Trajectory Clustering Alghorithms - Two scalable and Comovement-based approaches.

Federico Naldini

Alma Mater Studiorum - Università di Bologna, Cesena.

federico.naldini3@studio.unibo.it

15/10/2019

Overview

Che cosa sono i *Comovements patterns*?

I *Comovements patterns* sono raggruppamenti di oggetti che hanno viaggiato assieme per un certo periodo di tempo.
L'interesse per questi insieme può essere basato su diversi fattori, come ad esempio il numero degli elementi, la durata del viaggio, la loro effettiva vicinanza e il criterio utilizzato per calcolarla.
A seconda delle differenti caratteristiche utilizzate nel definire i raggruppamenti, è possibile definire delle tipologie di raggruppamenti.

Tipologie di *Co-movements pattern*

Le tipologie di *Co-movements pattern* possono essere divise sulla base di diversi fattori; tra tutti spicca in particolare la metrica di similarità utilizzata. Possono essere impiegate due misure:

- **Similarità basata sulla densità**
- **Similarità basata sulla distanza**

Similarità basata sulla densità

- **Convoy**: Identifica raggruppamenti di oggetti che hanno percorso traiettorie simili per almeno T istanti consecutivi, negli algoritmi classici prima viene applicato un algoritmo di *clustering* basato sulla densità e successivamente viene effettuato *pruning* sulla base del criterio temporale sopraespresso.
- **Swarm**: Rispetto a **Convoy** scarta il vincolo di sequenzialità degli istanti temporali, basandosi solo sui vincoli spaziali.
- **Platoon**:Rimuove il vincolo descritto da **Convoy**, sostituendolo con un vincolo locale sugli istanti consecutivi tramite un parametro L che identifica la lunghezza minima di ogni sottosequenza di istanti consecutivi.
  Inoltre aggiunge un altro parametro K, che identifica la lunghezza minima della sequenza temporale T di ogni raggruppamento.

Similarità basata sulla distanza

- **Flock**: La stessa idea presentata in **Convoy**, ma utilizzando una metrica di similarità basata sulla definizione di uno spazio chiamato *disk* di raggio *r*.
- **Group**:Rilassa il vincolo temporale introdotto da **Flock** introducendo un vincolo sulla lunghezza delle sequenze di istanti consecutivi, analogamente a quanto fatto in *Platoon*; tuttavia a differenza di quest'ultimo non introduce un vincolo sulla lunghezza totale della sequenza.

General Co-movement Patter Mining

Il framework **GCMP** si pone come obiettivo di mettere a disposizione degli utilizzatori una piattaforma configurabile per realizzare tutte le tipologie di *Co-movement mining* sopradescritte.
In particolare definisce diversi parametri per la definizione del problema:

General Co-movement Patter Mining: Parametri 1

- **M**: Numero minimo di elementi presenti in un raggruppamento per considerarlo interessante.
- **K**: Numero minimo di istanti temporali in cui un certo raggruppamento esiste.

General Co-movement Patter Mining: Parametri 2

- **L**: Data una sequenza temporale di istanti T, si identificano z sottosequenze tali che ogni sottosequenza è composta da istanti consecutivi(ad esempio con $T = (1,2,3,5,6)$ si ottengono due sottosequenze $T' = (1,2,3)$ e $T'' = (5,6)$); **L** identifica la lunghezza minima accettabile di tutte le z sottosequenze così individuate(ad esempio con **L** $= 3$ la sequenza non rispetta il vincolo, mentre con **L** $= 2$ sì).
  Una sequenza che rispetta il vincolo sopradescritto viene definita *L-consecutive*
- **G**: Data una sequenza temporale di istanti T, **G** identifica il massimo *skew* accettabile tra un elemento della sequenza e il successivo. Una sequenza si definisce *G-connected* se per ogni coppia di elementi consecutivi al suo interno lo *skew* in questione è minore o uguale a **G**

General Co-movement Patter Mining: Definizione

Un **GCPM** trova un set di oggetti O, rimasti assieme per una sequenza di istanti T che soddisfa i seguenti vincoli:

- *Closeness*: per ogni istante di T, gli oggetti O devono appartenere allo stesso *cluster*.
- *Significance*: La dimensione del raggruppamento deve essere maggiore di **M**.
- *Duration*: La dimensione di T deve essere maggiore di **K**.
- *Consecutiveness*: T è *L-consecutive*.
- *Connection*: T è *G-connected*.

