Temporal Data Mining : Learning From Positive Examples

Ahmed-Antoine BOUHEMAD

Internship

4 juillet 2019

This report is downloadable at the following address
https://github.com/ahmedAnB/Data-clustering

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UGA
Bâtiment IMAG
Université Grenoble Alpes
700, avenue centrale
38401 Saint Martin d'Hères
France
tel : +33 4 57 42 22 42
fax : +33 4 57 42 22 22
http://www-verimag.imag.fr/

Temporal Data Mining : Learning From Positive Examples

Ahmed-Antoine BOUHEMAD

VERIMAG

4 juillet 2019

Abstract

During my internship I worked on Data Clustering. This task aims to analyse data, by dividing it in set of differents structures, using common features which correspond to a spatial similarity. This technique is used in many fields including image analysis, bioinformatics, machine learning.

Keywords: Data Clustering, Data mining

Tutor: Thao DANG
Nicolas BASSET

Notes:

Contents

Part I
Contexte

Dans cette partie nous allons montrer quels ont été les principes pour générer N points en D dimenions. Pour vérifier la cohérence des algorithmes, les points ont été générés à partir de differents clusters initiaux. Ces clusters initiaux peuvent être de differentes formes : des rectangles de tailles differentes ou des cercles de rayons differents
Le code utilisé pour cette partie est dans le fichier CREATION_POINT.PY

1 Génération de Rectangles

Les fonctions utlisées pour générer les rectangles sont : CREATION_POINT_RECTANGLES(NB_POINT, NB_RECTANGLE, DIMENSION) et CREATION_POINT_RECTANGLES_2(NB_POINT, NB_RECTANGLE, DIMENSION).
Ci-après, le pseudo-code correspondant.

Algorithm 1 Générer N points dans $n_{rectangles}$ en D dimensions

Require: $N \geq 0, n_{rect} \geq 0, D \geq 0$
Ensure: liste de N points répartis dans n_{rect} rectangles
  $m \leftarrow \lfloor \frac{N}{n_{rect}} \rfloor$
  $points \leftarrow [\,]$
  for $j \in [\![1, n_{rect}]\!]$ **do**
    $cote \leftarrow$ génération d'une liste de D nombre aléatoie dans]0, 0.5]
    sauvergarde des sommets et des cotés qui permettent de representer des carrés
    for $i \in [\![1, m]\!]$ **do**
      $point \leftarrow$ génére un point dans le rectangle j
      on stocke le points dans la listes points
    end for
  end for
  return points, carrés

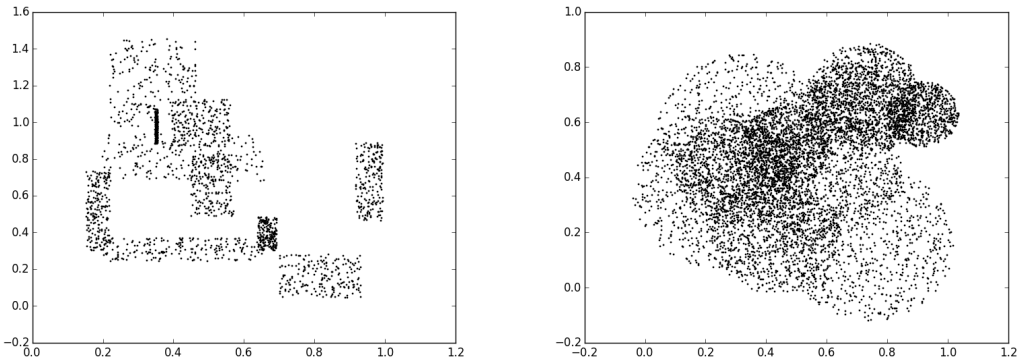

Figure 1: Génération de points sur 10 rectangles et de cercles

2 Génération de cercles

Les fonctions utilisées pour générer les cercles ressemblent fortement à celle pour les rectangles qui sont : CREATION_POINT_CERCLES et CREATION_POINT_SUR_CERCLE. Cette dernière fonction permettant de générer des points de manière uniforme sur un cercle. Nous allons décrire l'algorithme utilisé pour cette génération uniforme.

