Progetto di Algoritmi Euristici

K-means e K-means++

Michele Valsesia

Anno accademico 2017/2018

Introduzione

Il processo di clustering suddivide un insieme di entità, generalmente corrispondenti a punti di uno spazio metrico, in gruppi omogenei e ben separati. Un gruppo è omogeneo se è costituito da entità con caratteristiche simili tra loro ed è ben separato quando le sue entità non sono contenute in altri gruppi. I gruppi sono chiamati *cluster* ed il risultato della suddivisione identifica una *partizione* dello spazio delle entità.

Definito uno spazio Euclideo $\mathbb{R}^{\mathrm{d}}$, un vettore $X = [x_1, \ldots, x_N]$ con $x_i \in \mathbb{R}^{\mathrm{d}}$ ed un sottoinsieme C di X, si vuole trovare una partizione $P_K = C_1, \ldots, C_k$ di X composta da K cluster tale per cui:

- $C_k \neq \emptyset;\ k = 1, 2, \ldots, K$
- $C_i \cap C_j = \emptyset;\ i, j = 1, 2, \ldots, K;\ i \neq j$
- $\bigcup_{k=1}^{K} C_k = X$

La partizione di una serie di punti di uno spazio Euclideo viene calcolata per mezzo del metodo della *Minima Somma di Quadrati (MSCC)*. Questo metodo soddisfa simultaneamente sia la condizione di omogeneità che quella di non separabilità descritte dal processo di clustering. Ogni cluster è identificato da un punto c_i, chiamato *centroide*, che rappresenta il baricentro dei punti ad esso associati. Per ogni punto viene calcolato il quadrato della distanza euclidea rispetto a tutti i centroidi presenti. Un punto è infatti assegnato al cluster alla quale corrisponde la distanza minima.
L'obiettivo principale della Minima Somma di Quadrati consiste nell'individuare la partizione che minimizza la somma delle distanze euclidee al quadrato tra un punto ed il suo relativo centroide.
Da un punto di vista matematico, dato un insieme $X = [x_1, \ldots, x_i, \ldots, x_N]$ con $x = [x_{1i}, \ldots, x_{di}]$ di N punti in uno spazio Euclideo $\mathbb{R}^{\mathrm{d}}$, si vuole partizionare X in K sottoinsiemi disgiunti C_j, tali per cui la somma del quadrato delle distanze euclidee di ciascun punto x_i da un centroide c_j, associato ad un cluster C_j, sia minima. Sia P_k l'insieme di tutte le partizioni di X in K gruppi e sia la partizione P definita come $P = C_1, \ldots, C_k$, allora la funzione obiettivo che individua la migliore partizione di X è definita come:

$$f_{MSSC}(P) = \min_{p \in P_k} \sum_{i=1}^{N} \min_{j=1,\ldots,K} (\|x_i - c_j\|_2)^2 \tag{1}$$

La distanza euclidea al quadrato viene usata per evitare il calcolo della radice quadrata, costosa computazionalmente, e non danneggia la struttura generale del problema, in quanto viene considerata solo per operazioni di confronto. Infatti:

$$(\|x_i - c_{j1}\|_2)^2 \geq (\|x_i - c_{j2}\|_2)^2 \Leftrightarrow (\|x_i - c_{j1}\|_2) \geq (\|x_i - c_{j2}\|_2) \tag{2}$$

Il valore di un centroide c_j è ottenuto dalla media aritmetica dei punti del cluster.

$$c_j = \frac{1}{|C_J|} \sum_{i \in C_j} x_i \tag{3}$$

K-means

L'euristica che implementa il metodo della Somma dei Minimi Quadrati è conosciuta come *K-means*. La soluzione di partenza dell'algoritmo viene generata estraendo uniformemente un numero K di centroidi casuali dagli N punti dell'istanza X ed assegnando i restanti punti al cluster che minimizza il quadrato della distanza euclidea tra il centroide ed il punto stesso. La procedura che consente di trovare una partizione dello spazio Euclideo si compone principalmente di tre fasi:

- L'uso dell'equazione 3 per calcolare i valori dei nuovi centroidi
- Il ricalcolo del quadrato della distanza euclidea, per ogni punto, rispetto a tutti i centroidi
- L'assegnamento di ciascun punto al cluster che minimizza la distanza euclidea al quadrato

Il processo sopradescritto termina o quando non avviene più nessun riassegnamento o quando viene superata una soglia massima di ripetibilità stabilita a priori. Nello pseudocodice sottostante, $m(x_i)$ è la funzione di *membership* che associa un punto x_i al cluster corrispondente.