General Co-movement Patter Mining: Parametri 3

Configurando i vari parametri, posso ottenere le cinque tipologie di *Co-movements*

Pattern	M	K	L	G	**Clustering**
Group	2	1	2	$\|\mathbb{T}\|$	disk
Flock	·	·	K	1	disk
Convoy	·	·	K	1	density
Swarm	·	·	1	$\|\mathbb{T}\|$	density
Platoon	·	·	·	$\|\mathbb{T}\|$	density

Overview

L'algoritmo di risoluzione proposto dal framework *GCPM* si compone di tre fasi:

- **Generazione degli snapshot**
- **Star Partitioning**
- **Apriori Enumeration**

Il risultato final dell'algoritmo consiste in una serie di tuple nella forma $<O,T>$, dove O è un insieme di oggetti identificati da un ID univoco, mentre T rappresenta la sequenza temporale per cui l'insieme di oggetti O ha viaggiato in luoghi vicini.

Preprocessing dei dati

Unica operazione da eseguire per il preprocessing è l'unificazione di tutti i dati temporali alla stessa scala di misura: è importante che fra le misure temporali delle varie traiettorie sia definita una scala e dei criteri di misura che permettano di determinare se un istante è venuto o meno prima di un altro, se due istanti sono consecutivi.

Generazione degli snapshot 1

Il primo passo dell'algoritmo consiste nella generazione degli *Snapshots*, che altro non sono che raggruppamenti di oggetti in uno specifico istante temporale t

Generazione degli snapshot 2

Ogni traiettoria associata ad un oggetto viene scomposta nell' insieme di punti che la compongono P, ciascun punto p viene poi aggregato in un nuovo insieme, chiamato P' sulla base dell'attributo $p.t$, ovvero la sua coordinata temporale.

Generazione degli snapshot 3

Su ogni *P'* così individuato viene applicato un algoritmo di Clustering(*DBSCAN* nel caso si necessiti di un algoritmo *density based* , *disk-based clustering* qualora si voglia utilizzare una metrica basata sulla distanza). I clustering di oggetti così individuati sono raggruppati in un set e vengono associati all'istante temporale t che identifica *P'*.
Gli *Snapshots* sono dunque nella forma $<t, S(t)>$, dove t è l'istante temporale mentre $S(t)$ è il set di clusters associato

G_t e G_A

Dato uno *snapshot* s_i definiamo il grafo G_{ti} come un grafo dove i nodi sono tutti gli oggetti del dataset e due nodi sono connessi da un arco se e solo se appartengono a uno stesso cluster. Definiamo G_A, chiamato anche *Associated Graph*, come l'insieme di tutti i G_i individuati dagli *snapshot*: L'insieme dei nodi sarà ancora una volta l'insieme degli oggetti e due nodi saranno connessi se compaiono all'interno di uno stesso cluster di un qualunque *snapshot*. Essendo possibile che due oggetti compaiano in più *snapshot* all'interno di uno stesso cluster, sugli archi di questo grafo verrà mantenuta traccia degli istanti temporali in cui due oggetti compaiono nello stesso cluster.

Directed star

G_A rappresenta il punto di partenza per catturare le relazioni tra coppie di oggetti, tuttavia non essendo G_A un grafo orientato, occorre una tecnica per attraversare il grafo in maniera efficace e evitare di individuare coppie di nodi già visitate.

Directed Star realizza tale struttura in due passi:

1. viene realizzato un ordinamento globale degli oggetti.
2. si scorre G_A partendo dal nodo con Global ID=1, per ciascun nodo N si prendono in considerazione tutti i suoi vicini(ovvero connessi con un arco) con Global ID>N.Global ID, vengono poi generate tutte le coppie $<N, V_i: T(i)>$dove N è il nodo esaminato, V_i è l'i-esimo nodo vicino e $T(i)$ è la sequenza di istanti temporali contenuta nell'arco tra N e V_i

Ordinamento globale

Data la natura scalabile dell'algoritmo di generazione delle coppie sopracitato, è fondamentale che nel criterio di ordinamento e numerazione si scelga un criterio che permetta ad ogni nodo di avere circa lo stesso numero di vicini.
Dati n oggetti, ci sono $n!$ possibili ordinamenti.
Utilizzare un criterio di ordinamento random, come ad esempio ordinando sugli IDs degli oggetti da performance tutto sommato accettabili.

