Università degli Studi di Padova

Facility location problem analysis

Applicazione al sistema di Taxi
della città di Porto (PL)

Piona Davide **1149616**
Luca Venir **1139089**
Matteo Stringher **1151875**
Carlo Rizzardo **1156404**
Stefano Rettore **1158049**

1 Dataset e obiettivi

Per la nostra analisi è stato cercato un dataset che contenesse informazioni utili ad una analisi del problema di facility location. La nostra ricerca si è quindi concentrata verso una selezione di dati che fornisse la reale posizione di un utente che necessita di un servizio.
La scelta è ricaduta sul dataset *"Taxi Service Trajectory"*, di dimensione 1,89 GB e contenente quasi due milioni di righe. Questo dataset fornisce un'accurata descrizione delle reali traiettorie eseguite da 422 taxi della città di Porto, in Portogallo. Le corse vengono fornite come sequenza di posizioni GPS, delle quali solo la prima è stata considerata come punto di interesse, cioè la coordinata di partenza del viaggio. L'obiettivo della analisi è valutare quale sia il miglior algoritmo per il problema di facility location al variare del numero di centri, cioè di piazzole da cui far partire i taxi. Per questo abbiamo testato diversi algoritmi di clustering, sempre al variare di k: k-means per quanto riguarda la distanza L2, mentre PAM, CLARA e CLARANS per la reale distanza sulla superficie terrestre. Mentre *K-Means* minimizza la somma dei quadrati delle distanze, gli algoritmi PAM, CLARA e CLARANS sono algoritimi *K-Median* e minimizzano, nel nostro caso, la somma delle distanze geodetiche, più adatta a rappresentare il costo complessivo per i taxi, legato linearmente alla distanza.
Nel dataset ogni riga corrisponde ad un intero percorso di un taxi e contiene 9 feature: TRIP_ID, CALL_TYPE (A,B,C), ORIGIN_CALL, ORIGIN_STAND, TAXI_ID, TIMESTAMP, DAY_TYPE (A,B,C), MISSING_DATA, POLYLINE. Il dataset, caricato per una Prediction Challenge in ambito Machine Learning, è disponibile al link:
`https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015`

2 Descrizione del lavoro svolto

Per svolgere il lavoro di gruppo è stato utilizzato il sistema di versionamento Git e la piattaforma online GitHub. Come ambiente di sviluppo è stato scelto Eclipse compilando tramite Gradle.

2.1 Filtraggio dataset

Dopo aver caricato il dataset su un JavaRDD di tipo *TaxiTrip* è stato necessario filtrarlo mantenendo solo le informazioni importanti. Il primo filtraggio è stato effettuato sul parametro CALL_TYPE: sono state selezionate solo le corse di tipo 'C', cioè "le corse richieste su una strada casuale". Queste rappresentano punti di reale interesse da parte del cliente, informazioni dunque utili per cercare i centri dei cluster.
Successivamente sono state filtrate le corse non complete: talvolta alcuni dati erano assenti, nonostante il parametro MISSING_DATA corrispondente assumesse valore 'FALSE', quindi sono stati filtrati in modo più semplice valutando se POLYLINE, ossia la sequenza di posizioni GPS, avesse una lunghezza minima.
Al termine di questi filtraggi preliminari ogni elemento chiave-valore di tipo *TaxiTrip* è stato trasformato in un elemento chiave-valore di tipo *Position* contenente: l'identificatore della corsa, pickup-latitude, pickup-longitude. Infine, per evitare di avere posizioni anomale causate da rumore, sono stai filtrati gli elementi di tipo *Position*, mantenendo solo quelli aventi valori di latitudine e longitudine ragionevoli.
Al termine di questo processo il dataset risultante su cui è stato effettuato l'analisi presentava 523016 righe. Una rappresentazione grafica viene fornita in figura 1.
Come spiegato negli obiettivi, il nostro progetto confronta alcuni algoritmi di clustering tra cui un algoritmo *K-Means* e diversi approcci *K-Median*: una prima implementazione è basata sull'algoritmo PAM adattato per MapReduce, una seconda implementazione con algoritmo CLARA e un'ultima implementazione CLARANS adattato per MapReduce. Vengono ora presentate le diverse soluzioni, che hanno caratteristiche differenti sia per quanto riguarda la qualità del clustering risultante che per il tempo di esecuzione.