Da un punto di vista pratico, il K-means viene utilizzato per raggruppare in classi gli oggetti presenti in un'immagine. Le classi sono costituite da elementi con caratteristiche simili tra loro, come ad esempio la forma, il colore e la posizione delle varie entità considerate. Partizionare un'immagine aiuta ad eliminare dati superflui, consentendo una riduzione del filesize, e ad estrarre le feature fondamentali per il confronto tra immagini. Il K-means è anche un algoritmo di apprendimento non supervisionato che viene applicato in diversi ambiti del machine learning.

K-means

1: **function** KM$(X, K, Maxit)$
2: Scegliere i centroidi iniziali $c_k = (k = 1, \ldots, K)$
3: **for** $i := 1, \ldots, N$ **do**
4: $m(x_i) \leftarrow \text{argmin}_{j=1,\ldots,K}(\|x_i - c_j\|_2)^2$
5: $l \leftarrow 0$
6: **while** *m varia* && $l < Maxit$ **do**
7: $l \leftarrow l + 1$
8: **for** $j := 1, \ldots, K$ **do**
9: *Calcola i centroidi* c_j
10: **for** $i := 1, \ldots, N$ **do**
11: $m(x_i) \leftarrow \text{argmin}_{j=1,\ldots,K}(\|x_i - c_j\|_2)^2$

K-means++

Il *K-means++* è stato creato per ridurre la dipendenza che si presenta tra la soluzione finale e i K centroidi casuali scelti inizialmente. Differisce dal K-means classico solo per il diverso modo di generare la soluzione di partenza. Il primo centroide viene scelto con probabilità uniforme tra gli N punti dell'istanza, mentre i restanti $K - 1$ vengono estratti da una distribuzione di probabilità pesata. Il peso di ciascuno dei punti corrisponde alla minima distanza euclidea al quadrato tra il punto ed i centroidi correntemente individuati.

VNS

La Variable Neighbourhood Search (VNS) è una metaeuristica di scambio usata per migliorare una soluzione prodotta da un'euristica di scambio. È basata su una gerarchia di intorni ad ampiezza crescente costruiti a partire da una sequenza di k operazioni, tratte da un insieme O di operazioni elementari. Stabilita una dimensione minima k_{min} dell'intorno, la VNS estrae in maniera casuale, dall'intorno corrente, la soluzione iniziale ed applica la stessa euristica di scambio usata per produrre la soluzione che si vuole migliorare.
La nuova funzione obiettivo viene confrontata con quella trovata precedentemente e se è minore, la dimensione dell'intorno viene riportata a k_{min}, la nuova funzione obiettivo viene salvata e la soluzione trovata diventa quella ottima. Nel caso in cui il valore della funzione obiettivo fosse maggiore, la dimensione dell'intorno viene aumentata di un determinato valore δ_k.
Il processo descritto sopra viene iterato fino a quando non si raggiunge la massima dimensione dell'intorno, stabilita a priori dal parametro k_{max}, oppure non viene superato un certo tempo

di esecuzione.

La metaeuristica si chiama Variable Neighbourhood perché l'intorno considerato varia a seconda dei risultati ottenuti dall'euristica di scambio:

- Se la soluzione migliora, la VNS riparte dall'intorno di minima dimensione. Viene così applicata la cosiddetta procedura di intensificazione.
- Se la soluzione non migliora, la VNS aumenta la dimensione dell'intorno. Viene così applicata la cosiddetta procedura di diversificazione.

Il metodo fa uso di tre parametri:

- k_{min} è la minima dimensione dell'intorno e viene generalmente posto a 1
- k_{max} è la massima dimensione dell'intorno e deve essere sufficientemente alto da garantire soluzioni molto diverse fra loro
- δ_k è la variazione del parametro k tra due intorni consecutivi ed è generalmente posto a 1

Nello pseudocodice sottostante, la funzione $SteepestDescent(x^{(0)})$ è l'euristica di scambio, mentre $ExtractNeighbour(x^*, k)$ è la procedura che estrae a caso una soluzione iniziale dall'intorno corrente.