Algorithm 2 Générer N points dans $n_{cerlces}$ en D dimensions

Require: $N \geq 0, n_{cercles} \geq 0, D \geq 0$
Ensure: liste de N points répartis dans n_{rect} cercles
  $m \leftarrow \lfloor \frac{N}{n_{rect}} \rfloor$
  $points \leftarrow [\,]$
  for $j \in [\![1, n_{cerlces}]\!]$ **do**
    $rayon \leftarrow$ uniform(0.1, 0.3)
    $center \leftarrow$ point aléatoire en D dimensions
    for $i \in [\![1, m]\!]$ **do**
      $point \leftarrow$ génére un point dans le cercle j
      $point_{cercle} =$ [center[i] $+(xi - center[i]) \times \frac{rayon}{distance(point,\ center)}$ for i, xi in enumerate($point$)]
    end for
  end for
  return points, carrés

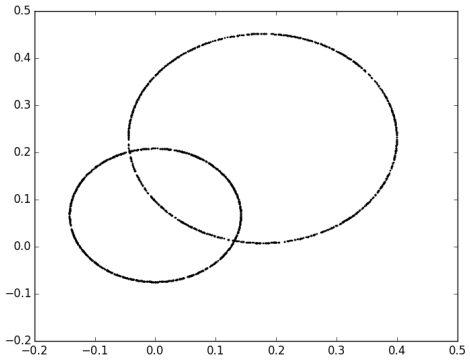

Figure 2: Génération de points sur deux cercles

Part II
Résolution du problème

Dans cette partie, nous allons voir la mise en place pour régler le problème de partionnement de données. C'est-à-dire comment l'algoritme a réussi à comprendre quelles sont les points qui étaient assez proches pour former un cluster. Dans notre algorithme les clusters seront des rectangles et l'algorithme devra rendre les clusters optimaux qu'il a trouvé. Il faut que le nombre de clusters soit minimal et qu'ils occupent le moins d'espace possible.
Pour résoudre ce problème, l'algorithme développé va dans un premier temps créer une table hachage qui permettra de partitionner l'espace en un quadrillage, puis l'algorithme en déduira un premier ensemble de rectangles, enfin l'algoritme appliquera une méthode des plus proches voisins afin d'avoir le nombre optimal de clusters. Le code se trouve dans le fichier MAIN.PY pour la première version de l'algorithme des plus proches voisins et la création de la table de hachage.

3 Création de la table de hachage

L'algorithme générant la table de hachage fonctionne ainsi : pour chaque point, il va créer une clef qui va dépendre des coordonnées du point, et si les points ont la même clef,

on pourra en déduire qu'ils sont proches les uns des autres et ils vont donc appartenir au même ensemble de points stockés dans la table de hachage pour la clef correspondante.

Cette algorithme est de complexité $O(N \times D)$

Algorithm 3 Créer une table de hachage regroupant des points proches

Require: $points \neq \emptyset, D \geq 0, \varepsilon > 0$
Ensure: table de hachage
  $m \leftarrow \lfloor \frac{N}{n_{rect}} \rfloor$
  $HT \leftarrow \{\ \}$,
  for $point = (x_1, ..., x_D) \in points$ **do**
    $lower = (\lfloor \frac{x_i}{\varepsilon} \rfloor)_{\forall i \in [\![1,D]\!]}$
    if $lower \in HT.keys$ **then**
      On ajoute point à $HT[lower]$
    else
      $HT[lower] = [point]$
    end if
  end for
  return HT

4 Déduction des rectangles minimums

Pour trouver un premier ensemble de clusters, on va extraire chaque clef de la table de hachage afin d'en déduire une bounding box pour les points appartenants à une même clef. Afin d'avoir cette bounding il suffit de prendre le rectangle le plus petit qui contient tout les points en question.

