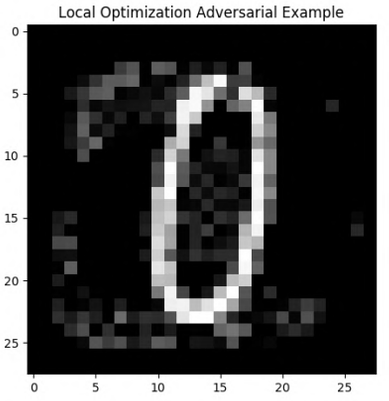


Figure 4.53: Crafted adversarial example against K-Means with local optimization $\Delta = 120, s = 5, p = 1$.

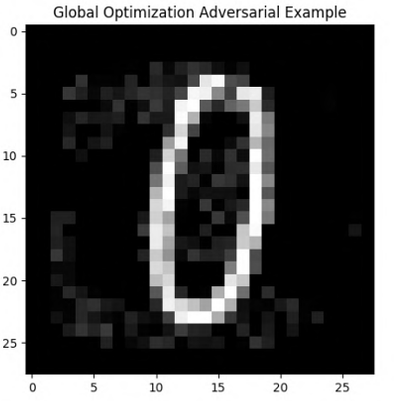


Figure 4.54: Crafted adversarial example against K-Means with global optimization $\Delta = 120, s = 5, p = 1$.

The two figures above show two adversarial examples crafted with the goal of moving a digit "0" towards the cluster of digits "3". The two optimization strategies were adopted and the results can be compared. In both cases we can notice that the algorithm has injected noise for highlighting the shape of a "3" digit. Even using the local optimization the resulting sample seems to be very good, it preserves the shape of a digit "0" and we can also notice that the injected noise around the foreground is close to a "3" shape. A way for analyzing the correctness of the algorithm is to project the noise mask obtained.

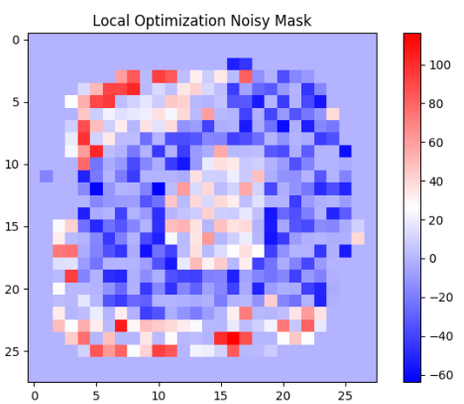


Figure 4.55: Crafted adversarial noisy mask against K-Means with local optimization $\Delta = 120, s = 5, p = 1$.

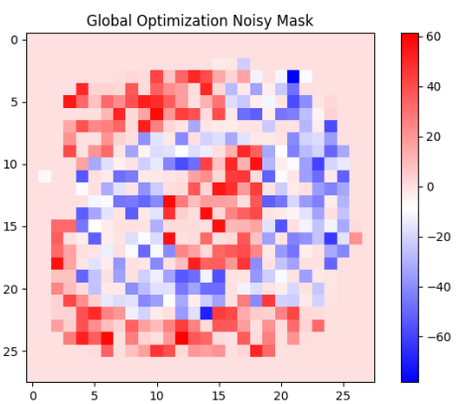


Figure 4.56: Crafted adversarial noisy mask against K-Means with global optimization $\Delta = 120, s = 5, p = 1$.

Comparing the two figures we can notice how the local one gives more importance to remove the shape "0", on the other hand the global one gives more emphasis on injecting the "3" shape.

4.3.3 Synthetic Dataset

The first experiment for evaluating the robustness of the clustering algorithms considers a 2 dimensional synthetic dataset obtained using the `make_blobs`[2] function, which basically generates isotropic Gaussian blobs. We decided to use the same dataset shown in Fig. 4.47, composed by 2 evident clusters of 250 samples each. The three algorithms in absence of adversarial noise provide an *ARI* accuracy equal to 1.0 using Gaussian similarity with Euclidean norm between points. Then the 3 algorithms (Dominant Sets, Spectral and K-Means clustering) are tested against multiple adversarial examples crafted using Alg. 5.

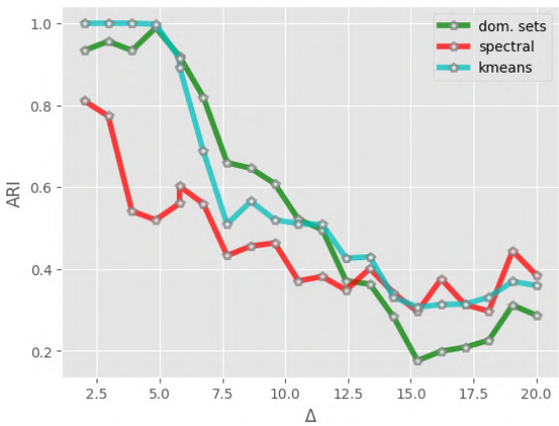


Figure 4.57: *ARI* over maximum noise power Δ.

