Ming Hua
Jian Pei
Ada W. C. Fu
Xuemin Lin
Ho-Fung Leung

Efficiently answering top-k typicality queries on large databases

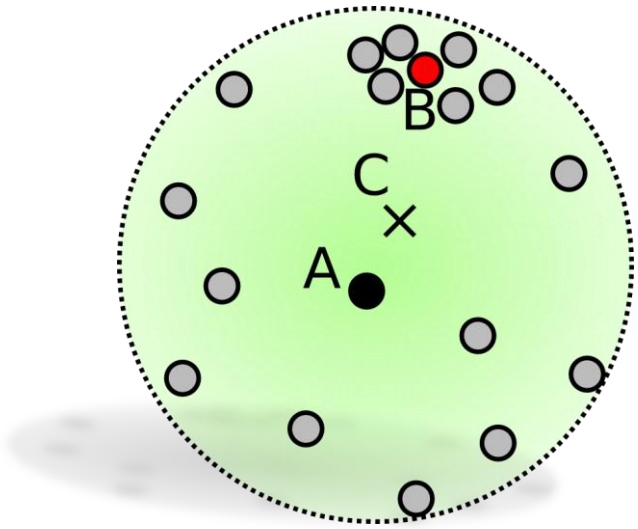
Scenario

- Imparare dagli esempi è una strategia efficace estesamente adottata nella pratica.
- Esempi significativi sono spesso migliori di mere descrizioni di caratteristiche, come primo passo per comprendere un nuovo concetto.
- Come devono essere scelti gli esempi candidati?
  - Devono essere tipici;
  - Esempi reali sono preferibili a casi ideali;
- Usare esempi tipici per analizzare e riassumere un insieme di oggetti può essere utile anche per interrogazioni efficaci su Data Base.

Scenario - Esempio

- Come spieghiamo il concetto di mammifero ad un bambino?

Tipicità



Nell'insieme a lato, il punto A rappresenta la mediana, il punto C la media e il punto B il punto più tipico.

Dato che la media e la mediana tendono a rappresentare il centro geometrico dell'insieme, non sono necessariamente correlate alla distribuzione di probabilità, "catturata" invece dalla tipicità.

- La tipicità di un oggetto dipende dalla distribuzione degli altri oggetti appartenenti allo stesso insieme.
- Questo implica che una semplice interrogazione di tipo top-k non porta ad ottenere il risultato cercato.
- Le *top-k typicality queries* sono un tipo speciale di interrogazioni top-k nelle quali compare anche una misura di tipicità basata su certi attributi specificati.

Tipicità semplice

Per misurare la tipicità in un insieme di oggetti S si introduce il concetto di tipicità semplice.

1. **SIMPLE TYPICALITY:** dato un insieme di oggetti S, un oggetto $o \in S$ è più tipico degli altri se è più probabile che appaia in S; quindi considerando l'insieme S come campione di una variabile casuale continua $\hat{S}$ ed o, la *simple typicality* è definita come :

$$T(o,S) = f(o)$$

con f funzione di densità di probabilità in $\hat{S}$.

ESEMPIO DI QUERY:

SELECT $A_1,\ldots,A_n$ **FROM S WHERE P**
ORDER BY SIMPLE TYPICALITY ON $(A_{i1},\ldots,A_{il})$
LIMIT k

Tipicità discriminativa

La tipicità semplice appena introdotta funziona fintantoché gli oggetti dell'insieme preso in esame sono considerati indipendentemente da altri oggetti *non* nell'insieme. Per questi casi si introduce la tipicità discriminativa.

2. **DISCRIMINATIVE TYPICALITY:** dato un insieme di oggetti S ed un sottoinsieme target C ($C \subset S$), la *discriminative typicality* di un oggetto $o \in C$ misura quale oggetto è più tipico in C ma non in (S\C) ed è definita come:

$$DT(o,C,S) = [f(C|o) - f((S\backslash C)|o)] \times f(o)$$

con $f(C|o)$, $f((S\backslash C)|o)$ ed $f(o)$ funzioni di densità di probabilità condizionate

ESEMPIO DI QUERY:

```
SELECT A1,…,An FROM S WHERE P
ORDER BY DISC TYPICALITY ON (Ai1,…,Ail)
LIMIT k
```

Nel seguito, per semplicità, ci si riferirà alla sola tipicità semplice

Tipicità - Esempio

- Jeff è un ragazzo che gioca a basket. Il suo sogno è poter giocare un giorno nell'NBA.