Figura 1: I punti del dataset delineano le strade della città di Porto

2.2 Algoritmo *K-Means*

L'algoritmo di *K-Means*, per sua natura utilizza la distanza L2. Essa non tiene conto della distanza sulla superficie terrestre. I dati appaiono all'algoritmo come disposti su un piano. Nella nostra implementazione è stato utilizzato direttamente dalle API come strumento di confronto e verifica.

2.3 Algoritmo MapReduce-PAM

Per trovare k cluster, PAM parte da un insieme di k centri casuali e confronta tutti i rimanenti $N-k$ elementi residui con i centri attuali. Se sostituendo un centro si ottiene una funzione obiettivo minore, tale centro viene effettivamente sostituito e si procede con un nuovo confronto. La complessità di PAM è $O(k(N-k)^2)$ in ogni iterazione, considerando anche il calcolo della funzione obiettivo.
PAM non è un algoritmo distribuito, viene eseguito iterativamente su un singolo worker. Per essere parallelizzato, deve essere adattato scegliendo una strategia di divisione del dataset per eseguire l'algoritmo iterativo su ciascuna partizione. Ognuna di queste restituisce k centri rappresentativi della ricerca locale nella sample. Successivamente vengono aggregati tutti i punti ottenuti e eseguendo nuovamente PAM sul dataset ricavato si ottiene, con buona approssimazione, il clustering per il dataset. Lo pseudocodice viene presentato nell'algoritmo 1. L'approccio seguito

Algoritmo 1 : MR-PAM

Input $D = \{p_i : 0 \leq i \leq N\}$ dataset, l

Round 1 do

MAP: partiziona D in l partizioni $D_0, ..., D_{l-1}$ e assegna casualmente ogni punto ad una partizione con probabilità $\frac{1}{l}$.

REDUCE: $\forall$ partizione D_i esegui in parallelo PAM ottenendo un insieme di k centri $T_i = \{c_1, ..., c_k\}$.

Round 2 do

MAP: identità

REDUCE: raccogli tutti gli insiemi di centri T_i in un unico $T = \bigcup_{i=0}^{l-1} T_i$. Esegui PAM con un singolo worker su T.

Output $S : \{c_1, ..., c_k\}$ insieme di centri ottenuto dall'esecuzione di PAM su T

ricalca quello per MR-Farthest-First Traversal. Scegliendo il numero di reducer l pari a $\sqrt{\frac{N}{k}}$ è possibile minimizzare lo spazio richiesto da ogni reducer e si ottiene: spazio locale $M_L = O(\sqrt{Nk})$, spazio aggregato $M_A = O(N)$, ed un numero di round pari a 2.

MR-PAM Round 1: Map

Ogni punto è assegnato ad uno degli l worker con probabilità $\frac{1}{l}$; in questo modo si ottiene una divisione del dataset in l parti ognuna con $O(\sqrt{Nk})$ elementi in media. La scelta di assegnare casualmente la partizione, invece di dividere equamente i punti, permette una maggiore semplicità nella realizzazione. Altrimenti sarebbe necessario numerare gli elementi del dataset per dividerli equamente.

```
JavaPairRDD<Integer, Position> dDataset = dataset.mapToPair((point) -> {
        return new Tuple2((int) (Math.random() * l), point);
}).cache();
```

La fase di Map in Spark è realizzata con il metodo `mapToPair`, assegnando ad ogni punto una coppia chiave-valore, dove la chiave rappresenta l'indice del reducer a cui è assegnato, generato casualmente.

MR-PAM Round 1: Reduce

```
JavaPairRDD<Integer, Position[]> dCenters =
   dataset.groupByKey().mapToPair((partition) -> {
        return new Tuple2(partition._1(), iter_pam(partition._2(), k));
});
```

Il Reduce in Spark viene realizzato tramite un comando `groupByKey().mapToPair()` anziché il metodo `reduce`, poiché l'intera partizione deve essere ridotta utilizzando l'algoritmo PAM.
Il metodo `groupByKey()` raggruppa tutta la partizione mentre il Map successivo riduce l'intera partizione in una singola coppia chiave-valore con chiave indice del reducer e valore l'array di k centri.