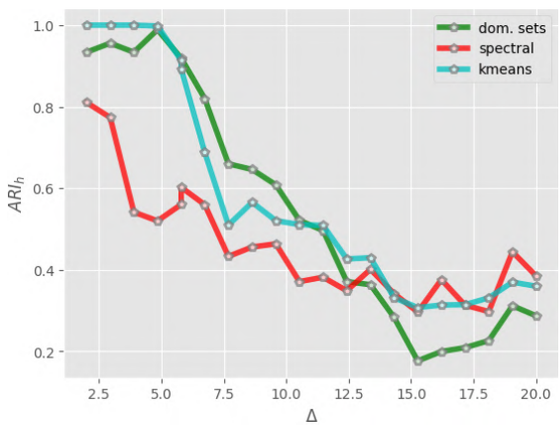


Figure 4.58: ARI_h over maximum noise power Δ.

The two figures above show that the closer adversarial samples are to the target cluster the worst the resulting clustering labeling is for all the three algorithms. As we have seen also in other experiments, Spectral seems to be most sensitive to small adversarial perturbations. On the other hand, K-Means and Dominant Sets clustering preserve a similar behavior. Note how the two plots look quite similar, since the true clustering and the initial one are exactly the same.

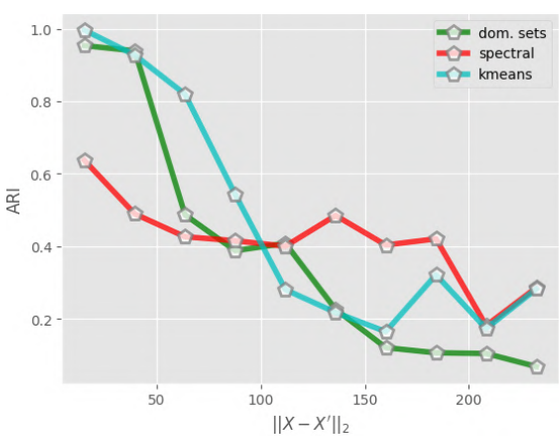


Figure 4.59: *ARI* over the attacker's capacity.

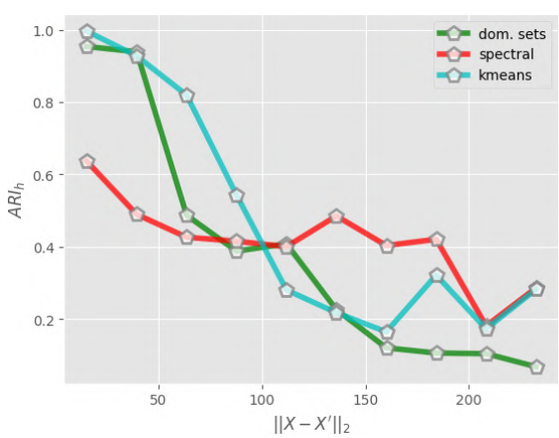


Figure 4.60: ARI_h over the attacker's capacity.

[2] `make_blobs` - Sklearn library

From Fig. 4.59 and 4.60 show the performance of the three algorithms by changing the attacker's capacity $||X - X'||_2$. The two plots highlight the sensibility of Spectral clustering in presence of very small perturbations, and the greater robustness provided by the other two algorithms.

4.3.4 DIGIT Dataset

The Digit dataset is similar to MNIST, since it contains images of hand-written digits, but with respect to the last one it contains 1797 8×8px images. Using this dataset, we decided to simulate a possible scenario in which the attacker wants to move samples from the "0" cluster to the "3" one.

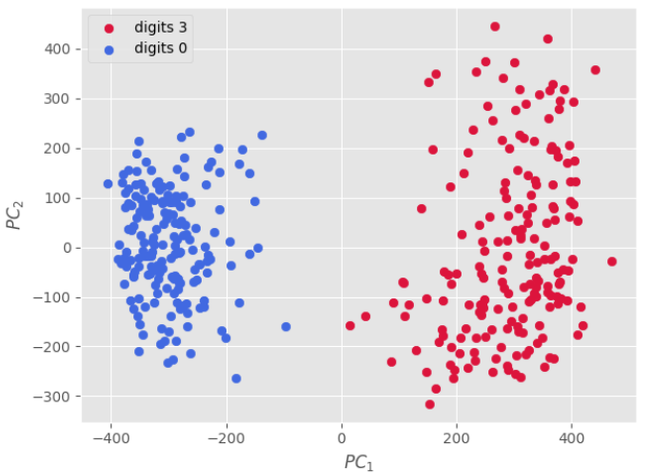


Figure 4.61: Principal component visualization from cluster "0" and "3".

From the principal component visualization, shown in Fig. 4.61, we can notice that the two clusters are very well separated. Considerably, even in the original space they result to be strongly separate, since the three clustering algorithms are able to reach an ARI measure equal to 1, meaning that all digits are correctly clustered. In the figures below we propose a visualization of two samples taken from clusters "0" and "3".