Può essere interessato a sapere:

- "Quali sono i 3 giocatori più tipici dell'NBA?"
  - Una ***simple top-k typicality query*** trova i k giocatori più tipici tra tutti i giocatori dell'NBA.

Oppure:

- "Quali sono le 2 guardie più tipiche, distinguendo le guardie da tutti gli altri ruoli dei giocatori dell'NBA?"
  - Una ***discriminative top-k typicality query*** trova i k giocatori con i più alti valori di tipicità discriminativa tra i giocatori dell'NBA appartenenti al sottoinsieme delle guardie.

Query Answering algorithms

- Per poter calcolare la tipicità di un'insieme di oggetti è essenziale stimare la densità di probabilità degli stessi.
- Si usa un *Kernel estimator* , caratterizzato da una funzione kernel *K* (preferibilmente Gaussiano) e da un parametro di larghezza di banda *h*.
- In uno spazio metrico generico l'unico parametro usato per la stima è la distanza (o similarità) tra due oggetti $\in$ S.

Approssimazione della funzione di densità di probabilità:

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} G_h(x, o_i) = \frac{1}{n\sqrt{2\pi}}\sum_{i=1}^{n} e^{-\frac{d(x,o_i)^2}{2h^2}}$$

con $G_h(x,o_i)$ kernel Gaussiano e $d(x,o_i)$ distanza tra x ed o_i nello spazio metrico.

The Exact Algorithm

Input: a set of n obj $o_1,\ldots,o_n$ and an integer k
Output: top-k simple typicality values
FOR each obj o, T(o) = 0;
FOR i = 0 **TO** n-1
FOR j=i+1 **TO** n

$$w = \frac{1}{n\sqrt{2\pi}} \sum_{i=1}^{n} e^{-\frac{d(o_i,o_j)^2}{2h^2}}$$

$T(o_i) = T(o_i)+w;$
$T(o_j) = T(o_j)+w;$
END-FOR
END FOR
Return the top-k obj according to T(o).

L'algoritmo ha complessità $O(n^2)$ ed è troppo costoso per query on-line su grandi DB!

RT – Randomized Tournament Algorithm

- Per ottenere una prima stima della tipicità, mantenendo contenuta nel contempo la complessità del problema, si propone il metodo Randomized Tournament (RT).
- RT consiste nel dividere gli *n* oggetti dell'insieme considerato in gruppi casuali di dimensione *t*, ottenendo quindi $\lceil \frac{n}{t} \rceil$ gruppi.
- All'interno di ogni gruppo si usa l'algoritmo esatto per trovare l'oggetto col più alto valore di tipicità semplice.

RT – Randomized Tournament Algorithm

- Gli oggetti "vincitori" di ogni gruppo sono selezionati per il round successivo.
- Dopo il primo round, il numero di elementi si è dimezzato
- Ripetendo l'algoritmo sui "sopravvissuti", si arriverà ad ottenerne 2.

RT – Randomized Tournament Algorithm

- Il calcolo della tipicità all'interno di un gruppo ha complessità $O(t^2)$ ed i round totali sono $\lceil \log_t n \rceil$.
- Assumendo $n = t^m$, il numero totale di gruppi è un $O(n/t)$, per cui la complessità dello selezionare il vincitore finale risulta essere

$$O(t^2 \cdot n/t) = O(tn).$$

- Si può pensare di ripetere v volte un torneo per ottenere un risultato più accurato, quindi la complessità diviene $O(tnv)$.
- Se ci interessano i primi k oggetti ritornati dalla query (query top-k), allora il torneo andrà ripetuto k ulteriori volte per determinare le prime k posizioni, giungendo ad una complessità $O(tnvk)$.