MR-PAM Round 2

L'ultimo round viene effettuato dal solo master, che aggrega i risultati usando il metodo `collect()` ed esegue nuovamente l'algoritmo PAM.

```
List<Tuple2<Integer, Position[]>> lCenters = dCenters.collect();
ArrayList<Position> t = new ArrayList(); //unisco tutti i Ti
for (Tuple2<Integer, Position[]> ti : lpartitionCenters)
        t.addAll(Arrays.asList(ti._2()));
return Kmedian_PAM.getPAMCenters(t, k);
```

2.4 Algoritmo CLARA

CLARA (*Clustering LARge Applications*) è un algoritmo *K-Median* basato sul partizionamento, realizzato da Kaufman e Rousseeuw [2] per gestire dataset molto grandi.
CLARA estrae alcune sample di una dimensione data, possibilmente piccola, dal dataset ed esegue PAM in ciascuna di esse in modo analogo a quanto fatto dalla nostra implementazione di MR-PAM tuttavia, per ottenere una migliore approssimazione, invece di aggregare tutti i risultati T_i ottenuti e rieseguire PAM una seconda volta, calcola per ogni T_i la funzione obiettivo sull'intero dataset e poi sceglie come risultato il T_i con funzione obiettivo minore. Ciò consente di scegliere la sample che meglio approssima il dataset. Con questo algoritmo, supponendo $sampleSize = \sqrt{Nk}$, $nSample = \sqrt{\frac{N}{k}}$, e utilizzando un numero di reducer $l = \sqrt{\frac{N}{k}}$, la complessità è equivalente a quella ottenute con l'algoritmo MR-PAM poichè ogni sample diventa della stessa dimensione di una partizione di PAM e si ha un numero di sample uguale a quello di PAM. Si ottiene invece un vantaggio riducendo, anche in modo considerevole, i valori di sampleSize e nSample, in quanto CLARA nel secondo round invece di aggregare i dati sceglie il T_j più rappresentativo dell'intero dataset. È quindi possibile eseguire un clustering lavorando solo una minima parte del dataset. Secondo dati sperimentali, Kaufman e Rousseeuw in [2] dimostrano che è sufficiente utilizzare CLARA con appena 5 sample

da $S = 40 + 2k$ elementi per ottenere un risultato approssimativo del clustering in tempi molto inferiori rispetto al calcolo sull'intero dataset. In questo modo viene limitata la complessità temporale, che non è più vincolata dalla dimensione del dataset. Da segnalare che il risultato rischia di allontanarsi dal risultato ottimale nel caso in cui vengano estratte dal dataset sample non molto rappresentative. Nel nostro codice questa variante viene identificata con il nome di CLARAFAST.

Algoritmo 2 : CLARA

Input $D = \{p_j : 0 \leq k \leq N\}$ dataset, nSample, sampleSize

Round 1 do

MAP: estrai da D l sample $S_0, ..., S_{nSample-1}$ di dimensione sampleSize.

REDUCE: $\forall$ sample S_i esegui in parallelo PAM ottenendo un insieme di k centri $T_i = \{c_1, ..., c_k\}$.

Round 2 do

MAP: invia tutti i clustering T_i e una partizione D_j di D dei punti ad ogni reducer e $\forall T_i$ calcola la funzione obbiettivo parziale con la partizione D_j mappando i risultati in $\{T_i, \phi_{j,i}\}$.

REDUCE: raccogli tutti i $\{T_i, \phi_{j,i}\}$ e $\forall i : \{T_i, \sum_j \phi_{j,i}\} = \{T_i, \phi_i\}$

Output $S : \{c_1, ..., c_k\} = T_k$ tale per cui $k = \arg\min \phi_i$

CLARA Round 1

L'estrazione con reimbussolamento delle sample di dimensione precisa è fornita dal metodo `takeSample`, Vengono estratti `sample_size * n_sample` elementi e poi assegnati equamente alle `n_sample`.
La scelta di eseguire una sola volta il metodo `takeSample()` di Spark è motivata dalla lentezza del metodo: risulta più conveniente eseguire una sola chiamata per estrarre tutti gli elementi necessari e suddividere successivamente gli elementi estratti tra le sample, piuttosto che eseguire `n_sample` chiamate diverse.

```
List<Position> t = dataset.takeSample(true, sample_size * n_sample);
ArrayList<Tuple2<Integer, Position>> toSample = new ArrayList();
for (int i = 0; i < t.size(); i++)
    toSample.add(new Tuple2((i % n_sample), t.get(i)));
JavaPairRDD<Integer, Position> sample = sc.parallelizePairs(toSample);
```

La fase di Reduce, a livello di codice, è analoga all'implementazione precedente MR-PAM.