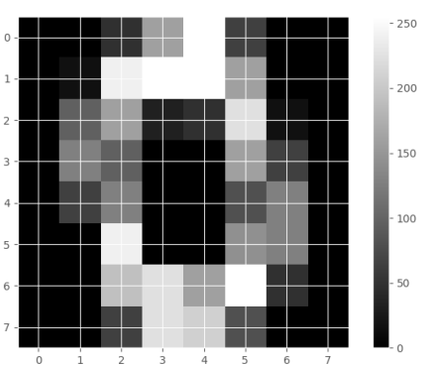


Figure 4.62: Digit "0".

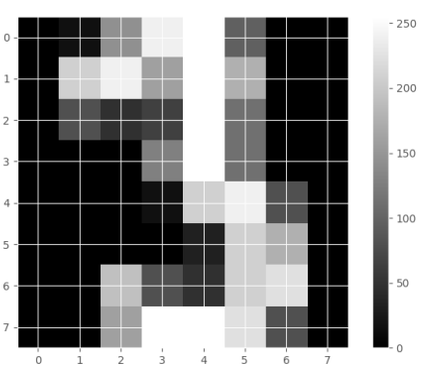


Figure 4.63: Digit "3".

In the following, we propose the results obtained by the three algorithms in presence of adversarial noise for the Digit dataset.

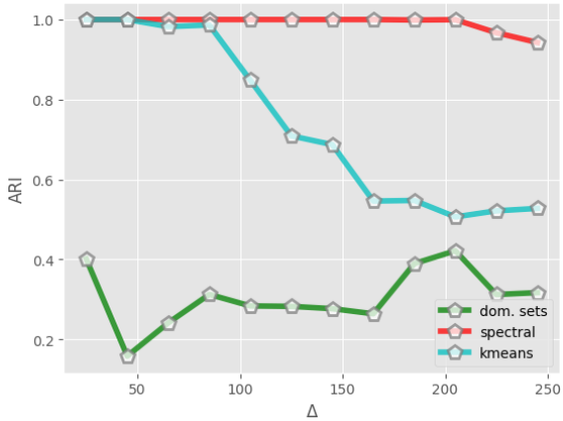


Figure 4.64: *ARI* over maximum noise power Δ.

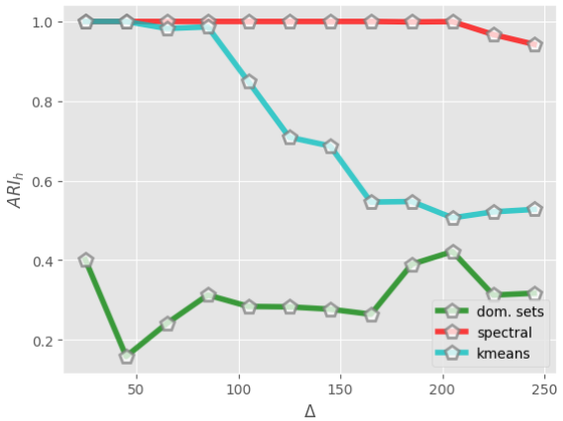


Figure 4.65: ARI_h over maximum noise power Δ.

Considerably, with respect to the previous analyses, Spectral clustering does not seem to be affected by the adversarial noise. The robustness provided by the Spectral clustering algorithm now is strongly greater than the ones provided by the other two algorithms. Conversely, Dominant Sets seems to be strongly influenced by the adversarial noise injected in DIGIT.

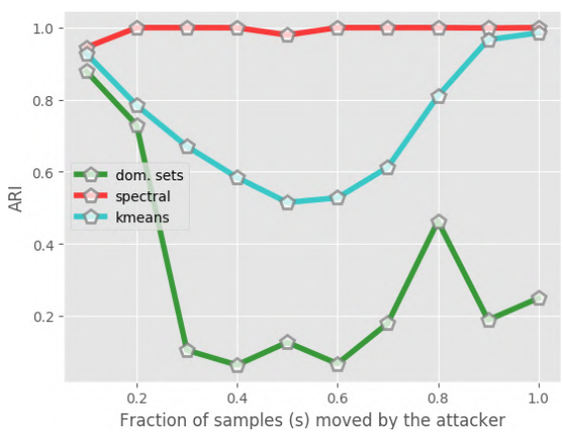


Figure 4.66: *ARI* over percentage of moved samples s.

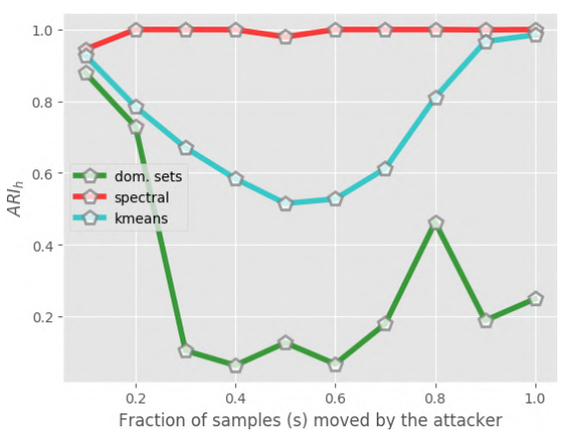


Figure 4.67: *ARI* over percentage of moved samples s.