Variable Neighbourhood Search

```
1:  function VNS(X, x^(0))
2:      x ← SteepestDescent(x^(0))
3:      x* ← x
4:      k ← k_min

5:      for i := 1, ..., ℓ do
6:          x' ← ExtractNeighbour(x*, k)
7:          x' ← SteepestDescent(x')

8:          if f(x') < f(x*) then
9:              x* ← x'
10:             k ← k_min
11:         else
12:             k ← k + δ_k

13:         if k > k_max then
14:             k ← k_max

15:     return (x*, f(x*))
```

Scelte Implementative

Il K-means implementato partiziona uno spazio Euclideo bidimensionale. I parametri forniti in input dall'utente sono 7:

- Il nome del file contenente gli N punti (entità) che l'algoritmo deve partizionare
- Il numero K di cluster richiesti
- Il numero massimo di iterazioni *Maxit* usate dal K-means
- La dimensione massima dell'intorno k_{max} usata dalla VNS
- Il tempo di esecuzione t della VNS
- Il seed da usare per generare i numeri casuali

Prima di eseguire il K-means, il processo verifica che i parametri inseriti siano corretti. Ad eccezione del nome del file contenente i punti da partizionare, che deve essere una stringa, i restanti parametri sono numeri interi.
Per evitare di avere un numero K di cluster pari o superiore al numero di punti dell'istanza, K deve essere sempre inferiore a N.
La dimensione massima dell'intorno k_{max} è sempre inferiore o uguale a K, altrimenti la VNS creerebbe delle soluzioni casuali con un numero di centroidi superiori a K.
Il parametro di seed è l'unico che può assumere un valore negativo in modo da specificare all'algoritmo l'uso di un seme non fissato.
L'utente non può inserire parametri ulteriori a quelli richiesti. Se anche solo una delle condizioni sopradescritte non viene rispettata, il processo termina restituendo un errore.

Il progetto è stato realizzato con una politica di lavoro che mira a ridurre il tempo di calcolo a sfavore del numero delle strutture dati. Ad esempio, per il calcolo delle distanze tra un punto x_i ed un centroide k, viene usato un array di double, invece per la funzione di membership $m(x_i)$ un array di interi.

Alla prima iterazione, la funzione obiettivo ed i centroidi vengono calcolati in tempo lineare, ma a partire dalle iterazioni successive vengono calcolati in tempo costante. Questi aggiornamenti si verificano quando un punto x_k viene assegnato ad un altro cluster. Dato un centroide c_j associato al cluster C_j di cardinalità n_j ed un centroide c_i associato al cluster C_i di cardinalità n_i, il nuovo valore dei centroidi è determinato dalle seguenti formule:

$$c_j \leftarrow \frac{n_j c_j - x_k}{n_j - 1} \qquad\qquad c_i \leftarrow \frac{n_i c_i + x_k}{n_i + 1} \tag{4}$$

Per aggiornare la funzione obiettivo si sottrae la distanza tra x_k ed il vecchio centroide e si aggiunge quella relativa al nuovo centroide.

$$f_{MSSC} = [f_{MSSC} - \min_{old=1,\dots,K}(\|x_k - c_{old}\|_2)^2] + \min_{new=1,\dots,K}(\|x_k - c_{new}\|_2)^2 \tag{5}$$

La generazione della soluzione casuale, descritta dalla VNS, viene implementata con il meccanismo di *Shaking*, che consiste nel sostituire k centroidi della soluzione corrente con punti dell'istanza estratti in maniera casuale.

Meccanismo di Shaking

```
1: function SHAKING(X, C, k)
2:     j ← 0

3:     while j < k do
4:         r ← rand()
5:         r1 ← ⌊(K − j + 1) ∗ r⌋
6:         r2 ← ⌊(N − j + 1) ∗ r⌋

7:         c(r1) ← x(r2)

8:         j ← j + 1
```

La soluzione finale viene salvata in un file *EPS*. Il formato EPS permette di rappresentare graficamente le varie partizioni ed i relativi centroidi. Ad ogni cluster è associato un colore diverso ed i centroidi sono identificati da rombi di colore nero.