5 Algorithme des plus proches voisins

On applique ensuite l'algoritme des plus proches voisins qui va nous retrouver les deux rectangles les plus proches. Cet algorithme est un algorithme naïf car il va tester tous les voisins possibles afin de déduire le

plus proche. Il est en complexité $O(n^2_{cluster})$ mais nous l'avons choisit, car en dimensions importantes il est plus efficaces que d'algorithmes tels que k-means, kd-tree.

Algorithm 4 Retourner les rectangles les plus proches

Require: $set_{rectangles} \neq \emptyset, D \geq 0$
Ensure: couple de rectangles les plus proches
  $nearest \leftarrow \{set_{rectangles}[0], set_{rectangles}[1]\}$
  $min\_distance \leftarrow distance(set_{rectangles}[0], set_{rectangles}[1]),$
  for $i \in [\![0, n-1]\!]$ **do**
    for $j \in [\![i+1, n-1]\!]$ **do**
      $dist = distance(set_{rectangles}[i], set_{rectangles}[j])$
      if $dist < min\_dist$ **then**
        $nearest \leftarrow \{set_{rectangles}[i], set_{rectangles}[j]\}$
        $min\_distance \leftarrow dist$
      end if
    end for
  end for
  return nearest

6 Conditions d'arrêt

Pour l'instant, la condition d'arrêt mise en place repose sur le nombre de clusters que l'utilisateur veut en sortie. Cependant, on pourra utiliser une condition d'arrêt portant sur l'évolution d'une fonction de coût en fonction du nombre de cluster et choisir le nombre de cluster minimisant la fonction et n'étant pas trop grand.

7 Exemple d'évolution de l'algorithme

Afin de visualiser l'algorithme j'ai mis en place une bibliothèque permettant d'afficher les points et les rectangles dans AFFICHAGE_POINT.PY.

8 Optimisation Nearest Neighboor

Pour optimiser le temps de l'algorithme du plus proche voisins, on a mit en place un système de liste triée ayant une initialisation en complexité quadratique mais pour la mise à jour et trouver le minimum la complexité est en $O(n \log n)$.