The two plots provided in Fig. 4.66 and 4.67 highlight the importance of the number of samples moved from cluster "0" towards cluster "3". Indeed, we can notice that the major component that affects the results provided by Dominant Sets is exactly the parameter s. When only a small percentage of samples are moved, Dominant Sets is still able to detect the organization of the true clusters. But when the percentage of samples moved becomes too high then Dominant Sets is not able to maintain the original performance anymore. Remember that Dominant Sets is based on the extraction of dominant sets (maximal cliques for weighted graphs). The main reason for these results related to Dominant Sets is that when we move samples, we are basically reducing similarities between nodes inside the dominant set. In other words, given a dominant set $S \subseteq X$ we know by definition that $W_s(i) > 0 \ \forall i \in S$, importance of i in S. We also know that

$W_s(i)$ is computed considering the similarity between i and the rest of samples in S. Now let's define T the set of target samples in S that are sensitive to be moved. Then, when the adversarial algorithm moves samples T in adversarial way it is basically reducing the similarities between samples in T and samples in $S \setminus T$. Consequently it is reducing also the importance of sample in T to be part in S. For that reason only when we move few samples or all of them the Dominant Sets clustering algorithm is able to reconstruct the original cluster. The worst case is reached when we move more or less $40-60\%$ of samples from S, because basically the attacker is constructing a new cluster.
Another interesting pattern is provided by K-Means, in fact we can see that the performance decrease until we reach a certain point and then the greater s is, the better the results are. The reason is that when we move the entire cluster what we are done is basically to leave unchanged the entire structure. The worst case, if the attacker moves all the samples, is reached when the two clusters are completely merged.

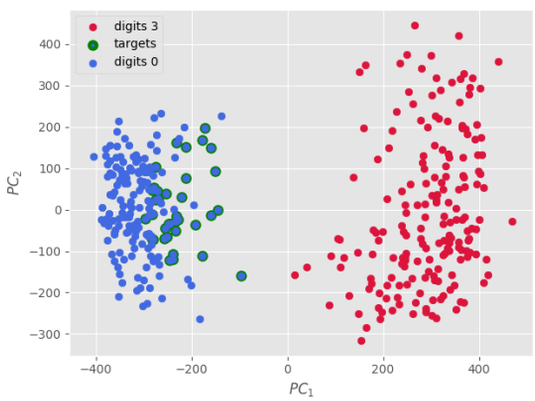


Figure 4.68: PC projection of "3" and "0" clusters, that define X. Green points are the targets ($s = 1\%$) of "0" sensitive to be moved towards "3".

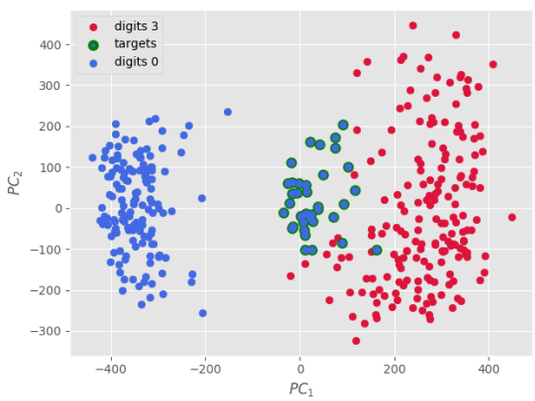


Figure 4.69: PC projection of X', obtained moving target samples ($s = 1\%$) by $\Delta = 180$ towards cluster "3".

Comparing Fig. 4.69 with Fig. 4.68 we see, through a principal component visualization, that target samples have been correctly moved towards the right direction.

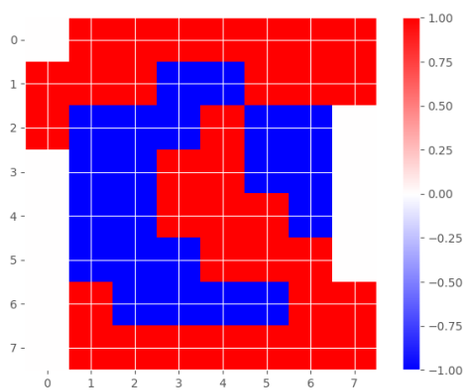


Figure 4.70: Direction mask from cluster "0" to cluster "3".

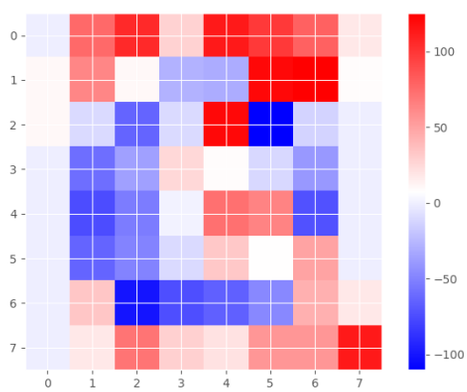


Figure 4.71: Optimal direction noise mask crafted against K-Means with $\Delta = 180$, $s = 1\%$ and local optimization.

Fig. 4.70 shows the right direction in order to transform a digit "0" into a "3". This mask represents in red the portions of the image for which intensities must be increased and in blue the portions which define regions where the algorithm needs to decrease intensities. We can notice that the red portion assumes the shape of a "3" digit and the blue one assumes the shape of a "0". The optimal noise mask extracted by the algorithm is shown in Fig. 4.71, which is quite similar to the direction mask. Applying this mask to a "0" sample we could convert it into a "3". In the following we proposed an example of its application.