Definizione di monotonicità

Dato un *GCMP* con un objectset $\{o_1, ..., o_n\}$, la fase di **Star Partitioning** produce coppie nella forma $<o_i, o_j.\ T>$. In linea teorica dovrebbe essere possibile applicare il principio *Apriori* nella generazione dei candidati, tuttavia è facile dimostrare che la proprietà della monotonicità non vale per queste tuple.

Monotonicità classica

Date due sequenze:
$<o_i, o_j.\ 1,2,3,6>$
$<o_i, o_z.\ 1,2,3,7>$
Con $\mathbf{L} = 2$, $\mathbf{K} = 3$ e $\mathbf{G} = 2$
E' intuitivo vedere che che entrambe le sequenze non risultano *L-Consecutive*.
Al contrario, il superset ottenuto fondendo le due sequenze:
$<o_i,o_j\ ,o_z.\ 1,2,3>$
Risulta *L-Consecutive*.
E' quindi dimostrato che occorre trovare una nuova definizione di monotonicità.

Tre nuovi concetti

Per giungere a una nuova definizione di monotonicità serve introdurre tre nuovi concetti:

- **Maximal G-Connected sequence**
- **Decomposable sequence**
- **Sequence semplification**

Maximal G-connected sequence

T' è definibile come *MGS* di *T* se *T'* è sottosequenza di *T* e non esiste *T"* tale che *T'* è sottosequenza di *T"* e *T"* è *G-connected*.
Ogni sequenza può avere *n* MGS di dimensioni differenti.
Ogni *MGS* gode delle seguenti proprietà:

1. Ogni *MGS* è disgiunta dalle altre se non per un elemento in comune.
2. Data una sequenza, l'unione di tutte le sue *MGS* dà come risultato la sequenza stessa.
3. Date due sequenze T_1 e T_2 dove T_1 è sotto-sequenza di T_2, allora per ogni *MGS* di T_1 si può trovare una *MGS* di T_2 tale che la *MGS* di T_1 sia sottosequenza della *MGS* di T_2

Sequence simplification

Data una sequenza T, questa viene definita come decomponibile se ogni sua *MGS* è *L-consecutive* e lunga almeno **K** elementi.

Decomposable sequence

Data una sequenza T, viene prodotta una nuova sequenza T' tramite una funzione *Sim* che si compone di due passaggi:

1. **f-step**: rimuove tutti i segmenti di T che non sono *L-consecutive*.
2. **g-step** tra le *MGS* ricavate dalla sequenza uscita dal passo precedente, rimuove quelle con dimensioni minori di **K**.

Nuova monotonicità

Dato un canditato $C = \{O{:}T\}$ dove O è una sequenza di oggetti e T una sequenza temporale, se $Sim(T) = \emptyset$ allora il candidato può essere eliminato e così tutte le sue soprasequenze.

