Universidad Zaragoza
1542

IDENTIFICACIÓN DE PATRONES Y ALGORITMOS DE CONSOLIDACIÓN EN BASES DE DATOS DE POSICIONAMIENTO

Pilar Barbero Iriarte

Director: Tomás Alcalá Nalváiz

17 de diciembre de 2015

Universidad de Zaragoza

INTRODUCCIÓN

Contexto:

- Empresa Zaragozana de telecomunicaciones.
- Almacenamiento de posiciones GPS de sujetos.

INTRODUCCIÓN

Problemas:

- Capacidad de guardado de posiciones limitada.
- No existe preprocesado antes de la inserción.
- No existe postprocesado después de la inserción.
- No todas aportan información.

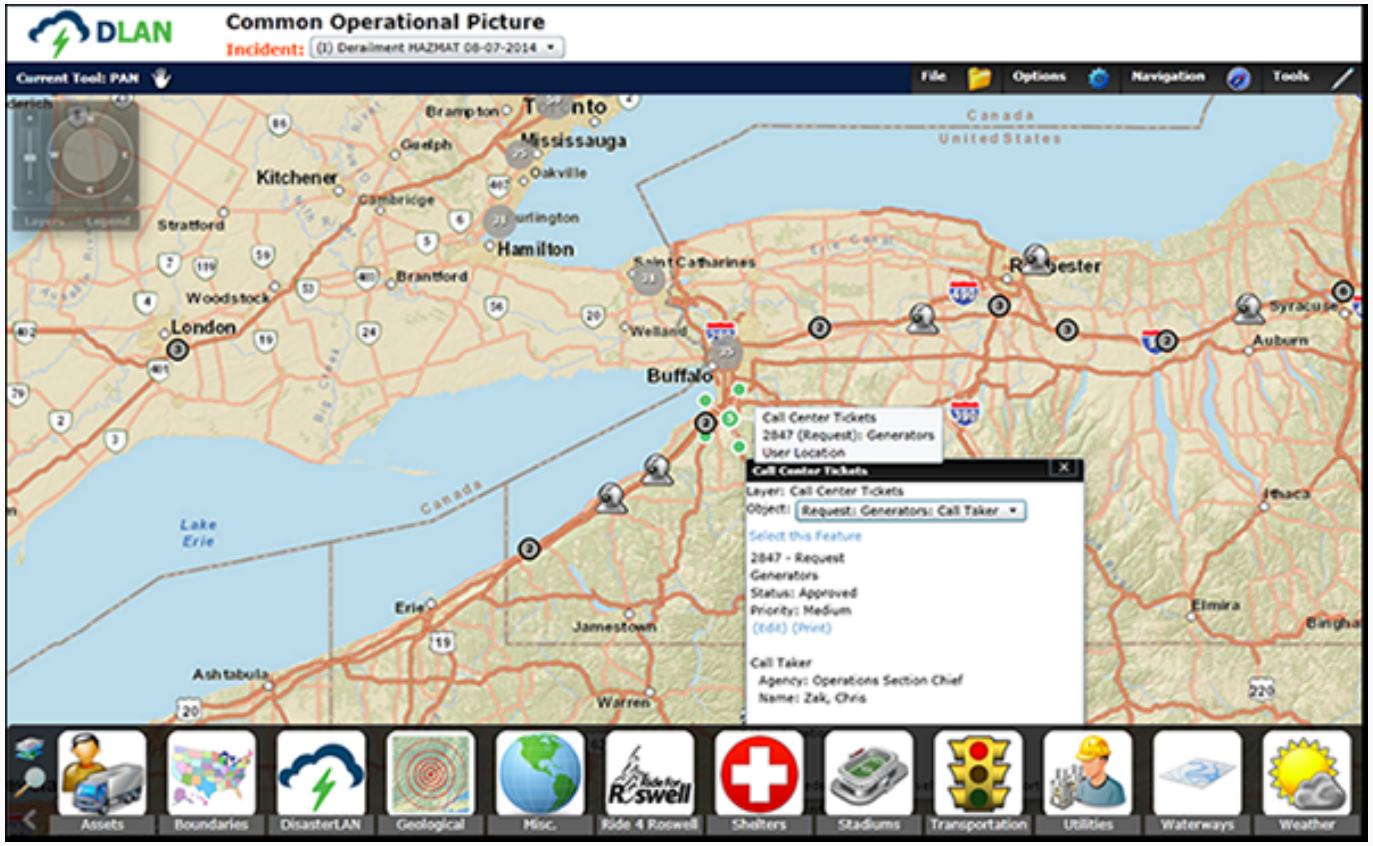
INTRODUCCIÓN

Objetivo:

- Eliminar posiciones repetidas.
- Eliminar posiciones que no aporten información.

ANÁLISIS DE LOS DATOS

- Dos bases de datos suministradas posiciones en torno a las ciudades de Salvador de Bahía y Río de Janeiro.
- 1971 recursos distintos.
- 4599974 posiciones en Salvador de Bahía y 6928467 en Río de Janeiro.



ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea (km/h)

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea (km/h)
- **Orientación**: entero que representa la orientación respecto al norte en grados

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea (km/h)
- **Orientación**: entero que representa la orientación respecto al norte en grados
- **Fecha**: dato tipo fecha que transformamos en un `timestamp`.

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea (km/h)
- **Orientación**: entero que representa la orientación respecto al norte en grados
- **Fecha**: dato tipo fecha que transformamos en un `timestamp`.
- **Cobertura**: booleano que indica si tiene cobertura

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea (km/h)
- **Orientación**: entero que representa la orientación respecto al norte en grados
- **Fecha**: dato tipo fecha que transformamos en un `timestamp`.
- **Cobertura**: booleano que indica si tiene cobertura
- **Error**: booleano que indica si ha habido error en la toma de posición

¿Cómo abordar el problema?