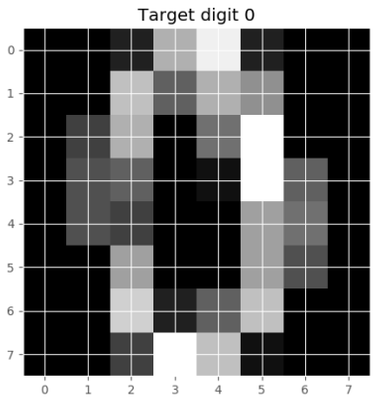


Figure 4.72: Target "0" digit.

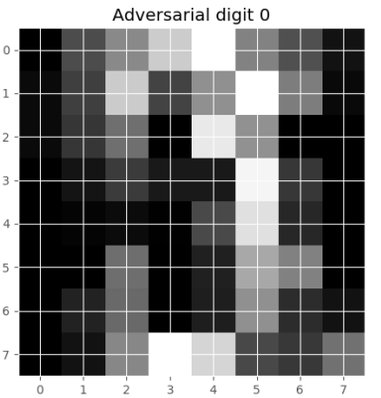


Figure 4.73: Adversarial "0" digit.

We can see from Fig. 4.73 how the original "0" digit, shown in Fig. 4.72, has been transformed for being more similar to a "3" digit.
In conclusion we report some plots that show how the 3 clustering algorithms react by changing the adversarial's capacity.

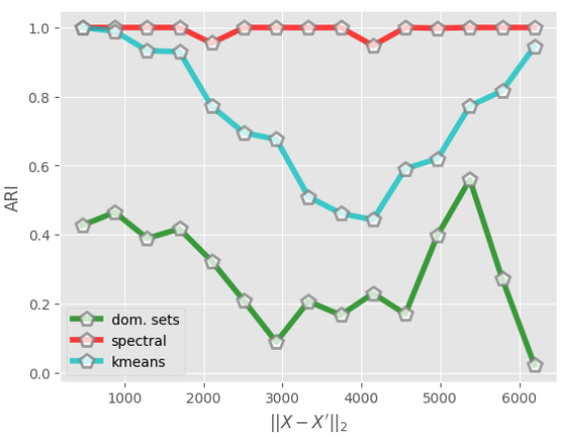


Figure 4.74: *ARI* over the attacker's capacity.

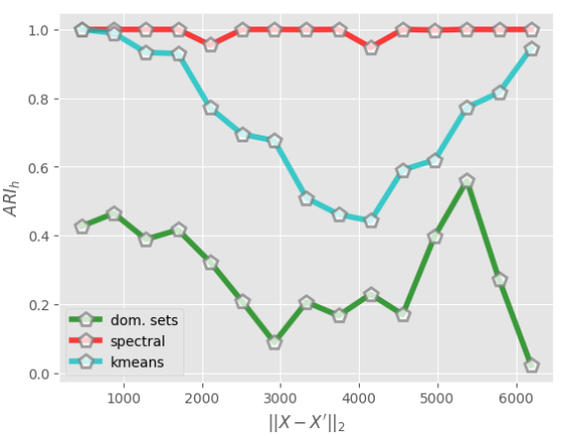


Figure 4.75: ARI_h over the attacker's capacity.

We can notice how the 2 figures above are very similar to the ones provided in Fig. 4.66 and 4.67, highlighting how the most important parameter that affects the results is essentially the percentage of samples moved.

4.3.5 Yale Dataset

In recent years face clustering has assumed an essential role in very sophisticated applications. It consists in grouping images of faces that are similar to each other (same people) for which no labels exists. In [13] and [20] some of multiple applications of face clustering are discussed, like grouping faces together in order to make more efficient face retrieval systems. Consider for example the scenario in which we have a huge dataset of faces and given an input image X we want to find a face similar to X. For solving that problem we need to compute the similarity of X with all faces in the dataset in order to get the most similar face and correspondingly identify the subject. This strategy may end up being computationally wasteful, especially for online applications. An idea for speeding up the system consists in clustering together similar faces and for each cluster a representative prototype is considered. Then, instead of comparing similarities between all the faces, we can limit the computations only to those prototypes.
Even in this scenario we could think to have an attacker may want to break the clustering process for satisfying a certain desire. For instance the attacker may desire to merge a collection of faces into the others, such that at the end one group is removed and possible retrieval of similar faces becomes more difficult.
Due to the sensibility of these applications we decided to test the three algorithms even for the face clustering setting. We decided to use the Yale dataset[3] that contains 165 320×243px grayscale images, 11 per subject. In order to focus our analysis only on faces and even for reducing the computational time, an initial pre-processing was done:

- For each image in Yale dataset use the `Cascade Classifier`[4] for detecting the center c of the face inside the image.
- Construct a box of $k \times k$ px centered in c for extracting the face.
- Use interpolation to down-size the input box into a $k' \times k'$ px image.