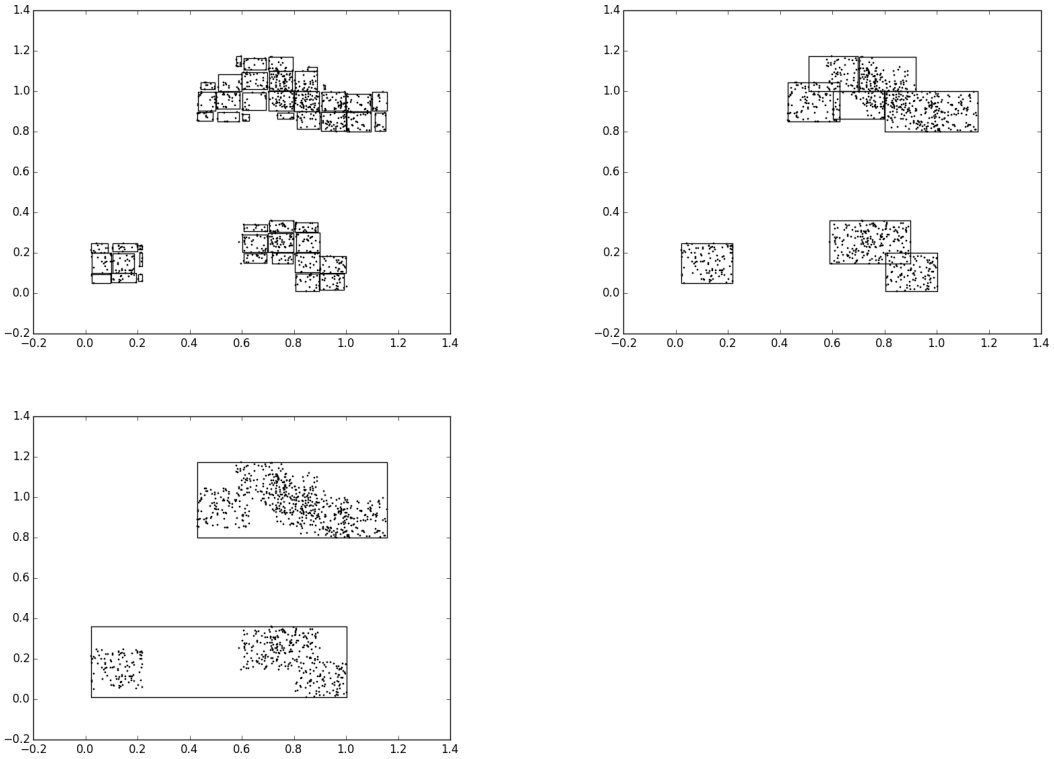

Figure 3: Visualisation de l'algorithme pour un ensemble de clusters initiaux carrés

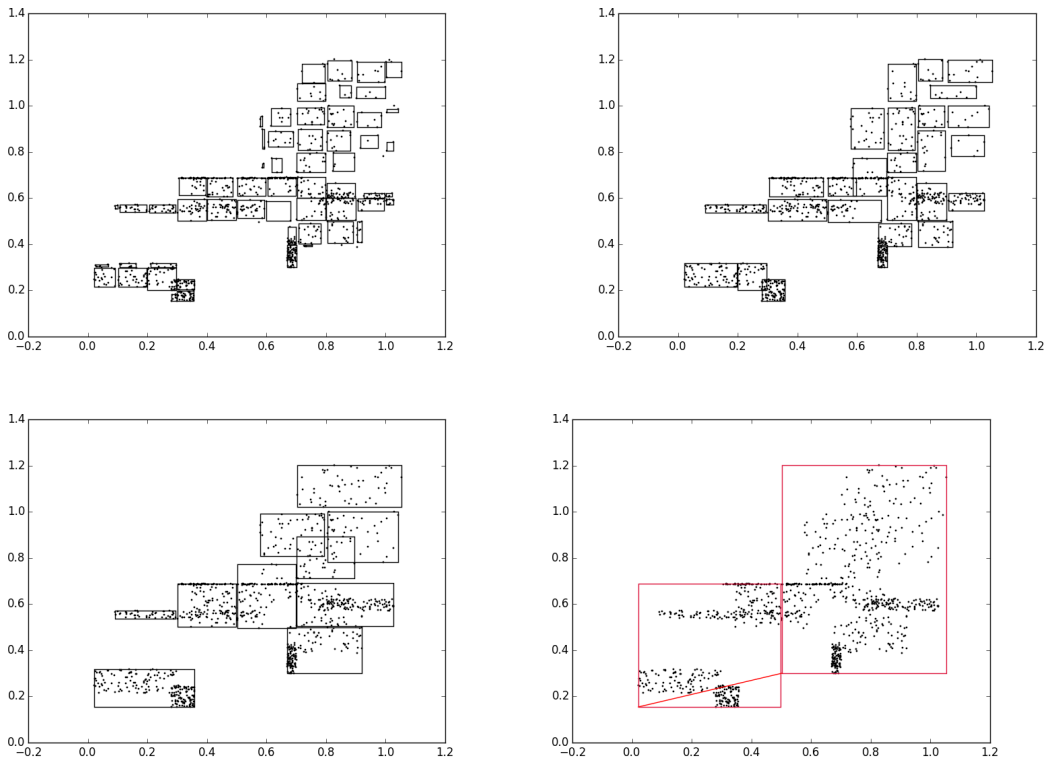

Figure 4: Visualisation de l'algorithme pour un ensemble de clusters initiaux en forme de rectangles

Part III
Résultats

9 Réalisation des expériences

Pour tester les algorithmes et donc évaluer les performances en fonction de différentes caractéristiques, on a réalisé des expériences dont on peut voir les résultats ci-après. Les fonctions pour tester les algorithmes ce trouvent dans TESTS.PY et EXPERIENCES.PY.