Algoritmo aPriori

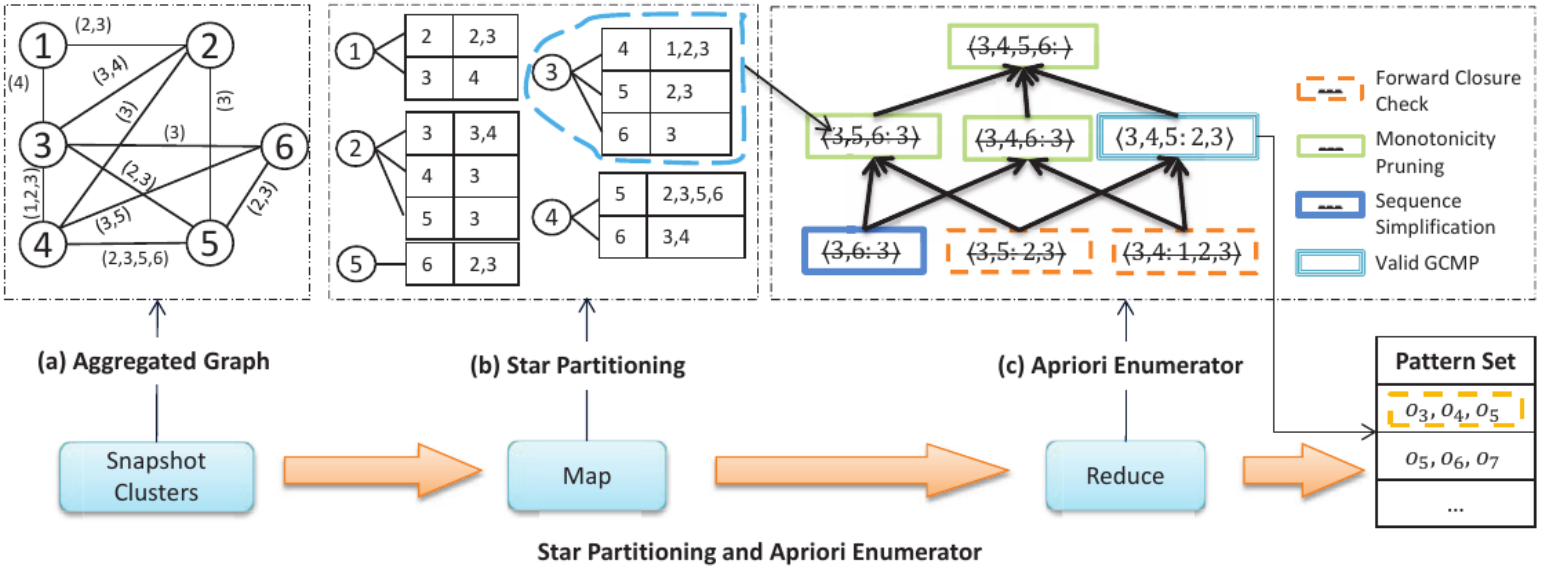
1. Ogni coppia va a generare il livello base della struttura
2. Viene fatto pruning sulle coppie basandosi sulla funzione di semplificazione *Sim*
3. Tramite un indicatore di livello, vengono generati dei candidati di dimensione del livello, il join tra due candidati del livello precedente C_1 e C_2 viene effettuato unendo gli object-set e intersecando le sequenze temporali.
4. Per ogni nuovo candidato viene calcolata la semplificazione e viene fatto pruning di conseguenza.
   Candidati che per uno dei vincoli parametrici non possono generare ulteriori candidati sono scartati
5. Per migliorare le performance, si utilizza il principio della *forward closure*: ogni pattern valido nei termini dei parametri espressi viene mandato in output.

Output

Un set di candidati nella seguente forma:

$$C = \{O{:}T\}$$

Flow del framework



Star Partitioning and Apriori Enumerator

Distribuited Subtrajectory Clustering overview

Il problema del *Distribuited Subtrajectory Clustering* viene suddiviso in tre passi:

- **Subtrajectory Join**
- **Trajectory Segmentation**
- **Clustering and Outlier detection**

In letteratura ci sono tante possibili soluzioni per affrontare il problema, ma nessuna di queste utilizza un approccio distribuito.

Formulazione del problema

Dato D come un set di traiettoria, si identifica r come una traiettoria composta da *timestamp location*, ovvero dati nella forma $<(loc_i,\ t_i)>$, lunga n.
Data una traiettoria r si definisce sotto-traiettoria l'insieme dei punti $\{r_i,..,r_j\}$ con $i > 1$ e $j < n$

Similarità tra le traiettorie

Per riuscire a ottenere una metrica resistente a traiettorie di lunghezza diversa, con scale temporali diverse e mancanza di coordinamento, si è scelto di utilizzare una variante pesata di *LCSS* con alcuni parametri collegati:

- ϵ_t: specifica il range temporale entro cui due punti sono considerati vicini.
- ϵ_{sp}: specifica il range spaziale entro cui due punti sono considerati vicini.