We decided to set $k = 150$ and $k' = 28$, in order to preserve a good quality of the images and maintain acceptable computational requirements. In the following we propose an example of face obtained after the pre-processing procedure against the original image:

Figure 4.76: Sample in Yale dataset.

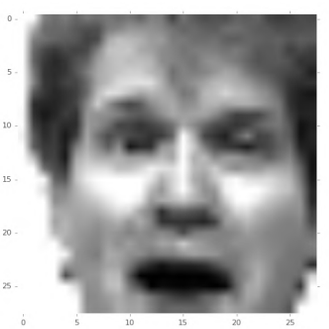


Figure 4.77: Retrieved face after the pre-processing procedure.

[3]Yale dataset
[4]Cascade Classifier documentation.

Then we decided to simulate the scenario in which the attacker wants to move face samples from a cluster to another. We randomly picked two clusters (subject-1 and subject-4), that define our X, with the purpose of moving samples from subject-4 cluster towards the subject-1 group. In absence of adversarial noise the following results are obtained:

Clustering algorithm	*ARI*
Spectral	0.6675
K-Means	0.6675
Dominant Sets	1.0

We can immediately see that in the natural setting Dominant Sets works better than the other clustering algorithms taken in consideration.

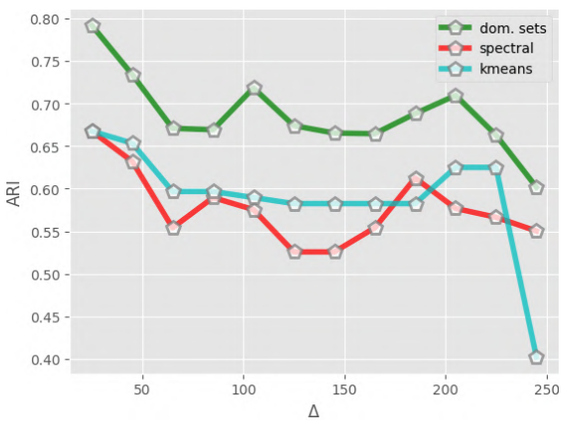


Figure 4.78: *ARI* over maximal power noise Δ.

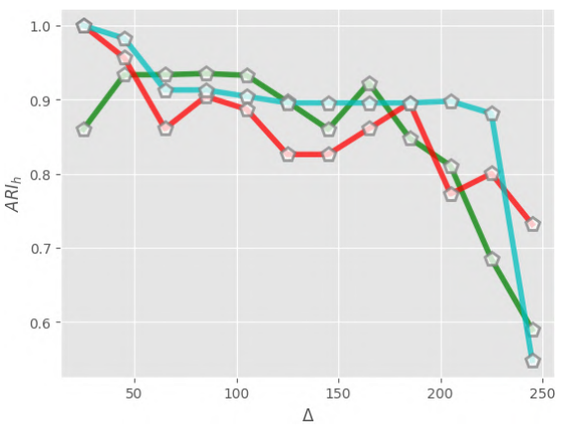


Figure 4.79: ARI_h over maximal power noise Δ.

Looking at 4.78 we can notice that respect to the ground truth Dominant Sets is more robust than the others, but if we consider Fig. 4.79 then we can see that the three algorithms decrease their performance in a similar way. The major advantage of Dominant Sets seems to be that in this configuration and in absence of adversarial noise, it works better than the other two algorithms.

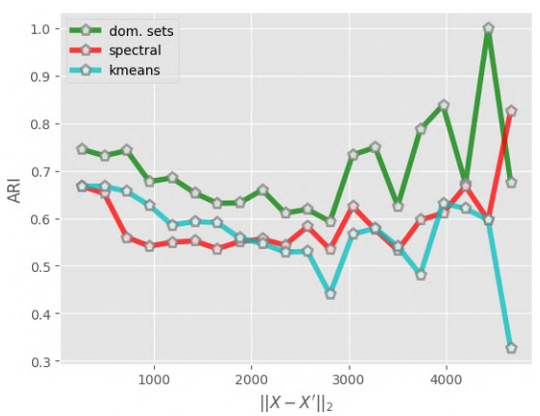


Figure 4.80: *ARI* over the attacker's capacity.

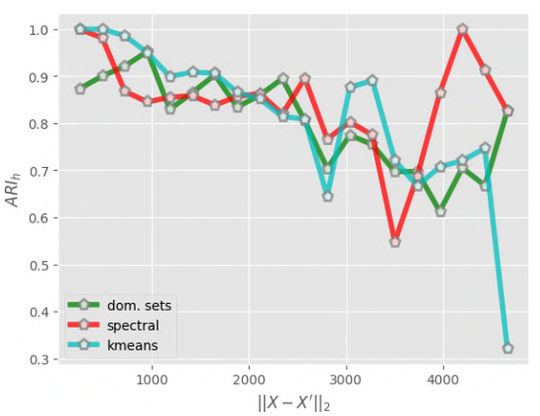


Figure 4.81: ARI_h over the attacker's capacity.

Considerably we can notice that after a certain point of the adversarial's capacity the *ARI* obtained by Spectral and Dominant Sets clustering seems to increase.