RT – Randomized Tournament Algorithm

- Dopo aver determinato il primo vincitore, per determinare il secondo in classifica è necessario ripetere il torneo, facendo però "perdere" il primo classificato appena determinato al primo incontro.

Approssimazione Locale

- L'algoritmo RT è efficiente ma non garantisce la qualità dell'approssimazione.
- Oggetti "lontani" da quelli che si considerano portano un contributo che decade esponenzialmente all'aumentare della distanza ($G(x,o_i)$ è un'esponenziale negativa). Per questo si introduce il concetto di *local simple typicality*.
- **Local Simple Typicality:** dato un insieme di oggetti S ed un sottoinsieme C ($C \subset S$) ed una soglia di vicinato σ, la *local simple typicality* di un oggetto *o* è definita come:

$$LT(o,C,\sigma) = f(o|LN(C,\sigma))$$

$LN(C,\sigma)$ è la *σ-local neighborhood* di C, cioè l'insieme degli oggetti la cui distanza da *o* di almeno un oggetto è al massimo σ

Approssimazione Locale – Errore di stima

Approssimare la *simple typicality* di un oggetto con la sua *local simple typicality* introduce un errore massimo pari a

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{\sigma^2}{2h^2}}$$

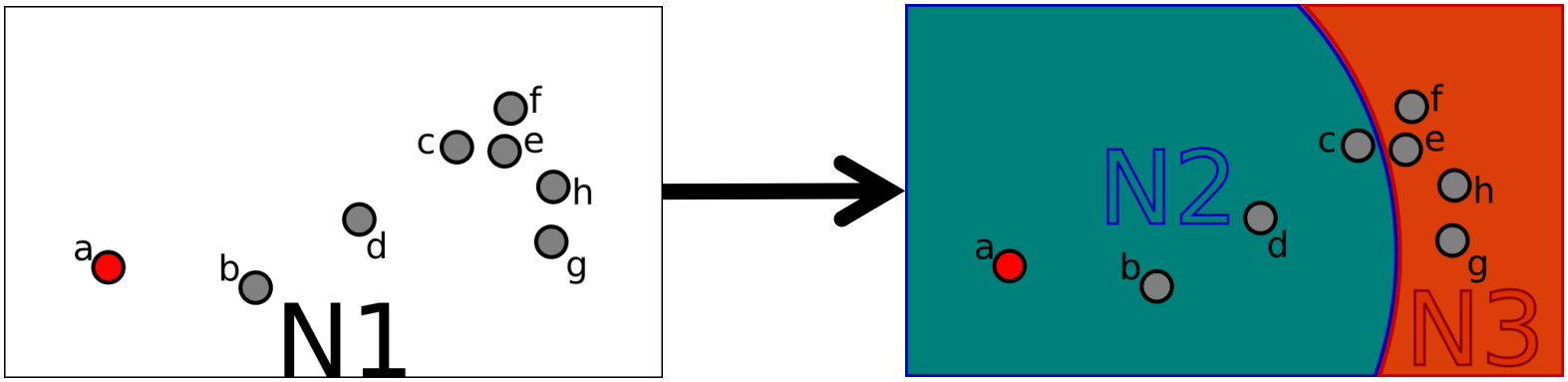
per qualsiasi oggetto dell'insieme C. L'errore introdotto ha quindi un upper bound garantito.