CLARA Round 2: Reduce

L'implementazione del calcolo della funzione obiettivo per tutti i punti avviene in maniera leggermente diversa rispetto a quella vista in teoria: il metodo `aggregate` di Spark condensa le fasi di Map e Reduce. La struttura dati `centers[][]` è un array, in cui ogni elemento rappresenta l'insieme di k centri relativi all'i-esimo sample.
Questo round è implementato nel metodo `parallelObjectiveFunction(Position[][] centers, int l)` presente nel file *Kmedian.java*.

2.5 Algoritmo MapReduce-CLARANS

CLARANS è un algoritmo di clustering basato sulla ricerca randomizzata e costituisce un compromesso tra i tempi di esecuzione lunghi di PAM e il rischio di ottenere clustering di scarsa qualità con CLARA. L'implementazione prevede un algoritmo in MapReduce totalmente analogo a quello utilizzato con PAM, con una modifica sulla componente iterativa eseguita su ogni reducer. In questo modo si migliora l'esecuzione in quanto il tempo speso nella fase di reduce domina rispetto ai tempi spesi nelle fasi di Map e shuffle.
Per capire l'approccio dell'algoritmo iterativo si immagini la ricerca dei k medioidi come una ricerca astratta attraverso

un certo grafo. In questo grafo $G(n, k)$ un nodo rappresenta un insieme di k punti $\{O_1, ..., O_k\}$ che corrispondono ai punti scelti come medoidi ed ogni nodo è connesso ai propri vicini tramite un arco: due nodi sono collegati se i punti differiscono di un solo elemento. È possibile verificare che in questo grafo ogni nodo ha $k(N - k)$ archi e dato che ogni nodo costituisce un insieme di k punti, esso rappresenta anche un possibile insieme di centri per un k-clustering. In questo grafo un approccio di tipo PAM verificherebbe ogni possibile vicino del nodo attuale e continuerebbe la ricerca fino al raggiungimento di un nodo di minimo locale, tale per cui i vicini hanno funzione obiettivo maggiore, CLARA non controllerebbe tutti i nodi vicini, ma solo quelli appartenenti alla sample estratta, mentre CLARANS ad ogni step estrae in modo casuale alcuni vicini da esplorare. Si impone $maxNeighbor$ come numero massimo di vicini da controllare e un parametro $numLocal$ che determina il numero di volte che CLARANS ripete la ricerca in modo da non limitarsi ad un singolo sotto-grafo e raggiungere un punto di minimo che sia anche minimo locale per il grafo complessivo.

Algoritmo 3 : CLARANS

Input Parametri $numLocal$ e $maxNeighbor$.

1. Inizializza i a 1, e $mincost$ al valore massimo possibile.

2. Inizializza $current$ ad un nodo arbitrario del grafo $G_{n,k}$.

3. Inizializza j a 1.

4. Prendi un vicino a caso S di $current$.

5. Se S ha una funzione obiettivo minore di $current$, imposta $current$ a S e torna allo **Step 3**.

6. Altrimenti, incrementa j. Se $j \leq maxNeighbor$, torna allo **Step 4**.

7. Altrimenti, se $j > maxNeighbor$ confronta la funzione obiettivo di $current$ con $mincost$ e se minore inizializza $mincost$ al valore della funzione obiettivo di $current$ e inizializza $bestnode$ a $current$.