- Desarrollo de algoritmos de consolidación a través de nociones de distancia y tiempo.
- Uso de algoritmos de *clustering* con el fin de identificar varias posiciones con su centro del clúster y consolidarlas en ésta.
- Se elige el el lenguaje `Python` por ser un lenguaje que tiene orientación a objetos y la gran cantidad de librerías científicas de las que consta.

POSICIÓN

Se define la clase `Position` en `Python` de la siguiente manera:

```
class Position:
    def __init__(self, id, resource, lat, lon, speed, track, date):
        self.id = id
        self.resource = resource
        self.lat = lat
        self.lon = lon
        self.speed = speed
        self.track = track
        self.date = date
```

NOCIONES DE VECINDARIO

Distintas nociones de vecindario para los algoritmos de consolidación simple,

- Vecindad involucrando el tiempo
- Vecindario utilizando la distancia euclídea
- Vecindario involucrando velocidad
- Vecindad t_0-alcanzable

VECINDAD INVOLUCRANDO EL TIEMPO

Las posiciones de nuestros sujetos vienen muestreadas además con el instante en el que fueron tomadas.

Definimos esta distancia temporal tal que:

$$d_T(p_0, p) = |time_p - time_{p_0}| < \delta$$

```
def is_neighboorhoudByTime(self, p, lapse):
    return abs(self.time − p.time ) < lapse
```

VECINDARIO UTILIZANDO LA DISTANCIA EUCLÍDEA

Utilizando la distancia euclídea, definimos un vecindario de la siguiente manera:

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < \varepsilon$$

donde p es un punto con latitud lat_p y longitud $long_p$.

```
def IsInNeighEUSimple(self, p, eps):
    return self.distance_eu(p) < eps
```

VECINDARIO INVOLUCRANDO VELOCIDAD

A mayor velocidad, puntos más alejados de lo que consideraríamos en el primer caso (fuera de nuestro vecindario simple), podrían estar dentro de nuestro nuevo radio, que dependería de la velocidad instantánea.

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < \varepsilon \cdot vel_{p_0}$$

```
def IsInNeighSpeedRelative(self, p, eps):
    if self.speed != 0:
        return self.distance_eu(p) < eps * self.speed
    else:
        return False
```

VECINDAD t_0-ALCANZABLE

Se fija intervalo de tiempo t_0, con el cual se define una vecindad t_0-alcanzable .

Sea la velocidad instantánea vel_{p_0}

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < vel_{p_0} \cdot t_0$$

- A velocidad reducida, vecindad t_0-alcanzable menor.
- A mayor velocidad, mayor vecindad t_0-alcanzable.

```
def IsInNeighT0Reachable(self, p, t0):
    return self.distance_eu(p) < t0 * self.speed
```

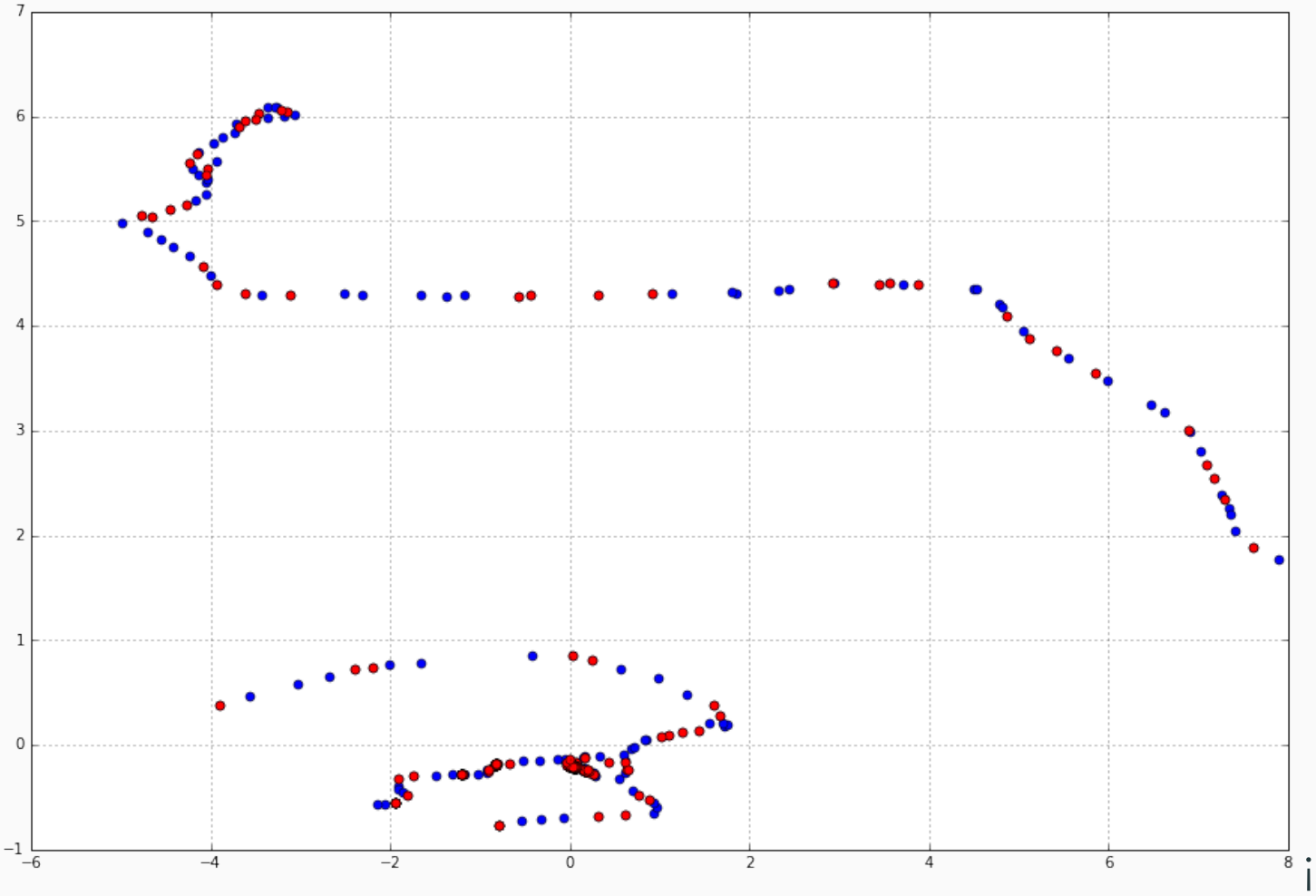
CONSOLIDACIÓN POR DISTANCIA

Utilizando los tres tipos de vecindarios que hemos definido, definimos el siguiente método que realizará la consolidación del tipo que le indiquemos:

```
1: function ConsolidationByDistance(positions, typeOfDistance, eps, t0)
2:     for each pos in positions do
3:         if pos.IsInNeighBorhood(typeOfDistance, next(pos), eps) then
4:             Remove pos from DB
5:         else
6:             Maintain pos in DB
7:         end if
8:     end for
9: end function
```

EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR DISTANCIA

Se fija un $\varepsilon = 0{,}0001$ y se realiza una consolidación utilizando la distancia euclídea:



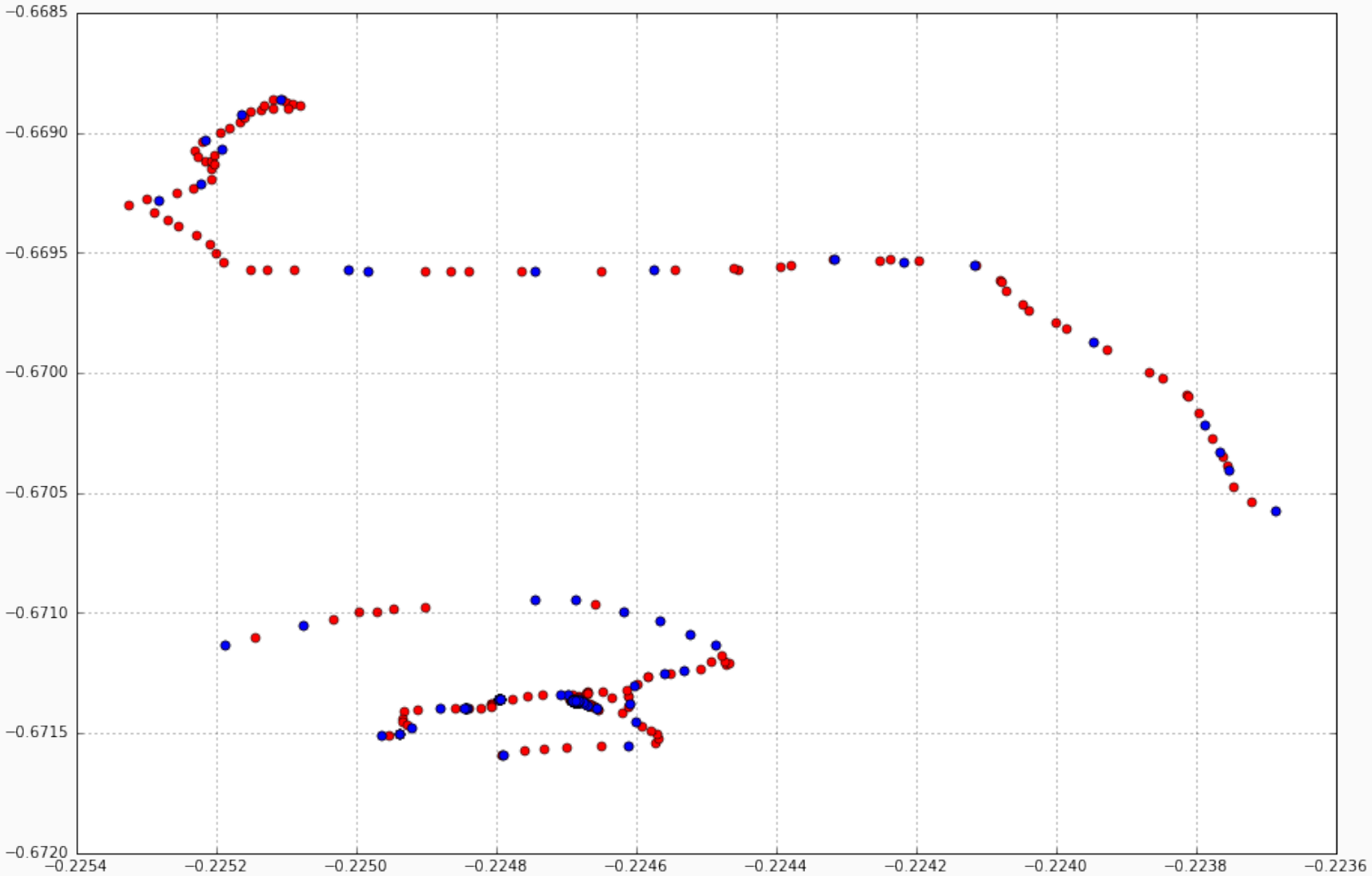
CONSOLIDACIÓN POR TIEMPO

Se fija un lapso de tiempo que se debe cumplir entre posición y posición, y se eliminan todas aquellas que estén cuya distancia temporal con su siguiente esté por debajo de este lapso fijado.

```
1: function ConsolidationByTime(positions, lapse)
2:     for each pos in positions do
3:         nextpos = pos++
4:         if IsInNeighboorhodByTime(nextpos, pos, lapse) then
5:             Remove pos
6:         end if
7:     end for
8: end function
```

EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR TIEMPO

Se realiza una consolidación por tiempo con un lapso de 20 segundos:



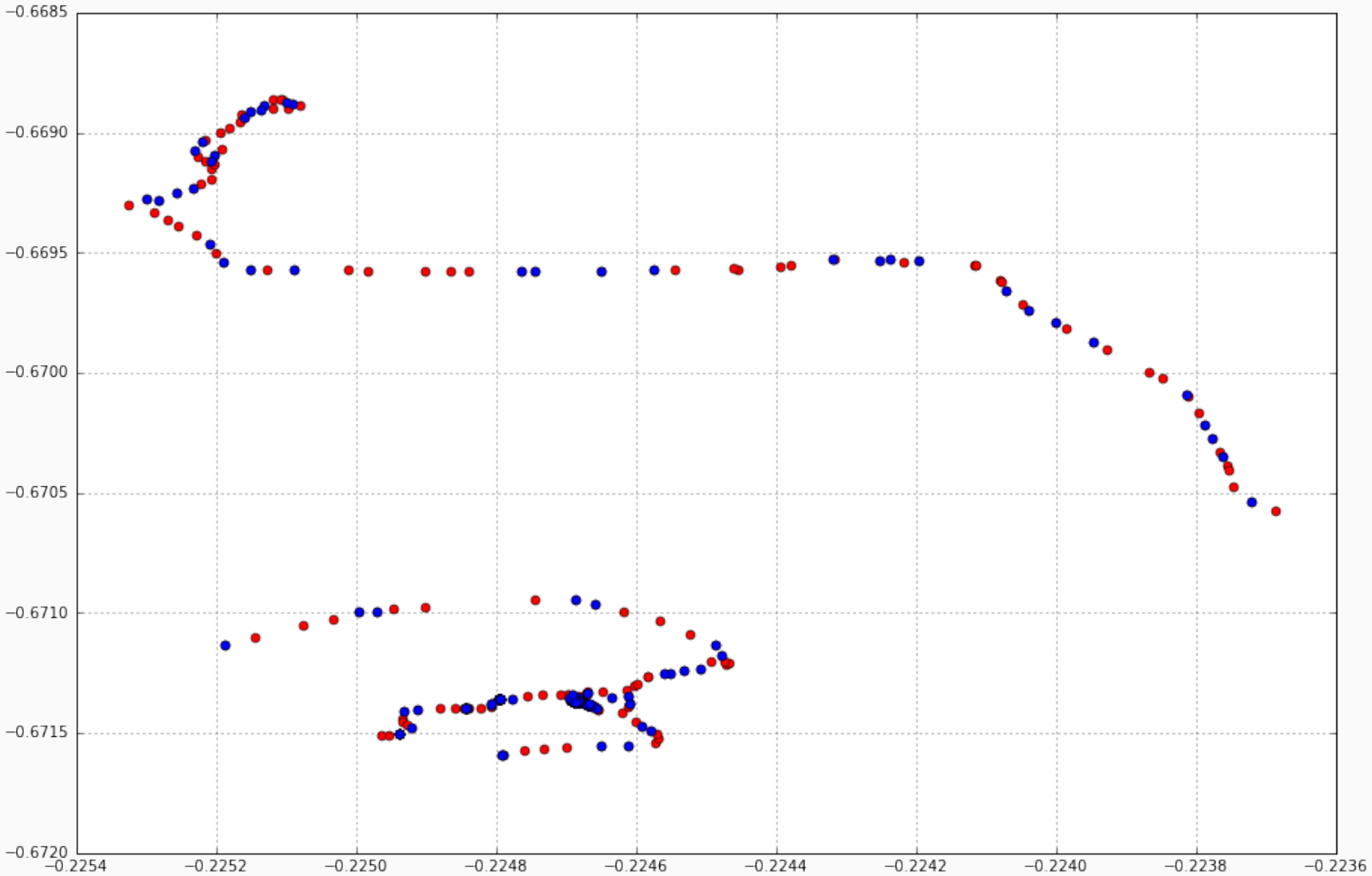
Consolidación por adelgazamiento

Se puede recurrir a un tipo de consolidación en la cual dada una lista de posiciones normalmente antiguas, se elimine un subconjunto de estas, por ejemplo, 3 de cada 5.

```
1: function ConsolidationByThinning(positions, j, k)          ▷ j < k
2:     for each pos in positions do
3:         if position.Index %k == 0 then
4:             for i = 0; i < j; i++ do
5:                 Remove position with index == position.Index
6:             end for
7:         end if
8:     end for
9: end function
```

EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR ADELGAZAMIENTO

Se realiza una consolidación por adelgazamiento, se mantienen 2 posiciones de cada 5:



Algoritmos de consolidación asociados a métodos de clustering

- Técnica o conjunto de técnicas multivariantes utilizadas en Minería de datos y Modelización matemática utilizadas para clasificar a un conjunto de individuos en grupos homogéneos.

Algoritmos a analizar:

- K-means
- DBSCAN
- DJ-Clúster

K-means es un método eficiente de *clustering* que tiene como objetivo la partición de un conjunto de n elementos en k grupos distintos. Dado un conjunto de n elementos, se construye dicha partición $S = \{S_1, S_2, \ldots, S_k\}$ con el fin de minimizar el término del error cuadrático:

$$\sum_{i=1}^{n} \sum_{x \in S_i} \mathrm{d}(x, m_i)$$

donde m_i es el centro de cada clúster S_i y $\mathrm{d}(x, m_i)$ es la distancia definida entre el punto x y m_i.