DLTA – Direct Local Typicality Approximation

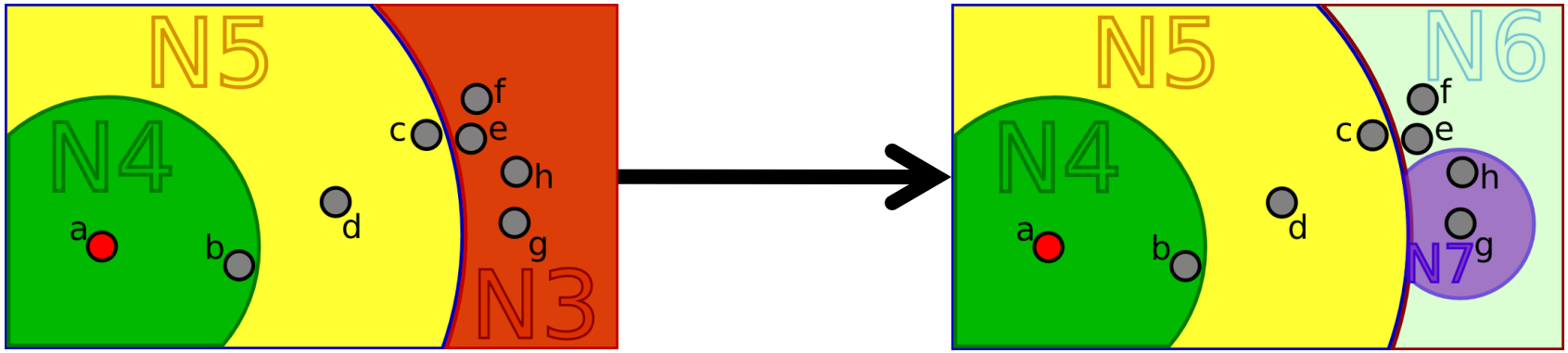
- Dalle osservazioni precedenti siamo quindi in grado di calcolare l'approssimazione della simple typicality di un oggetto garantendo anche la qualità.
- Data la soglia di vicinato σ, per ogni oggetto *o* di S si calcola prima il *σ-local neighborhood* e, successivamente, la *local simple typicality LT*.
- Per rispondere alle *top-k typicality queries* si prendono i k oggetti con i più alti valori di LT come approssimazioni delle *simple typicality* degli oggetti stessi.
- L'errore di stima risulta essere minore dell'upper bound definito prima.
- Per determinare il *σ-local neighborhood* si usa il VP-Tree, il quale è strutturato in modo da consentirne la ricerca in modo efficiente.

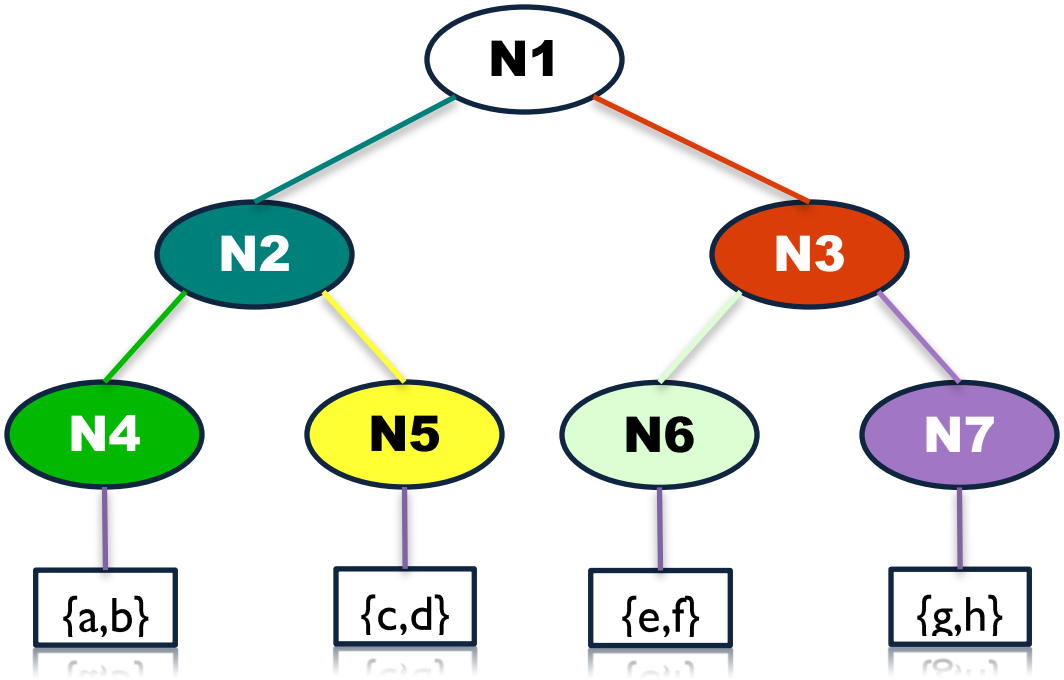
VP-Tree – Vantage Point Tree

- Il VP-Tree è un particolare tipo di BSP-tree che suddivide lo spazio tramite Ball Decomposition.
- La ricerca dei vicini di un punto si limita all'esplorazione di un solo sottoalbero, permettendo il "pruning" dell'altro.
- Per essere efficiente, l'albero prodotto deve essere il più bilanciato possibile.