Complessità nel caso pessimo

Per ognuno degli algoritmi, si analizza la corrispettiva complessità nel caso pessimo.

La valutazione tiene conto solo delle operazioni più costose temporalmente ed è composta da due fasi. Nella prima fase si analizza la complessità dell'euristica di scambio, mentre nella seconda quella della metauristica.

K-means e K-means++

Estrarre i centroidi casuali della soluzione di partenza del K-means richiede K passi. Il K-means++, per ciascuno dei $K-1$ centroidi iniziali, scorre gli N punti dell'istanza e calcola i relativi pesi della funzione di probabilità associata. Il calcolo della funzione obiettivo e della media, alla prima iterazione, ha un costo pari a N. Per ognuno degli N punti vengono calcolate le K distanze euclidee al quadrato. L'intero procedimento viene eseguito per un massimo numero di iterazioni pari a $Maxit$ e quindi il costo complessivo totale del K-means e del K-means++ nel caso pessimo è $\mathcal{O}(MaxitNK)$.

VNS

L'operazione di shaking richiede un numero di passi pari alla massima dimensione dell'intorno k_{max}. L'euristica di base viene ripetuta per k_{max} volte e per un certo tempo t stabilito a priori dall'utente. Nel caso pessimo, il tempo massimo della metauristica è $\mathcal{O}(tk_{max}NKMaxit)$.

Considerazioni finali

Generalmente il tempo di esecuzione t ha un valore trascurabile rispetto a quello degli altri parametri, perciò non influenza notevolmente la complessità. Il valore k_{max} non può superare il numero di cluster K, tuttavia l'algoritmo implementato prevede che K possa assumere il valore $N-1$. In sintesi, fissato $K = N-1$, la complessità massima ottenibile per mezzo di questo algoritmo è $\mathcal{O}(N^3)$.

Analisi dei Risultati

Computer. Tutti i test sono stati effettuati su un terminale con processore Intel(R) Xeon(R) CPU E5-1620 da 3.1 GHz e 16GB di RAM.

Istanze. Per questo progetto si sono utilizzate 10 istanze di TSP relative al Drilling Problem e contenute nel dataset *TSPLIB*. Le istanze sono costituite da un numero I crescente di punti bidimensionali compreso tra un minimo di 198 ed un massimo di 3795.

Test. Ogni test fornisce in output un file testuale contenente le funzioni obiettivo trovate dal K-means e dal K-means++ per ogni istanza, a partire da parametri di input fissati.

Parametri. Il numero di centroidi K è fissato a 10, mentre il parametro t viene fissato a 30 secondi per consentire all'algoritmo di avere un ampio margine di tempo per poter produrre i risultati. I numeri casuali vengono generati con un valore di seed pari a 1, in questo modo si garantisce la riproducibilità dei test siccome le soluzioni trovate sono le stesse per ogni esecuzione successiva. I parametri soggetti a variazione sono il massimo numero di iterazioni *Maxit*, del K-means e del K-means++, e la massima grandezza dell'intorno k_{max} della VNS.

Per valutare la differente qualità dei risultati prodotti dagli algoritmi sulle istanze di input, si è optato di far assumere al parametro k_{max}, che rappresenta la massima grandezza dell'intorno della VNS, i valori 5, 8 e 10. Questa scelta è stata fatta per evitare di avere delle soluzioni troppo dipendenti dal meccanismo di intensificazione e di diversificazione degli algoritmi. Per ognuno dei valori di k_{max}, vengono fatte tre prove con *Maxit* pari a 10, 30 e 50 e lasciando i restanti parametri invariati. Si è scelto di usare un modesto numero di iterazioni *Maxit*, per evitare di aumentare in maniera esagerata la complessità computazionale.