9.1 1^{re} expérience

On a évalué le taux d'erreur, pour différentes initialisations de l'algorithme et differents nombres de clusters initiaux.

9.1.1 Calcul du taux d'erreur

Le calcul du taux d'erreur se fait de manière probabilistique. Nous allons approximer l'aire de l'ensemble des clusters initiaux et l'ensemble des clusters trouvés après application de l'algorithme et nous allons évaluer le rapport des deux aires. Pour ce faire, nous allons calculer le taux de faux positive et de faux négatif.
Ainsi :

$$Erreur = \frac{volume(Rectangle\_green) - volume(Rectangle\_blue)}{volume(Rectangle\_green)}$$

Nous utilisons une approximation des volumes avec une méthode de Monte-Carlo:

$$Erreur = \frac{card(points \in Rectangle\_green, \notin Rectangle\_blue)}{card(points \in Rectangle\_green)}$$

Par conséquent le taux d'erreur en comptant les points faux positives est donné par :

$$Rectangle\_green = Rectangles\_initiaux$$

et

$$Rectangle\_blue = Rectangles\_appris$$

Reciproquement le taux d'erreur en comptant les points faux négatives est donné par :

$$Rectangle\_green = Rectangles\_appris$$

et $Rectangle\_blue = Rectangles\_initiaux$

On a donc calculé comment variait le taux d'erreur avec la taille de la grille de la table de hachage et avec le nombre de clusters initiaux.

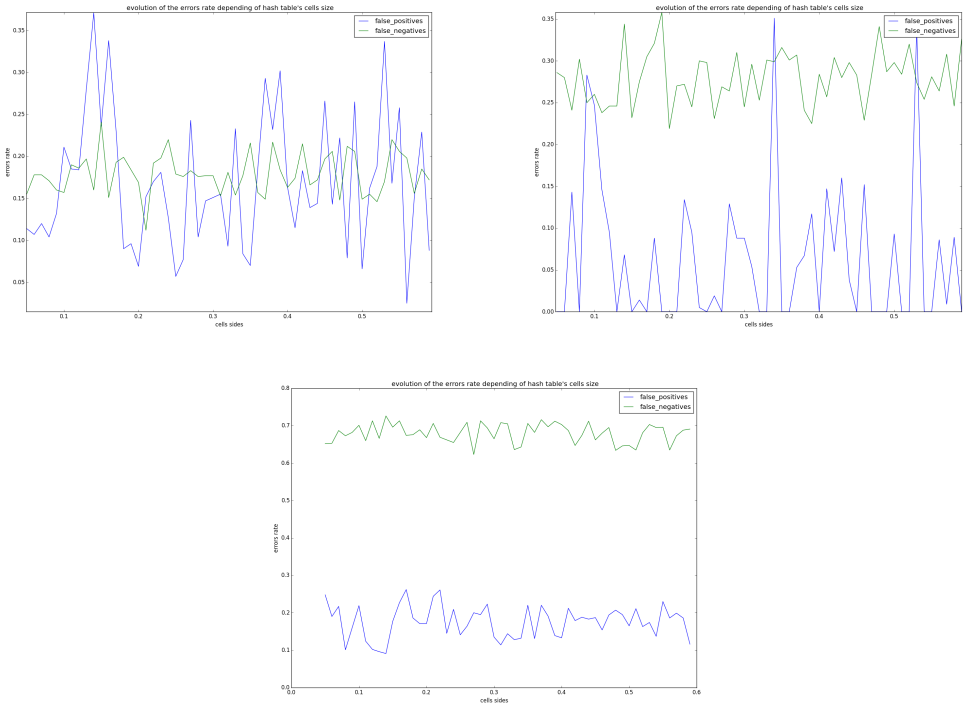

Figure 5: Evolution du taux d'erreur en fonction de la taille de la table de hachage pour 1000 pts 10 clusters en 3D et en 4D, et 5000 points 50 clusters en 3D (gauche vers droite)