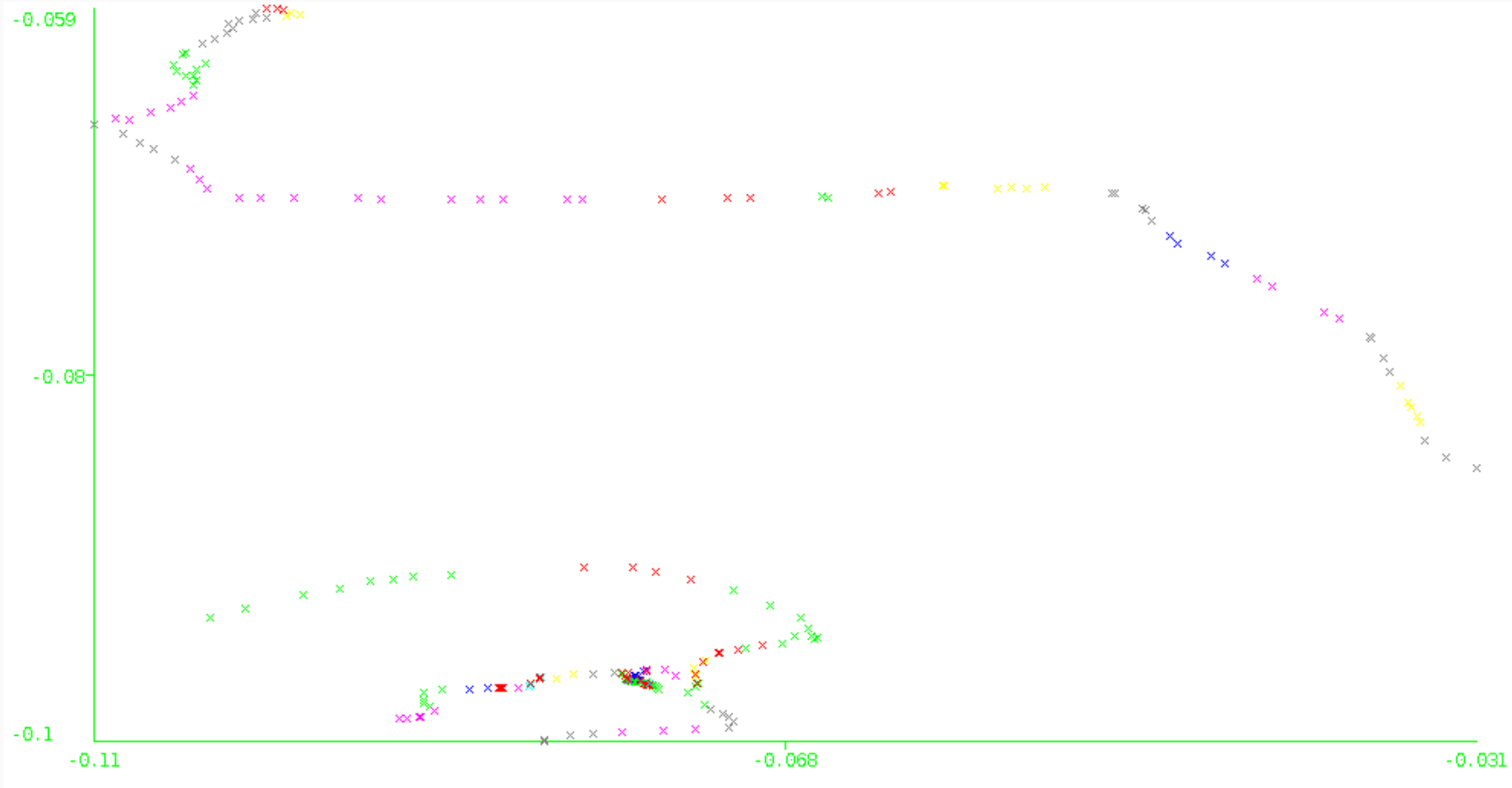
K-MEANS

Se procede:

1. Se prefija un número de clústers.
2. Conformamos los primeros centroides de los grupos de manera aleatoria.
3. Se itera sobre cada punto, encuentra el centro de clúster más cercano y se lo asigna a dicho clúster.
4. Se recalcula los centroides de los clústers y se reitera hasta que el error cuadrático se minimiza o se estabiliza.

EXPERIMENTO CON K-MEANS

Se utiliza el software **Weka** sobre un sujeto con 2000 posiciones para una consolidación al 25 %, es decir, a 500.



CONCLUSIONES K-MEANS

Resultados:

- 10 iteraciones
- Error cuadrático: 0,117363
- número de posiciones que ha agrupado por clúster entre 1 y 9

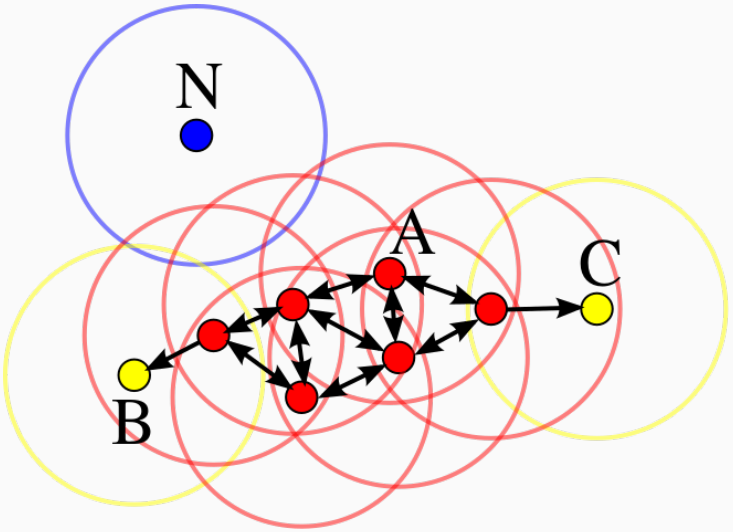
Inconvenientes:

- Número de clústers prefijado a 500.
- K-means no determinístico.
- No hay puntos considerados ruidos, sólo clústers unipuntuales.

DBSCAN

DBSCAN es un algoritmo de clustering basado en la densidad por lo que encuentra el número de clústers comenzando por una estimación de la distribución de densidad de los nodos correspondientes.

- Punto p núcleo: si posee un número mínimo de puntos (*minPts*) en su vecindario sobre ε.
- Punto q alcanzable por p: si existe un camino $p_1, \ldots, p_n$ tal que p_i está en el vecindario de p_{i+1} (o viceversa).
- Aislados: puntos no considerados ni núcleo ni alcanzables.