Subtrajectory Join: idea

Primo dei tre passi dell'algortimo, per ogni traiettoria $r \in D$ si cercano tutti gli oggetti con relative porzioni che si sono mossi in uno spaziotempo vicino a quello della traiettoria.
Ciò viene ottenuto con un'operazione di *Self Join*. per ogni coppia di traiettorie r e s vengono ricercate e trovate tutte le sotto-traiettorie comuni con una certa durata, candidate alla *Long Common Sub Sequence*
.
Definendo formalmente: Date una tolleranza spaziale ϵ_{sp} e una temporale ϵ_{t} e un limite inferiore alla durata temporale δ_{t}, si recuperano tutte le coppie $(r', s') \in$ D tali che la durata di entrambe le sottosequenze è maggiore della soglia δ_{t} e $\forall\ r_i \in r'\ \exists\ s_j \in s'$ tale che $d_s(r_i, s_j)$ e $d_t(r_i, s_j)$

Subtrajectory Join: implementazione

Per quanto riguarda l'implementazione del **Subtrajectory Join** si usa *DJT*, che si compone di due fasi:

- **Join**: fase in cui dati i punti in un certo intervallo temporale viene calcolata la vicinanza con ogni altro punto dello stesso intrevallo.
- **Refine**: fase in cui le traiettorie vengono riunificate portandosi dietro il loro vicinato.

Come preprocessing viene fatta un operazione di *Repartitioning*, in cui le traiettorie sono scomposte nei loro punti e questi sono divisi in partizioni con stessa densità sulla base del tempo.

Trajectory Segmentation: idea

Secondo dei tre passi dell'algoritmo, dato in ingresso una traiettoria e il suo vicinato, produce uno split in m sottotraiettorie sulla base di cambiamenti nella composizione del vicinato

Trajectory Segmentation: implementazione

Vengono individuati due possibili metodi per la ricerca di cambiamenti nel vicinato:

- **TSA$_1$**: individua le sotto-traiettorie mediante un cambiamento di densità nel vicinato.
- **TSA$_2$**: individua le sotto-traiettorie sulla base di un cambiamento nella composizione del vicinato.

A prescindere dal criterio utilizzato, l'algoritmo si basa su due *Sliding windows* che scorrono sulla traiettoria con un certo scarto temporale. Queste finestre sono caratterizzate dal parametro ω che individua la dimensione della finestra in termini di numero di elementi e τ che invece identifica la soglia di segmentazione per una sottotraiettoria.

Subtrajectory Clustering: idea

Terzo dei tre passi dell'algoritmo, prende in ingresso le sotto-traiettorie individuate al passo precedente e le clusterizza in un insieme di cluster separati.
Il meccanismo di clustering prevede l'elezione di un rappresentante per cluster, chiamato R_i, che viene utilizzato come punto di riferimento per l'assegnazione di una nuova sottotraiettoria a un cluster.
L'obiettivo di questa fase è di minimizzare la somma delle distanze fra il rappresentante di ogni cluster e gli elementi che ne fanno parte