VP-Tree


N5
N4
N3
a
b
c
d
e
f
g
h
N5
N6
N4
N7
a
b
c
d
e
f
g
h


N1
N2
N3
N4
N5
N6
N7
{a,b}
{c,d}
{e,f}
{g,h}

LT3 – Local Typicality Approximation using Tournaments

- Dalle osservazioni predecenti si evince che è possibile usare la *local typicality approximation* invece della *simple typicality* durante l'esecuzione dell'algoritmo RT garantendo anche la qualità della risposta.
- L'upper bound dell'errore commesso è lo stesso a meno di un fattore moltiplicativo che dipende dal numero di round presenti nel torneo (pari a $\lceil \log_t |S| \rceil$, dove S è l'insieme dei dati).
- Scegliendo valori opportuni di σ e di *t* la qualità è garantita.

PROBLEMA: i "partecipanti" al torneo in ogni gruppo sono scelti in modo *random* quindi è possibile che, nel caso peggiore, questi non siano "vicini" tra loro. La complessità dell'algoritmo risulta essere ancora $O(n^2)$.
COME RISOLVIAMO LA QUESTIONE?!

Local Typicality Tree (LT-Tree)

- L'LT-Tree è un particolare albero di ricerca costruito a partire da un VP-Tree.
- È possibile usare anche un MVP-Tree, un albero t-ario che usa più di un punto di vantaggio per partizionare lo spazio, in modo da ridurre il numero di round del torneo.
- Ad ogni nodo del VP-Tree è assegnato un *layer number* (la radice ha layer number zero). Sono rimossi dall'albero tutti i nodi i cui layer numbers non sono multipli di *t*.
- Per un nodo N il cui layer number è *jt*, colleghiamo N al suo antenato con layer number pari a (j-1)t.
- L'albero risultante è chiamato LT-Tree.
- Ogni nodo dell'LT-Tree porta con se le seguenti informazioni: *centro (approssimato)*, *raggio* e il *σ-neighborhood*.

Local Typicality Tree (LT-Tree)

- Una volta determinato per ogni nodo N dell'LT-Tree il suo centro c ed il relativo raggio (pari alla massima distanza tra c e gli altri oggetti in N), è possibile determinare il σ-*neighborhood* corrispondente.
- Questo ***nuovo*** vicinato così determinato fornisce sempre un'approssimazione di tipicità migliore.
- Per determinare quali oggetti appartengono al σ-*neighborhood* si esegue una range query che, a partire dalla radice, cerca i nodi che completamente giacciono nel σ-*neighborhood* di N.
- Una volta che tutti gli oggetti del nodo N' del VP-Tree si trovano nel σ-*neighborhood* di N, è possibile usare N' per rappresentarli e non occorre esplorare alcun sottoalbero di N'.

LT-Tree e Randomized Tournament

- Per rispondere ad una *top-k typicality queries* si esegue un torneo sull'LT-Tree bottom-up.
- Al primo passo il torneo ha inizio per ogni nodo foglia dell'albero.
- Il vincitore entra nel torneo del nodo padre.
- Il primo vincitore o_1 del torneo alla radice è l'approssimazione all'oggetto più tipico.
- Per l'approssimazione del secondo vincitore è possibile usare i risultati dei tornei eseguiti al primo ciclo in quanto gli unici tornei da ripetere saranno quelli che contengono o_1, escludendo l'oggetto stesso al primo torneo. I nuovi vincitori dei tornei intermedi rimpiazzeranno o_1 in tutti quelli successivi risalendo l'albero fino alla radice.
- In questo modo si ottengono i k oggetti con il più alto valore di tipicità che meglio rispondono alla query.
- L'approssimazione dell'oggetto più tipico risulta essere *molto vicina* a quello esatto.