Conversely, K-Means is drastically compromised when the adversarial's capacity becomes too high. In the following we propose some examples designed for moving faces of subject-4 towards the cluster of subject-1.

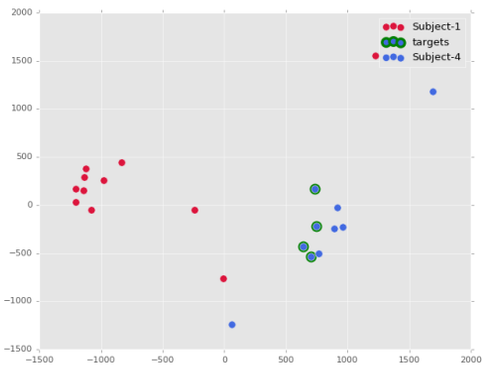


Figure 4.82: PC projection of subject-0 and subject-1 clusters, that define X. Green points are the target samples ($s = 4\%$) sensitive to be moved from subject-0 cluster to the subject-1 one.

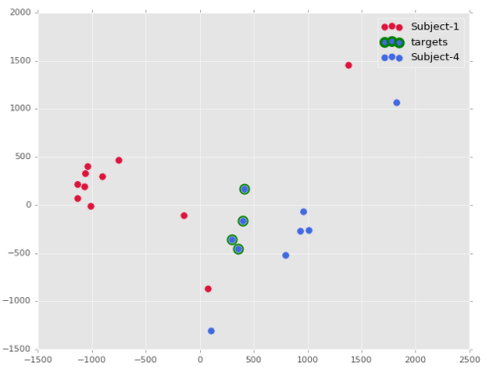


Figure 4.83: Optimal direction noise mask crafted against K-Means with $\Delta = 180$, $s = 1\%$ and local optimization.

Comparing the two principal component projections shown in Fig. 4.82 and 4.83 we can clearly see that the adversarial algorithm is working correctly, moving green target samples from the blue cluster to the red one.

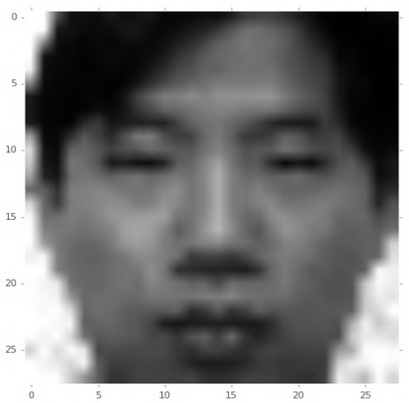


Figure 4.84: Target sample of subject-4 to move towards subject-1 cluster.

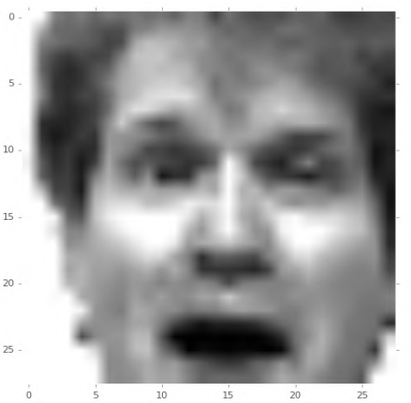


Figure 4.85: Sample in subject-1 cluster.

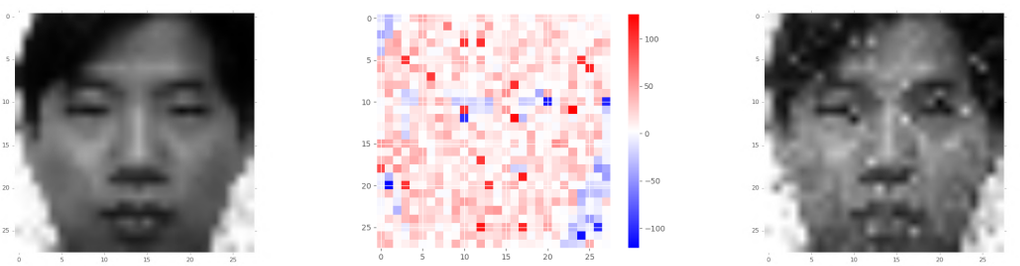


Figure 4.86: Target sample from subject-4 cluster (left), over which is applied the optimal noise adversarial mask (middle) for crafting an adversarial example (right).