Test di Wilcoxon. Per individuare l'algoritmo che forinisce il miglior partizionamento dello spazio bidimensionale di partenza, le funzioni obiettivo dei test, calcolate da ciascuno degli algoritmi a partire dalle stesse istanze di input, vengono fornite in ingresso ad uno script Perl che esegue il test di Wilcoxon. Il test di Wilcoxon è un analisi statistica che confronta quantitativamente due algoritmi e decreta quale è il migliore.

Risultati Ottenuti.

Tabella 1: Funzioni obiettivo con $k_{max} = 5$

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 921 277.326 987	1493.25
657	66 525 223.475 457	66 525 223.475 457	0
724	52 143 135.658 591	52 143 134.854 769	0.803 822
1655	141 861 655.971 517	141 861 656.100 074	0.128 557
1817	132 745 376.497 139	132 745 375.957 834	0.539 305
2152	158 791 141.767 284	158 791 141.771 142	0.003 858
2319	959 657 247.952 909	959 657 282.003 293	34.0504
3038	560 251 191.281 980	560 251 191.725 052	0.443 072
3795	106 393 708.697 129	106 393 708.697 129	0

(a) *Maxit* = 10

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 887 451.730 710	35 318.8
657	66 525 223.475 457	66 525 223.475 457	0
724	52 143 134.854 769	52 143 135.835 309	0.980 54
1655	141 861 655.971 517	141 861 655.971 518	$1.013\,28 \times 10^{-6}$
1817	132 745 376.207 258	132 745 375.976 270	0.230 988
2152	158 791 144.163 191	158 791 144.989 080	0.825 889
2319	959 628 248.694 286	959 653 175.730 093	24 927
3038	560 251 192.259 474	560 251 192.159 630	0.099 844 1
3795	106 393 708.697 129	106 393 708.697 129	0

(b) *Maxit* = 30

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 887 451.730 710	35 318.8
657	66 525 223.475 457	66 525 223.475 457	0
724	52 143 136.206 421	52 143 135.835 309	0.371 112
1655	141 861 655.971 517	141 861 656.591 375	0.619 858
1817	132 745 375.957 834	132 745 375.976 270	0.018 436
2152	158 791 141.973 905	158 791 144.989 080	3.015 18
2319	959 604 437.287 821	959 650 144.102 812	45 706.8
3038	560 251 194.955 438	560 251 192.247 621	2.707 82
3795	106 393 708.697 129	106 393 708.697 129	0

(c) $Maxit = 50$

Tabella 2: Funzioni obiettivo con $k_{max} = 8$

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 921 277.326 987	1493.25
657	66 526 822.704 837	66 525 223.746 162	1598.96
724	52 143 135.590 922	52 143 134.854 769	0.736 153
1655	141 861 655.971 517	141 861 655.971 517	0
1817	132 745 376.644 507	132 745 376.342 709	0.301 798
2152	158 791 141.801 942	158 791 141.767 284	0.034 658
2319	959 618 043.302 090	959 657 274.615 160	39 231.3
3038	560 251 191.512 914	560 251 191.802 791	0.289 877
3795	106 393 708.697 129	106 393 708.697 129	0

(a) $Maxit = 10$

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 922 770.580 162	0
657	66 525 223.667 243	66 526 822.416 213	1598.75
724	52 143 135.590 922	52 143 134.854 769	0.736 153
1655	141 861 655.971 517	141 861 655.971 517	0
1817	132 745 384.513 000	132 745 376.885 252	7.627 75
2152	158 791 143.153 830	158 791 141.777 732	1.3761
2319	959 654 421.529 881	959 644 195.465 368	10 226.1
3038	560 251 222.054 362	560 251 196.431 925	25.6224
3795	106 393 708.697 129	106 393 708.744 303	0.047 174

(b) $Maxit = 30$

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 922 770.580 162	0
657	66 525 223.667 243	66 526 822.416 213	0.142 696
724	52 143 135.590 922	52 143 134.854 769	1.220 85
1655	141 861 655.971 517	141 861 655.971 517	0
1817	132 745 384.513 000	132 745 376.885 252	0.471 023
2152	158 791 143.153 830	158 791 141.777 732	92.0981
2319	959 654 421.529 881	959 644 195.465 368	21 109.3
3038	560 251 222.054 362	560 251 196.431 925	12.9153
3795	106 393 708.697 129	106 393 708.744 303	0