Prova su Data set reale

- Query di esempio eseguite sullo Zoo Database , proveniente dall'UCI Machine Learning Database Repository.

Classe	# tuple	Più tipico	Più tipico (discriminativo)	Più atipico
Mammiferi	40	Cinghiale, Ghepardo, Leopardo, Leone, Lince, Mangusta, Puzzola, Puma, Procione, Lupo (T = 0,16)	Cinghiale, Ghepardo, Leopardo, Leone, Lince, Mangusta, Puzzola, Puma, Procione, Lupo (DT = 0,08)	Ornitorinco (T = 0,01)
Uccelli	20	Allodola, Fagiano, Scricciolo (T = 0,15)	Allodola, Fagiano, Scricciolo (DT = 0,04)	Pinguino (T = 0,04)
Pesci	14	Branzino/Spigola, Pescegatto, Aringa, Piranha (T = 0,15)	Branzino/Spigola, Pescegatto, Aringa, Piranha (DT = 0,03)	Carpa (T = 0,03)
Invertebrati	10	Gambero, Aragosta (T = 0,16)	Gambero, Aragosta (DT = 0,01)	Scorpione (T = 0,08)
Insetti	8	Falena, Mosca domestica (T = 0,13)	Moscerino (DT = 0,02)	Ape (T = 0,06)
Rettili	5	Orbettino (T = 0,17)	Crotalo (DT = 0,007)	Serpente marino (T = 0,08)
Anfibi	3	Rana (T = 0,2)	Rana (DT = 0,008)	Tritone, Rospo (T = 0,16)