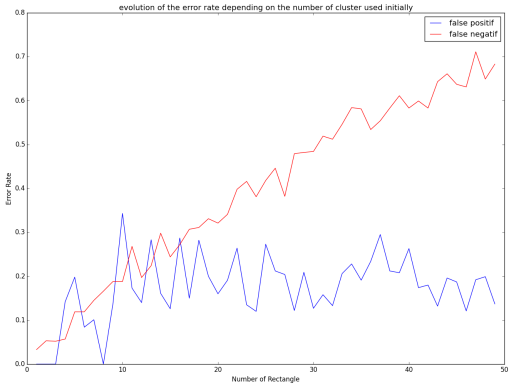

Figure 6: Evolution du taux d'erreur en fonction du nombre de cluster initiaux avec 5000 points et 50 cluster maximum en 3 dimension

9.1.2 Résultats de la 1ère expérience

9.2 2^{eme} expérience

Le but de cette expérience est de montrer un phénomène bien connu du regroupement de données en hautes dimensions : *le fléau de la dimension*. Il traduit le fait qu'en augmentant la dimension, le temps de calcul sera plus long car les points deviennent plus isolés.

9.2.1 Fléau de la dimension

Pour réaliser cette expérience on a lancé l'algorithme en générant 1000 points puis le programme a augmenté le nombre de dimensions graduellement.

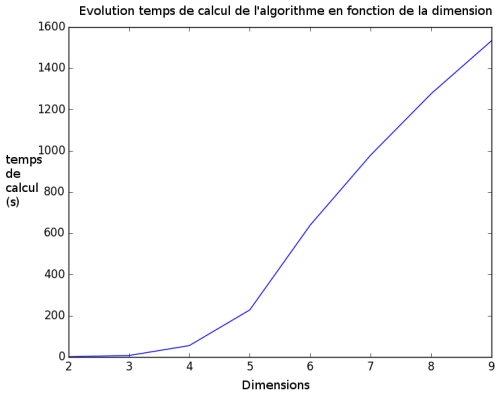

Figure 7: algorithme 1 : Visualisation de l'explosion du temps de calcul en fonction de la dimension

9.2.2 Isolation des points

Afin de comprendre les origines du phénomène précédent, on a tracé l'évolution du nombre de cellules que contenait la table de hachage en fonction de la dimension et de la taille de chaque cellule.

9.3 3^{eme} expériences : évolution du coup

Cette expérience avait pour objectif de traduire les valeurs prises par une certaine fonction de coût en fonction de l'évolution de l'algorithme.
Pour notre expérience, la fonction de coût est la suivante : $c(R = [p1, p2]) = distance(p1, p2)$. Dans notre cas, la distance en question dépend de la norme-2

10 Evalutation des performances

Le deuxième algorithme de recherche des plus proches voisins est beaucoup plus rapide que l'ancien. On montre un tableau qui compare leurs performances pour 1000 points puis on testera l'algorithme optimisé afin de voir ses limites.