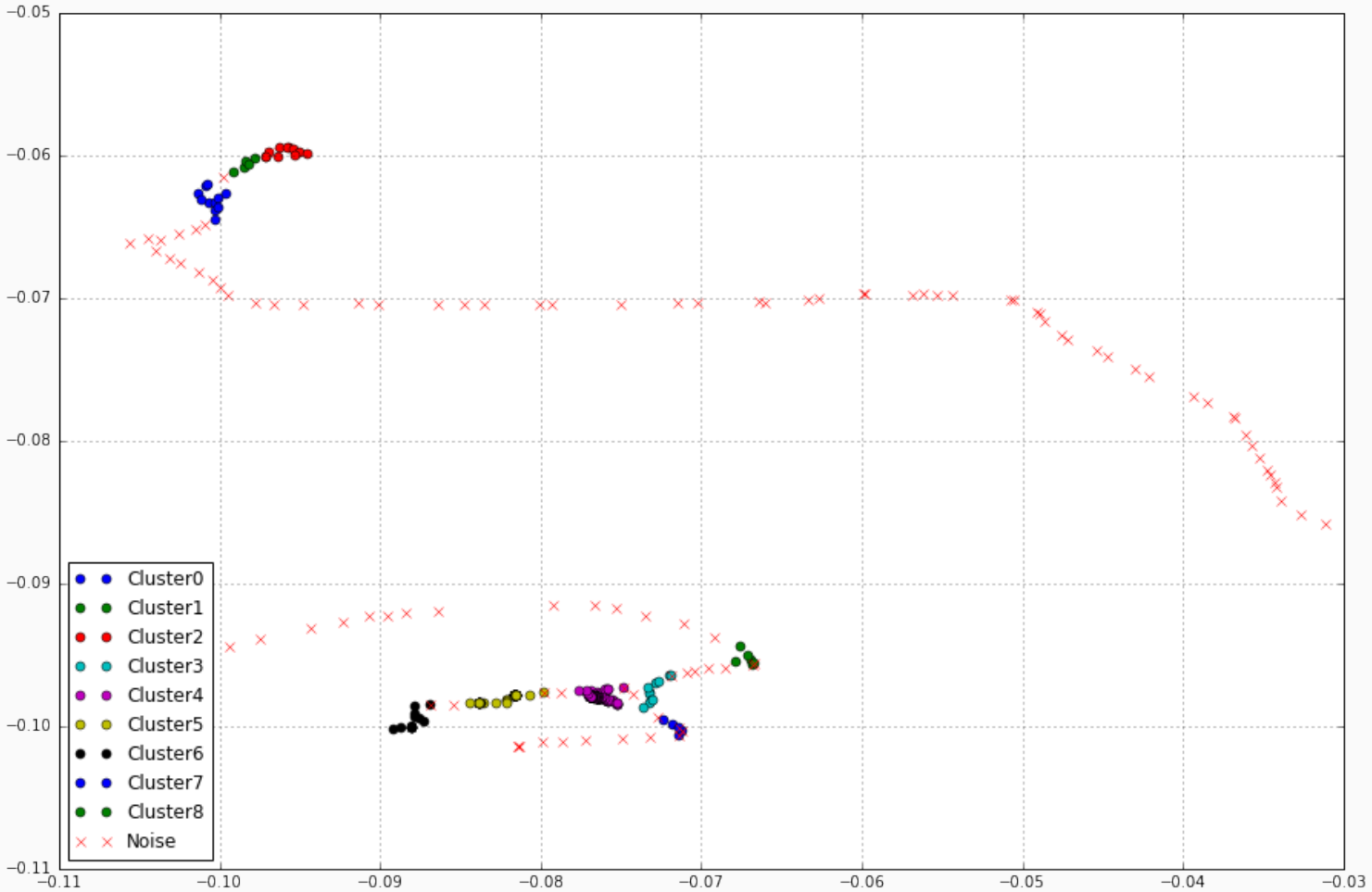
DBSCAN

DBSCAN necesita de dos parámetros: ε y *MinPts*.

```
1:  function DBSCAN(positions, eps, minPts)
2:      C = 0
3:      for each pos in positions do
4:          if pos has been visited then
5:              Continue next position
6:          else
7:              Mark pos as visited
8:              N(pos) = NeighborPts(pos, eps)
9:              if length(N(pos)) < MinPts then
10:                 Mark pos as noise
11:             else
12:                 C = next Cluster
13:                 expandCluster(pos, N(pos), C, eps, MinPts)
14:             end if
15:         end if
16:     end for
```

EXPERIMENTO CON DBSCAN

Elegimos un $\varepsilon = 0{,}0001$ y $minPts = 5$ ya que nos proporcionaría una consolidación de aprox. 20 %.



EXPERIMENTO CON DBSCAN

Resultados:

- 9 clústers
- Muchos puntos marcados como ruido

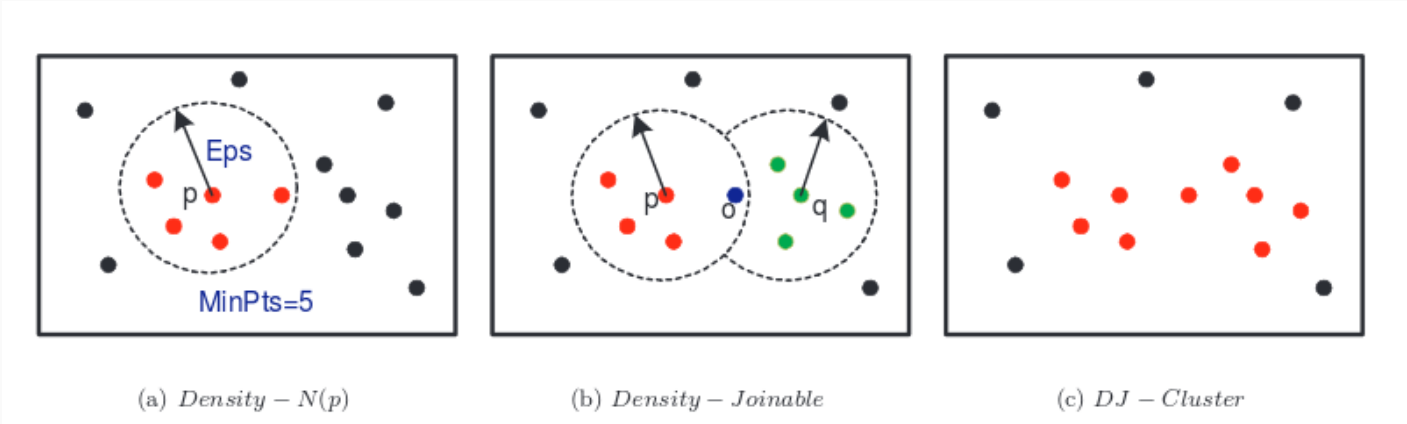
Inconvenientes:

- Decididir si esos puntos marcados como ruido son eliminables o no.