8. Incrementa i. Se $i > numLocal$ ritorna $bestnode$ e concludi l'esecuzione, altrimenti torna allo **Step 2**.

2.6 Clustering evaluation

Prima di eseguire alcune verifiche sulla qualità dei cluster, è stata implementata e quindi applicata la Statistica di Hopkins al dataset: sono stati estratti 30 campioni (crescenti in cardinalità) e su di essi è stato valutato il coefficiente H che, al crescere della cardinalità, si avvicina di molto a zero (oscilla tra $H = 0.0108$ a $H = 0.0017$); si conclude quindi che il dataset tende ad avere una struttura che si presta al clustering.
La verifica della qualità dei vari clustering viene effettuata in primo luogo confrontando la somma delle distanze di ogni punto dal centro del suo cluster. Essa coincide con la funzione obiettivo per gli approcci *K-Median*, ma differisce dalla funzione obiettivo dell'approccio *K-Means*. La scelta è giustificata dal fatto che lo scopo del problema è minimizzare la distanza da ogni cliente.
Inoltre, per verificare la qualità dei clustering, è stato implementato il Silhouette Coefficient e valutato su diversi cluster (con $K = 30, 40, 50, 60$) per tutti gli algoritmi implementati; dai risultati ottenuti è emerso che, al crescere di K, la media del coefficiente tende a crescere: al caso peggiore, $S = -0.5769$ (CLARANS su $K = 40$), al caso migliore, $S = 0.3111$ (CLARA su $K = 60$).

2.7 Visualizzazione dei dati

Per fornire una visualizzazione dei dati e del clustering risultante è stata sviluppata la classe *ClusteringDrawing* per rappresentare i punti di pickup nello spazio su immagini. Il metodo principale della classe, *draw*, riceve in input un RDD con le posizioni associate al relativo cluster, il metodo provvede a mappare le posizioni sull'immagine e colorarle in modo differente a seconda del cluster di appartenenza (saranno tutti bianchi se il numero dei cluster viene impostato a 0). All'ulteriore metodo *drawCenters* è affidato il compito di disegnare sull'immagine i centri dei cluster. Inoltre si può decidere di visualizzare i punti con una certa trasparenza, effetto che diviene interessante nei casi in cui vi siano più punti sovrapposti, dei quali si riesce così ad intuirne la densità. Si ottengono quindi delle

immagini in cui si riesce a riconoscere distinguere i diversi cluster. Un esempio viene mostrato nelle figure 2a, 2b, 2c, presenti nella sezione successiva.
L'utilizzo di un semplice RDD in input permette di ricevere i dati da algoritmi di clustering differenti tra loro. Data la difficoltà di scrivere in parallelo sull'immagine quest'operazione viene fatta direttamente estraendo i dati dall'RDD e processandoli linearmente, in ogni caso il nostro dataset non è così grande da generare difficoltà.

3 Risultati

Una prima analisi grafica conduce a conclusioni immediate. Confrontando le varie immagini si nota come i centri di *K-Means* siano maggiormente sparsi. Si può desumere già da ora, che *K-Means* non sia adatto a risolvere il problema di facility location. Si nota invece come gli algoritmi per *K-Median* ottengano centri abbastanza concentrati nell'area della città.

(a) *K-Median* PAM (b) *K-Means* (c) *K-Median* CLARAFAST

Figura 2: Risultati dei clustering

Successivamente è stata condotta una analisi parametrica al variare del numero di centri, con lo scopo di valutare l'andamento della funzione obiettivo. I risultati sono riassunti nel grafico in figura 3. Si nota come non vi sia una differenza sostanziale tra i vari risultati. I risultati sono abbastanza simili, eccetto per CLARA che restituisce un clustering leggermente migliore. Nella nostra implementazione per quanto riguarda CLARANS, nlocal è pari a 3 mentre maxneighbor è 1% di k(n-k) ovvero l'1% degli archi che controllerebbe PAM per ogni nodo raggiunto. Vengono esclusi dal grafico i risultati di *K-Means*, poichè non renderebbero fruibile la visualizzazione dei dati. Quest'ultimo nonostante sia molto veloce nella computazione, restituisce risultati non accettabili.
Oltre alla funzione obiettivo è necessario valutare la quantità di tempo impiegata dell'algoritmo. Nel grafico di figura 4 le differenze sono invece sostanziali. Nonostante PAM sia stato adattato per lavorare in parallelo la computazione risulta ancora molto lenta. L'algoritmo è poco adatto ad un calcolo su istanze troppo pesanti. Anche CLARA risulta molto lento e poco adatto, nonostante fornisca la miglior soluzione non è la scelta più adatta per questo tipo di analisi. La sua variante CLARAFAST, invece, risulta molto rapida come era atteso. Non essendo dipendente dalla taglia dell'istanza riesce a concludere la ricerca dei centri velocemente. Il suo risultato in termini di funzione costo sul dataset ha ottenuto buoni risultati, ma nell'analisi con punti casuali effettuata in fase di sviluppo aveva restituito un'approssimazione peggiore del 50% circa, rispetto agli altri algoritmi. Infine, CLARANS impiega un tempo inferiore rispetto a PAM e CLARA, con la garanzia di ottenere una buona soluzione.