(c) $Maxit = 50$

Tabella 3: Funzioni obiettivo con $k_{max} = 10$

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 922 770.580 162	27 922 770.580 162	0
657	66 525 225.850 837	66 525 223.475 457	2.375 38
724	52 143 134.985 571	52 143 134.854 769	0.130 802
1655	141 861 655.971 517	141 861 655.971 517	0
1817	132 745 376.199 681	132 745 375.976 271	0.223 41
2152	158 791 145.275 574	158 791 141.708 503	3.567 07
2319	959 653 165.670 748	959 660 997.631 989	7831.96
3038	560 251 194.214 985	560 251 192.043 423	2.171 56
3795	106 393 708.697 129	106 393 708.697 129	0

(a) *Maxit* = 10

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 945 424.730 378	27 922 770.580 162	22 654.2
657	66 526 822.416 213	66 526 822.690 530	0.274 317
724	52 143 135.603 037	52 143 135.590 922	0.012 115
1655	141 861 655.971 517	141 861 655.971 517	0
1817	132 745 376.626 625	132 745 375.960 405	0.666 22
2152	158 791 145.709 625	158 791 142.055 673	3.653 95
2319	959 628 211.472 722	959 605 188.058 435	23 023.4
3038	560 251 276.537 833	560 251 191.433 878	85.104
3795	106 393 708.697 129	106 393 708.697 129	0

(b) *Maxit* = 30

i	f_k^*	f_{k++}^*	$\lvert f_k^* - f_{k++}^* \rvert$
198	3 083 342.539 589	3 083 342.539 589	0
493	27 945 424.730 378	27 922 770.580 162	22 654.2
657	66 526 822.416 213	66 526 822.690 530	0.274 317
724	52 143 134.985 571	52 143 135.590 922	0.605 351
1655	141 861 655.971 517	141 863 743.904 387	2087.93
1817	132 745 376.497 139	132 745 375.960 405	0.536 734
2152	158 791 144.071 038	158 791 156.336 263	12.2652
2319	959 607 354.009 766	959 605 188.058 435	2165.95
3038	560 251 193.302 626	560 251 194.485 887	1.183 26
3795	106 393 708.764 099	106 393 708.697 129	0.066 97

(c) $Maxit = 50$

Tabella 4: Test di Wilcoxon

k_{max}	*Maxit*	N	W^+	W^-	*Vincitore*
5	10	7	16	12	K-means++
	30	7	12	16	K-means
	50	7	12	16	K-means
8	10	7	19	9	K-means++
	30	7	21	7	K-means++
	50	6	20	1	K-means++
10	10	6	15	6	K-means++
	30	7	26	2	K-means++
	50	9	21	24	K-means

Analisi dei risultati. Come si evince dai dati osservati, le funzioni obiettivo prodotte dai due algoritmi sono molto simili tra loro e le principali variazioni si hanno su istanze di media dimensione.

La funzione obiettivo dell'istanza più piccola è uguale in tutti i test, per ognuno degli algoritmi, e questo fa presuppore che il valore trovato sia un ottimo globale. Per quanto riguarda l'istanza di taglia massima, la variazione riscontrata tra un test e un altro è minima e questo fa presupporre che l'algoritmo abbia trovato un buon ottimo locale.

Il test di Wilcoxon mostra che per un valore basso di k_{max}, il K-means è l'algoritmo che ottiene un risultato migliore all'aumentare del numero di iterazioni. Sapendo che valori piccoli dell'intorno comportano una soluzione molto simile alla precedente, il K-means

risulta particolarmente adatto quando si vuole intensificare. All'aumentare della massima dimensione dell'intorno, il K-means++ produce risultati migliori, risultando più idoneo per la diversificazione.

In base alle osservazioni effettuate, usare il K-means++ con un valore di k_{max} pari a 8 e *Maxit* pari a 30, consente di trovare delle buone soluzioni in termini di tempo e costo computazionale.

Conclusioni

I due algoritmi consentono di ottenere una buona partizione dello spazio bidimensionale di partenza. A seconda delle diverse configurazioni dei parametri di input si ottengono diversi tipi di soluzioni, mantenendo una complessità computazionale relativamente bassa.