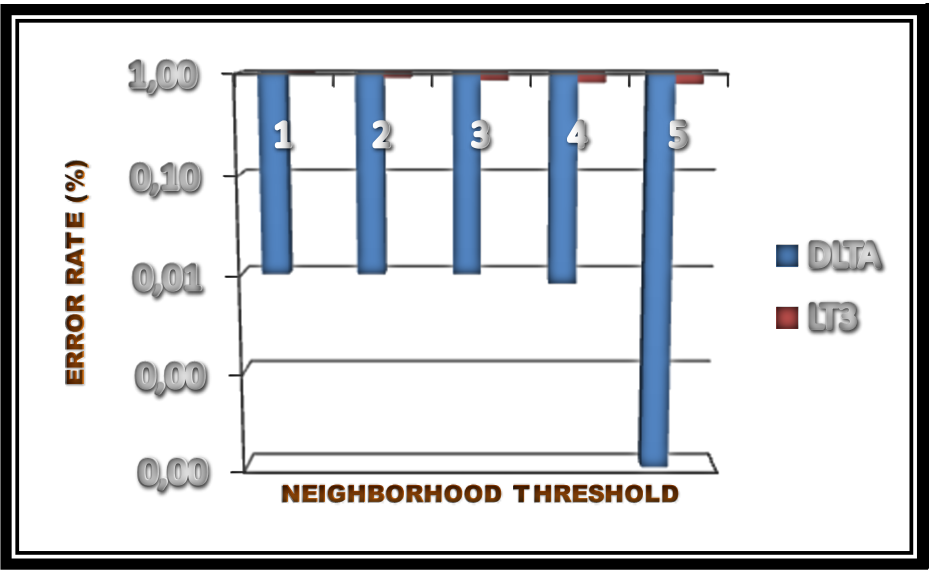
Risultati

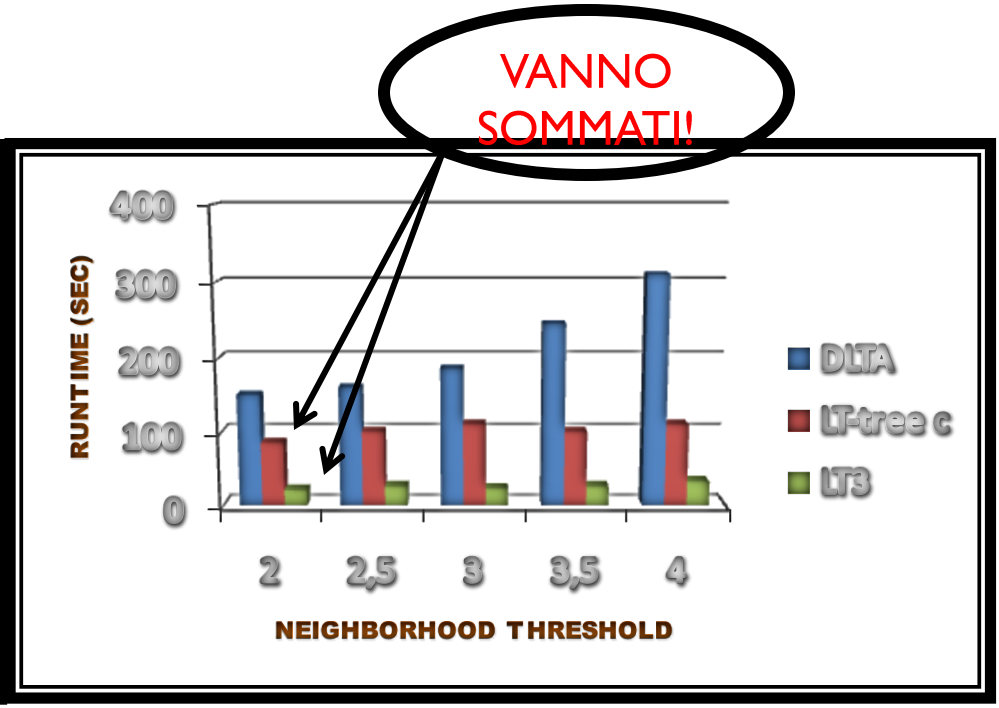
- Test effettuati su un insieme di dati sintetici con 72 attributi generato dal "Quadraped Animal Data Generator".
- Si riportano i risultati delle sole top-k simple typicality query.
- Misurazioni effettuate su una macchina
  - Pentium IV 3,0 GHz
  - 1 GB RAM
  - HDD 160 GB
  - Windows XP
- Gli algoritmi di test sono stati implementati in C++.

Risultati

- Di default, il numero di tuple è 10000, la dimensionalità è di 5 attributi e le query sono di tipo top-10.
- In caso di local typicality, la soglia di vicinato è impostata di default a 2h (con h larghezza di banda del kernel Gaussiano).
- Nel metodo Randomized Tournament, di default la dimensione dei gruppi è 10 e vengono eseguite 4 validazioni aggiuntive.
- Si è osservato che, nonostante la qualità di RT debba aumentare aumentando il numero di validazioni, il miglioramento oltre i 3 round aggiuntivi è piuttosto modesto.