Tabella 1: Coefficiente di silhouette per k = 30

	PAM	CLARA	CLARAFAST	CLARANS	K-means
Coefficiente di silhouette	-0.5158	-0.0012	-0.4506	-0.6445	-0.3506

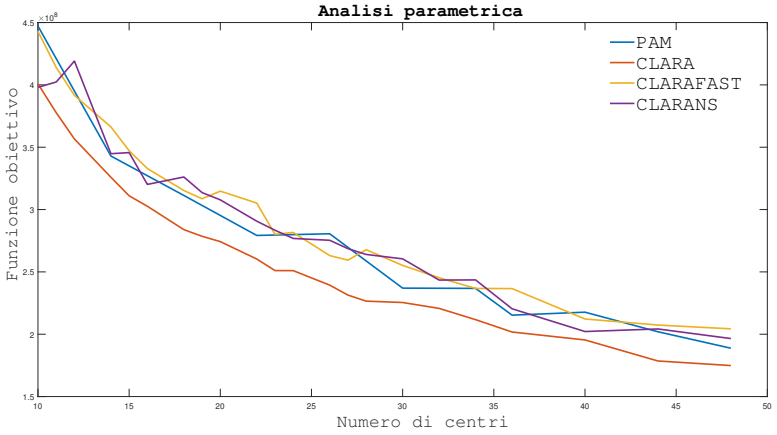

Figura 3: Analisi al variare di k

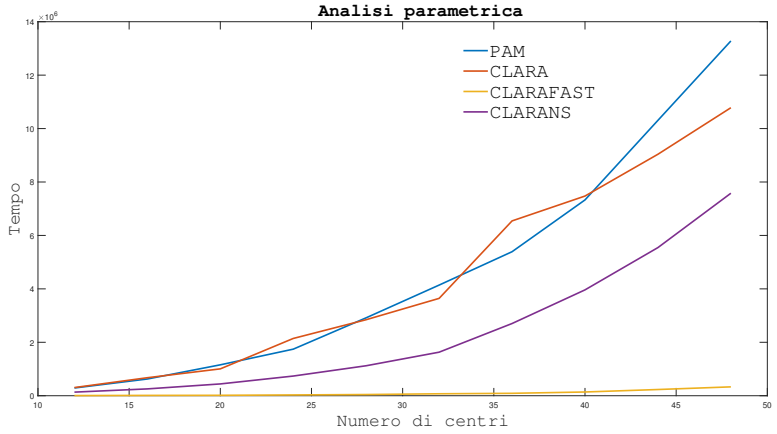

Figura 4: Analisi temporale

4 Conclusioni e sviluppi futuri

Nonostante *K-Means* sia molto veloce non risulta adatto, dato che ottiene una soluzione peggiore di quasi tre ordini di grandezza. CLARAFAST ha invece ottenuto risultati incoraggianti, ma sarebbe incauto affidarsi solamente a questo algoritmo a causa della sua forte dipendenza dalle sample estratte.
L'analisi non ha tenuto conto della reale distanza su strada, ma solo della distanza geodetica tra due punti, a causa del carico computazionale. Tale distanza può essere calcolata, utilizzando le API di Google, ma richiede di contattare i loro server numerose volte. Potrebbe essere svolta un'analisi su un ristretto sample da considerare come validation set, per valutare il percorso dei taxi.

Riferimenti bibliografici

[1] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. *CoRR*, abs/1109.1579, 2011.

[2] L. Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley.

[3] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. 2002.