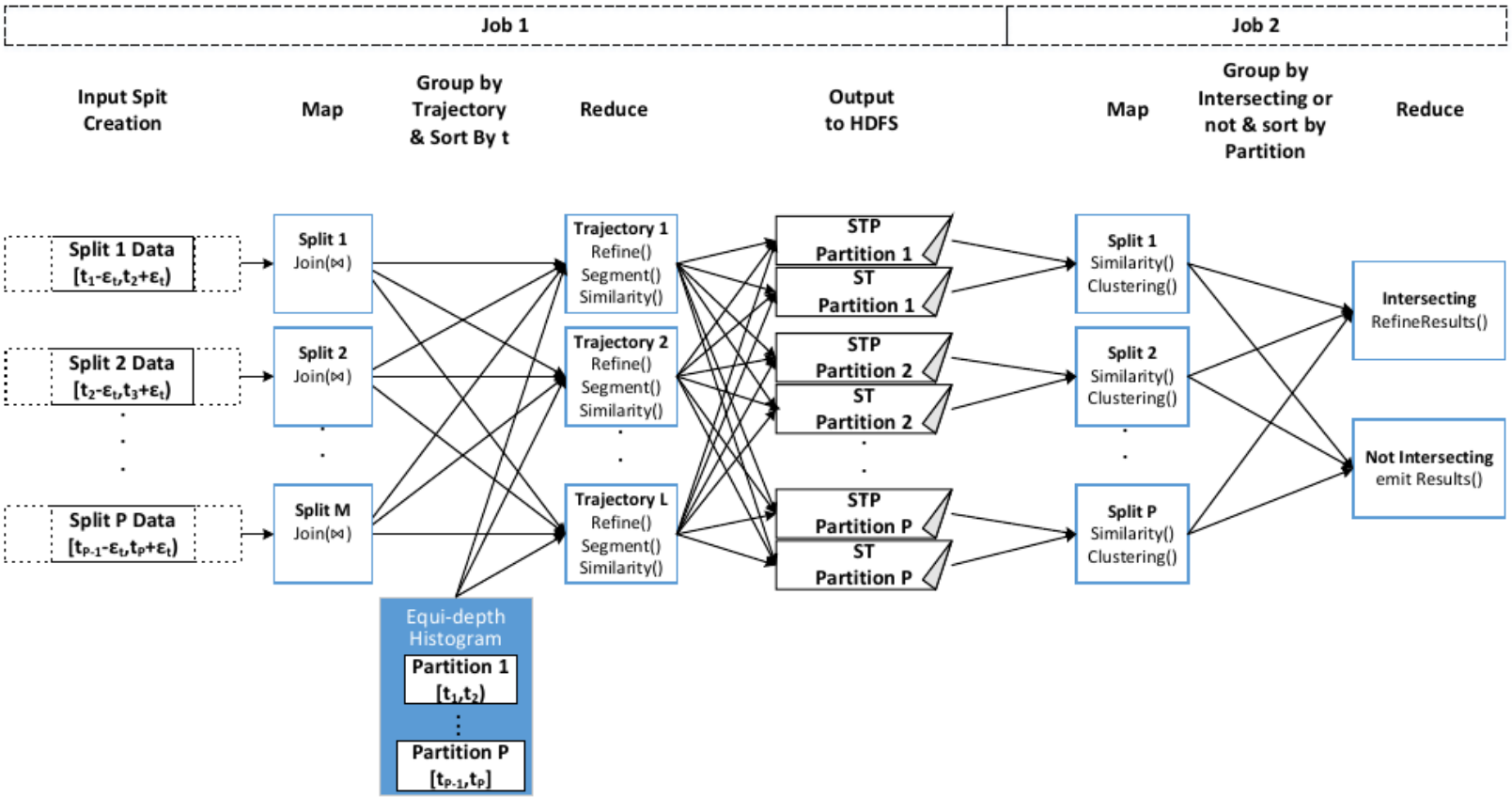
Subtrajectory Clustering: implementazione

L'input per questa fase è una trasformazione di quello emesso al termine della fase due, occorre infatti determinare la similarità tra ogni sottotraiettoria prima di procedere con le operazioni di clustering.
Sono aggiunti inoltre due parametri: α che rappresenta la distanza massima a cui una sottoraiettoria può essere da un rappresentante del cluster e K che è un *lower bound* sul voting(espresso nella fase due) per l'elezione dei rappresentanti. Alla fase di clustering segue poi una fase di raffinamento dei risultati, dove vengono eliminate sotto-traiettorie ripetute, si uniscono cluster simili e eventualmente si riassegnano gli outlier.

Output

Come output il framework presenta una serie di cluster di sotto-traiettorie e un insieme di outlier.

DSC flow


Job 1
Job 2
Input Spit Creation
Map
Group by Trajectory & Sort By t
Reduce
Output to HDFS
Map
Group by Intersecting or not & sort by Partition
Reduce
Split 1 Data [t1-εt,t2+εt)
Split 2 Data [t2-εt,t3+εt)
Split P Data [tp-1-εt,tp+εt)
Split 1 Join(⋈)
Split 2 Join(⋈)
Split M Join(⋈)
Trajectory 1 Refine() Segment() Similarity()
Trajectory 2 Refine() Segment() Similarity()
Trajectory L Refine() Segment() Similarity()
Equi-depth Histogram
Partition 1 [t1,t2)
Partition P [tp-1,tp]
STP Partition 1
ST Partition 1
STP Partition 2
ST Partition 2
STP Partition P
ST Partition P
Split 1 Similarity() Clustering()
Split 2 Similarity() Clustering()
Split P Similarity() Clustering()
Intersecting RefineResults()
Not Intersecting emit Results()

Conclusioni

Entrambi gli approcci presentati offrono un framework compatibile con le moderne tecnologie in ambito *Big Data*, entrambe propongono soluzioni a un problema simile, tuttavia le idee e implementazioni dei due framework sono totalmente differenti tra loro

Trajectory Clustering Alghorithms - Two scalable and Comovement-based approaches.

Federico Naldini

Alma Mater Studiorum - Università di Bologna, Cesena.

federico.naldini3@studio.unibo.it

15/10/2019