DJ-CLUSTER

Density-Joinable Clúster es un tipo de algoritmo de clustering basado en densidades de puntos.

1. Para cada punto, calculamos su vecindario fijado un ε.
2. Si el número de puntos del vecindario es menor que *minPts*, se deshecha.
3. Si el número de puntos es mayor que *minPts*, se crea un clúster.
4. Se comprueba si existen clústers densamente acoplables (si existen otros clústers con algún punto en común)



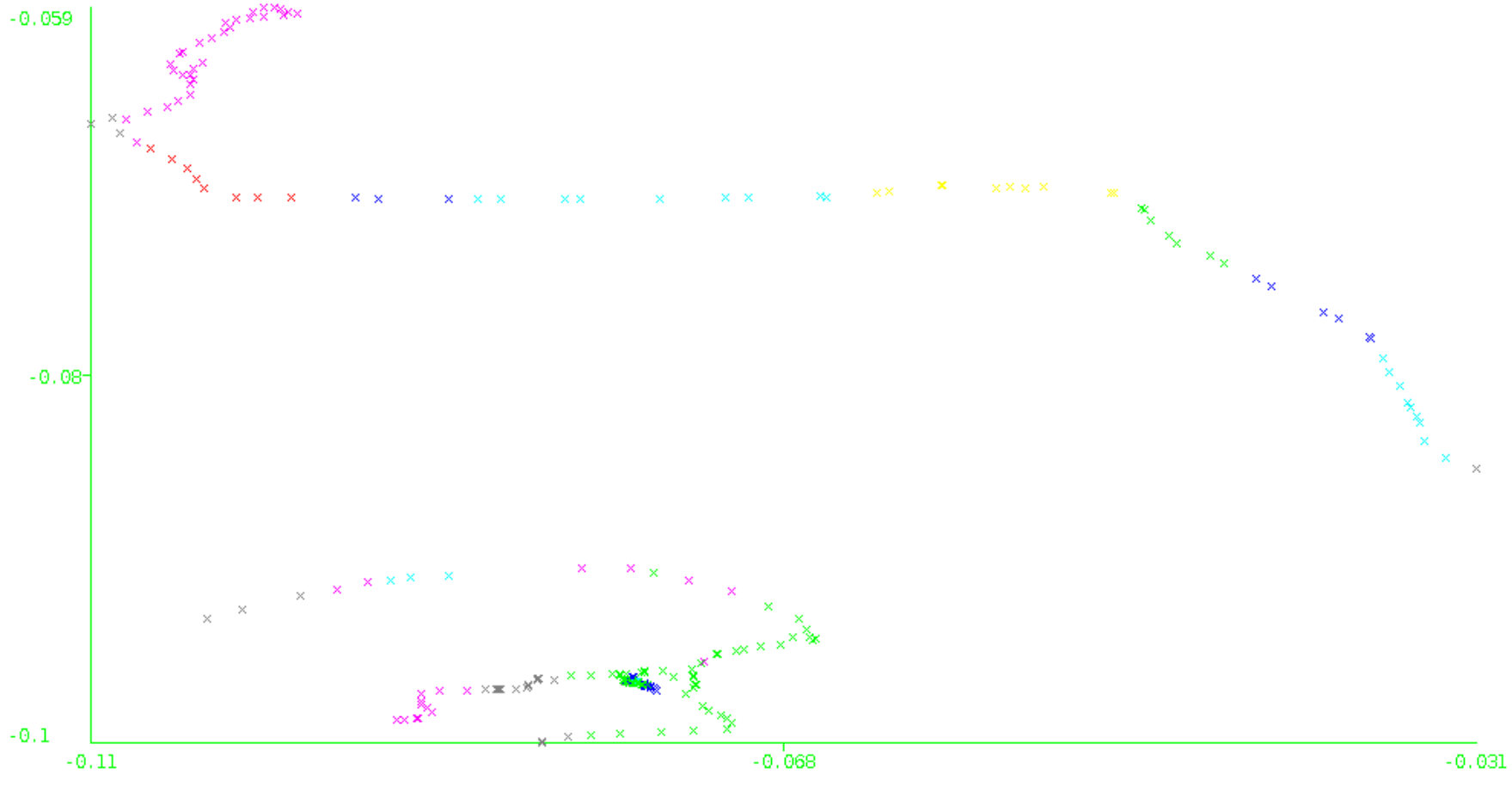
(a) $Density - N(p)$ (b) $Density - Joinable$ (c) $DJ - Cluster$

DJ-CLUSTER

1: **for each** p in set S **do**
2: Compute neighborhood $N(p)$ for ε and *MinPts*
3: **if** $N(p)$ is null ($|N(p)| < MinPts$ for ε) **then**
4: Label p as noise
5: **else if** $N(p)$ is density-joinable to an existing cluster **then**
6: Merge $N(p)$ with the cluster which is density-joinable
7: **else**
8: Create a new cluster C based on $N(p)$
9: **end if**
10: **end for**

EXPERIMENTO CON DJ-CLUSTER

Utilizando **Weka** dado un $\varepsilon = 0{,}0001$ y $minPts = 2$:



EXPERIMENTO CON DJ-CLUSTER

22 clústers en total:

N. Clúster	Cantidad puntos
0	3
12	605
58	7
142	449
....	...

COMPARATIVA TÉCNICAS SIMPLES

Se ha realizado estos experimentos sobre el mismo sujeto tomando una muestra de las primeras 2000 posiciones en el tiempo.