Chapter 5

Conclusions and Future Work

In this work, we have presented how adversarial machine learning is increasingly attracting researchers and privates due to the strong implications that it can have in real life. We have shown how in sophisticated domains, such as cyber-security, adversarial perturbations can bring to dangerous conclusions. We have in particular analyzed the impact of adversarial machine learning for the unsupervised paradigm. After having shortly reviewed image segmentation in terms of clustering, we have noted how fooling an image segmentation algorithm can bring sophisticated systems to make dangerous decisions.
We have testified a lack of adversarial analysis in clustering algorithms and we have addressed this lack by performing extensive experimentation on three clustering algorithms, discovering a strong sensitivity to the adversarial perturbations. In particular, we have developed three methods for crafting adversarial examples against K-Means, Spectral and Dominant Sets clustering. The first two methods have been designed for fooling image segmentation algorithms, and we have discussed how they work along with the differences between them. The last one, instead, has been designed for fooling feature-based data clustering. We have shown how the three clustering algorithms behave, for both applications, in presence of adversarial noise. In particular, we have seen how Spectral Clustering seems to be strongly sensitive to small perturbations with respect to the other two clustering algorithms. Moreover, we have seen how K-Means and Dominant Sets preserve a similar behavior against adversarial noise, but the latter has the advantage of working better in absence of adversarial noise.
During the development of this work we have realized some ideas that could be interesting to address in the future. The first one is related to the optimization of the adversarial algorithms. At the moment, the major component that results to be computationally costly is the noise evaluation, which requires multiple iterations of the clustering algorithms. A possible solution could be to develop the proposed clustering algorithms in order to work with GPUs, to speed up the computations. In our implementation the designed code is strongly scalable, meaning that it is possible to run the entire algorithm over parallel architectures without much effort. Right now, due to the absence of GPU-supported implementations of clustering algorithms, we can not run the adversarial algorithms on GPUs. Another idea for speeding up the adversarial target algorithm 5, is to introduce a new heuristic that avoids the injection of noise if the target samples are locally close to the desired cluster. Indeed, at the moment target samples are always moved towards the destination cluster, meaning that the algorithm injects small perturbations even if samples are already locally close. From multiple observations, we

have seen that the optimizer leaves local features more or less unchanged if samples are locally close. This consideration brings us to the conclusion that executing the optimization for those features can lead to waste computational resources, since only negligible perturbations are crafted.
The second topic of interest for future work is the connection between adversarial examples crafted against clustering and supervised classification models. A clustering algorithm can be seen as a way for estimating probability distributions of samples. If samples belong to the same group, then they are probably sampled from the same distribution. We believe that fooling clustering is tightly related to change the classes probability distributions. Supervised models are built with the goal of discriminating samples coming from different classes, therefore we think that adversarial examples crafted to fool clustering can even fool supervised algorithms. If we could verify the correctness of this idea then we might think to use the adversarial masks, like in Fig. 4.71, generated offline against clustering algorithms, for fooling online classifiers. This strategy could be interesting for drastically reducing the computational time required for crafting adversarial examples. The latest idea is related to the design of the target clustering algorithm 5. Indeed, it works moving samples from one cluster towards another one. We can imagine to generalize this framework allowing the attacker to move samples from a cluster towards multiple ones.

Bibliography

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 274–283, 2018.

[2] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2154–2156, 2018.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML/PKDD*, 2013.

[4] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. Is data clustering in adversarial settings secure? In *AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 87–98, 2013.

[5] Battista Biggio, Samuel Rota Bulò, Ignazio Pillai, Michele Mura, Eyasu Zemene Mequanint, Marcello Pelillo, and Fabio Roli. Poisoning complete-linkage hierarchical clustering. In *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*, pages 42–52, 2014. doi: 10.1007/978-3-662-44415-3_5.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[8] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *CEAS 2005 - Second Conference on Email and Anti-Spam, July 21-22, 2005, Stanford University, California, USA*, 2005.

[9] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, AISTATS 2001, Key West, Florida, USA, January 4-7, 2001*, 2001.

[10] Dongyu Meng and Hao Chen. Magnet: A two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 135–147, 2017.

[11] Glenn Milligan and Martha Cooper. A study of the comparability of external criteria for hierarchical cluster-analysis. *Multivariate Behavioral Research - MULTIVARIATE BEHAV RES*, 21:441–458, 10 1986. doi: 10.1207/s15327906mbr2104_5.

[12] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 849–856, 2001.

[13] Charles Otto, Dayong Wang, and Anil K. Jain. Clustering millions of faces by identity. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(2):289–303, 2018. doi: 10.1109/TPAMI.2017.2679100.

[14] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016.

[15] M. Pavan and M. Pelillo. A new graph-theoretic approach to clustering and segmentation. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I, June 2003. doi: 10.1109/CVPR.2003.1211348.

[16] Massimiliano Pavan and Marcello Pelillo. Dominant sets and pairwise clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:167–172, 2007.

[17] Marcello Pelillo and Edwin R. Hancock, editors. *Energy Minimization Methods in Computer Vision and Pattern Recognition - 11th International Conference, EMMCVPR 2017, Venice, Italy, October 30 - November 1, 2017, Revised Selected Papers*, volume 10746 of *Lecture Notes in Computer Science*, 2018. Springer.

[18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[19] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[20] Yichun Shi, Charles Otto, and Anil K. Jain. Face clustering: Representation and pairwise constraints. *IEEE Trans. Information Forensics and Security*, 13(7):1626–1640, 2018. doi: 10.1109/TIFS.2018.2796999.

[21] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 2007.

[22] Gregory L. Wittel and Shyhtsun Felix Wu. On attacking statistical spam filters. In *CEAS 2004 - First Conference on Email and Anti-Spam, July 30-31, 2004, Mountain View, California, USA*, 2004.

[23] Dong Yin, Kannan Ramchandran, and Peter Bartlett. Rademacher complexity for adversarially robust generalization. *CoRR*, abs/1810.11914, 2018.