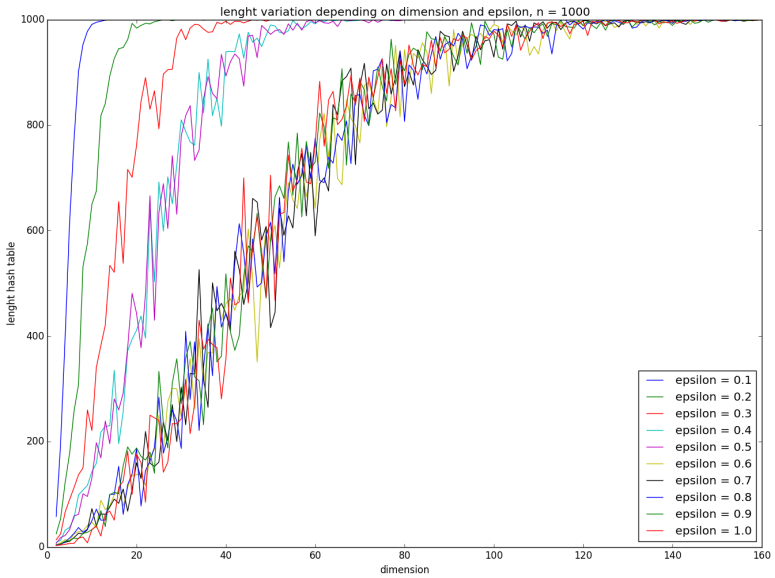

Figure 8: Evolution de la longueur de la table de hachage en fontion de epsilon et de la dimension (fléau de la dimension)

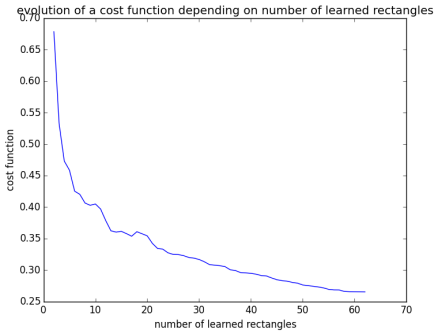

Figure 9: Evolution du coût en fonction de l'évolution de l'algorithme pour 1000 points et 10 clusters en 2 dimensions

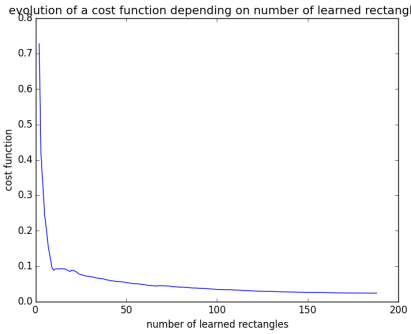

Figure 10: Evolution du coût en fonction de l'évolution de l'algorithme pour 1000 points et 10 cluters en 3 dimensions

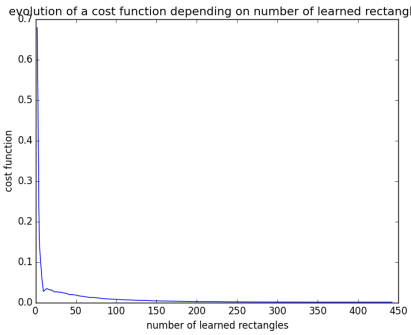

Figure 11: Evolution du coût en fonction de l'évolution de l'algorthme pour 1000 points et 10 cluters en 4 dimensions

Dimension	Temps algorithme Naïf (s)	Temps liste triée
2	0.16	0.1
3	11	4
4	211	16
5	2 151	81
6	16 082	175
7	++	198
8	++	233

Table 1: Temps de calcul des deux algorithmes pour 1000 points et 10 clusters intiaux

Dimension	Temps (s)
2	0.49
3	215
4	3 468
5	15 122
6	33 017

Table 2: Algorithme optimisé : temps de calcul pour 5000 points et 50 clusters intiaux

Part IV
conclusion

Pour conclure afin de résoudre le problème de clustering de données on a mis en place un algorithme composé de deux parties, une premiere parte permet de partitionner les points à l'ade d'une table de hachage, puis la seconde partie de l'algorithme fusionne petit à petit les clusters les plus proches afin d'en avoir un nombre optimal. Je remercie Thao et Nicolas pour leur aide et leur bienveillance tout au long de mon stage, je me remercie aussi l'equipe tempo et les equipes de VERIMAG pour leurs précieux conseils.