Método	Tiempo	N.
Cons. por adelgazamiento	<0.01 sec	800
Cons. por distancia simple	<0.01 sec	507
Cons. por distancia t_0-alcanzable	<0.01 sec	21
Cons. por tiempo	<0.01 sec	1786

COMPARATIVA TÉCNICAS CLUSTERING

Método	Tiempo	Clústers	Iteraciones
K-means	0.69 secs	500	9
DBSCAN	2 min 30 secs	9	111
DJ-Cluster	0.37 secs	22	11

- DBSCAN más lento de todos.
- DBSCAN consolidación mayor.
- K-means se queda en 500 clústers.
- DJ-Clúster baja de los 500.
- DJ-Clúster menor tiempo de ejecución.

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.

Algoritmos de clustering:

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.
2. Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

Algoritmos de clustering:

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.
2. Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

Algoritmos de clustering:

1. Más avanzados, pero más complejos a la hora de implementar.

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.
2. Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

Algoritmos de clustering:

1. Más avanzados, pero más complejos a la hora de implementar.
2. Importante un procesado previo.

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.
2. Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

Algoritmos de clustering:

1. Más avanzados, pero más complejos a la hora de implementar.
2. Importante un procesado previo.
3. Mejores a la hora de recuperar una traza con los datos borrados.

Algoritmos de consolidación simple:

1. Son *simples*, pero eficaces.
2. Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

Algoritmos de clustering:

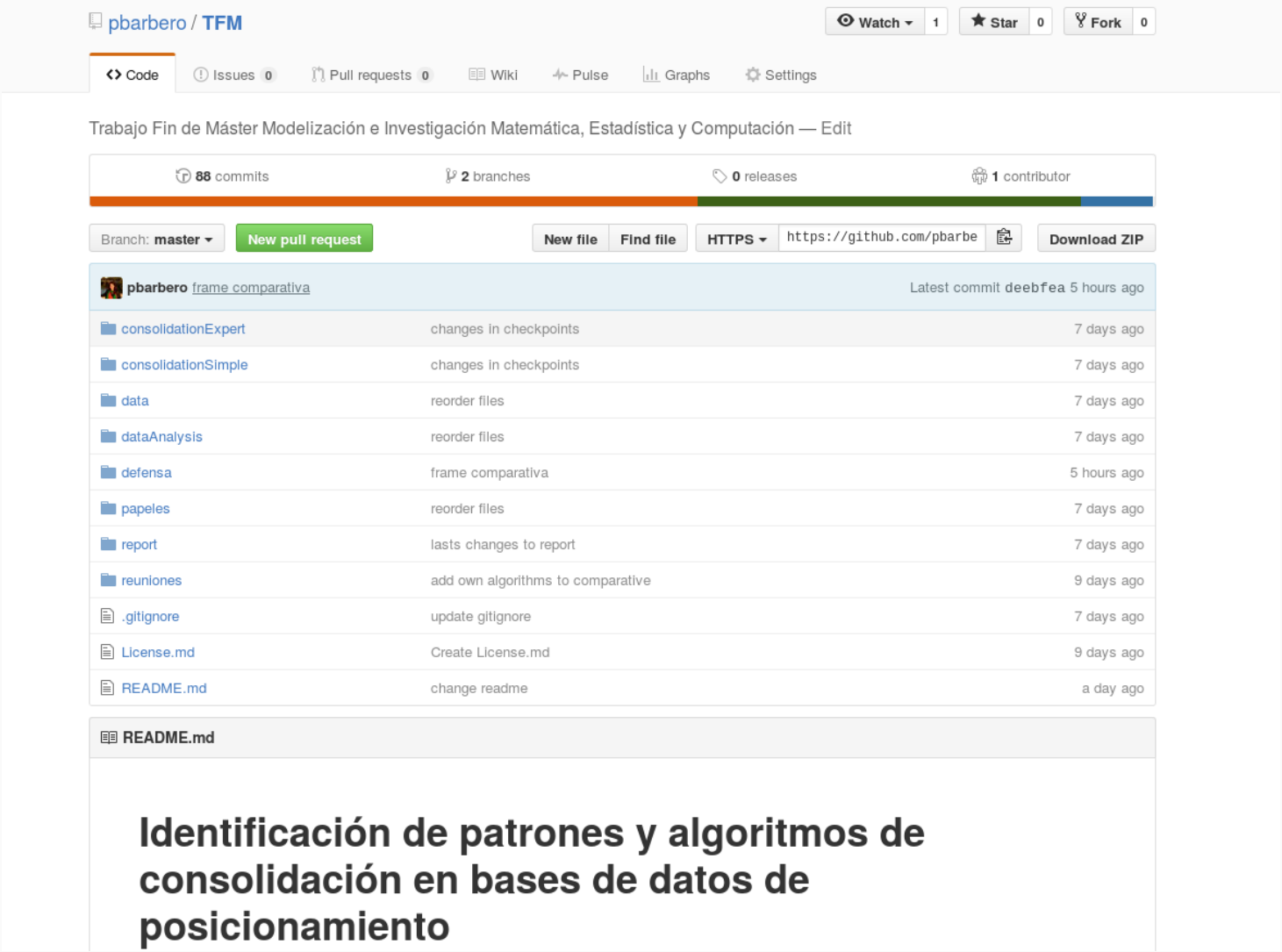
1. Más avanzados, pero más complejos a la hora de implementar.
2. Importante un procesado previo.
3. Mejores a la hora de recuperar una traza con los datos borrados.
4. Nociones de ruido implementadas.

DEMOSTRACIÓN

IPython Notebook

CÓDIGO

http://github.com/pbarbero/TFM



REFERENCIAS

- Frank E. and Witten I.H. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco, 2nd edition, 2005.
- Terveen L. Frankowski D., Ludford P. Shekhar S. and Zhou C. Discovering personal gazetteers: An interactive clustering approach. In In Proc. ACMGIS, pages 266–273. ACM Press, 2004.
- Nigam K. McCallum A. and Ungar L. H. Efficient clustering of high dimensional data sets with application to reference matching, 2000.
- Kafle S. Implementation of dbscan algorithm in python. https://github.com/SushantKafle/DBSCAN, 2014.