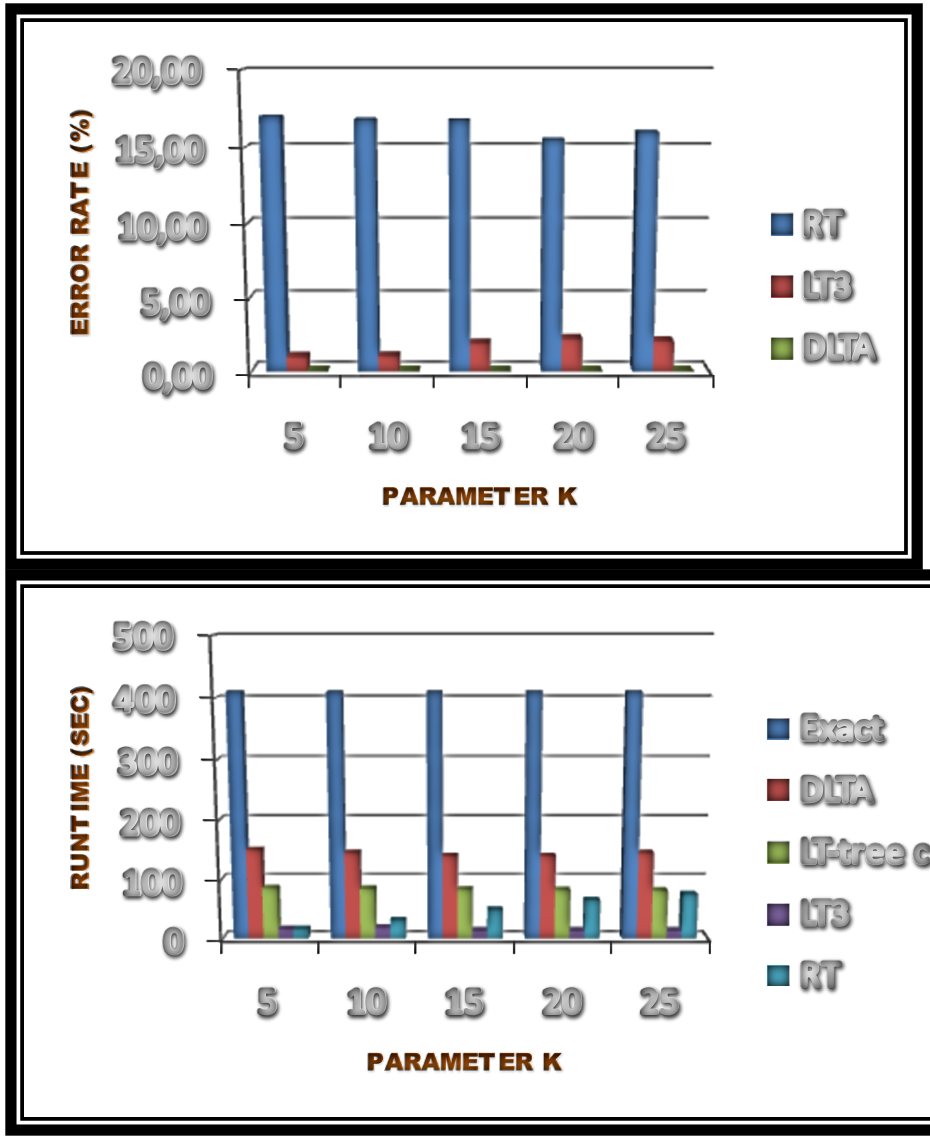
Risultati – Variazione soglia vicinato





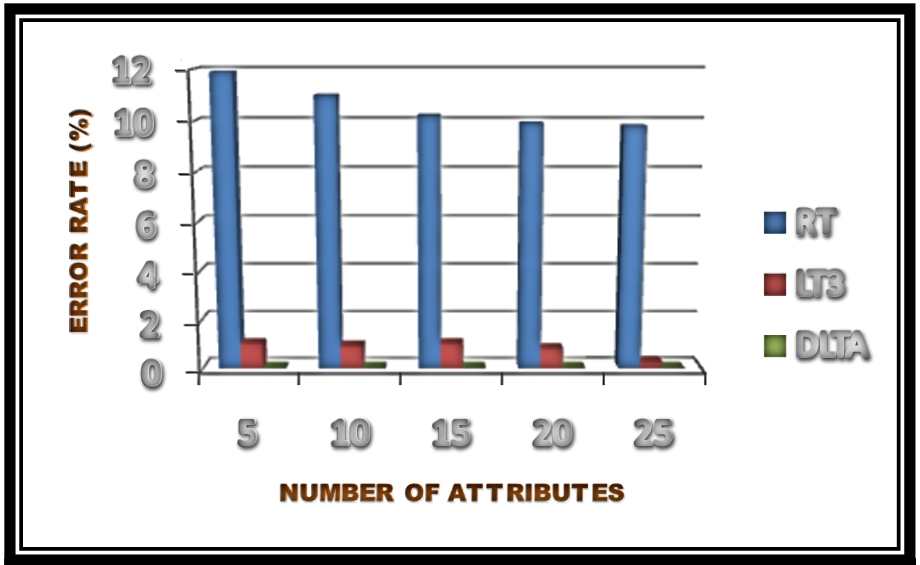
- LT3 si dimostra una valida alternativa a DLTA all'aumentare del valore di soglia per la ricerca del σ vicinato, nonostante il peggior tasso d'errore.
- NB: la voce "LT-tree c" (qui e nel seguito) riportata nel grafico si riferisce alla costruzione dell'albero LT.

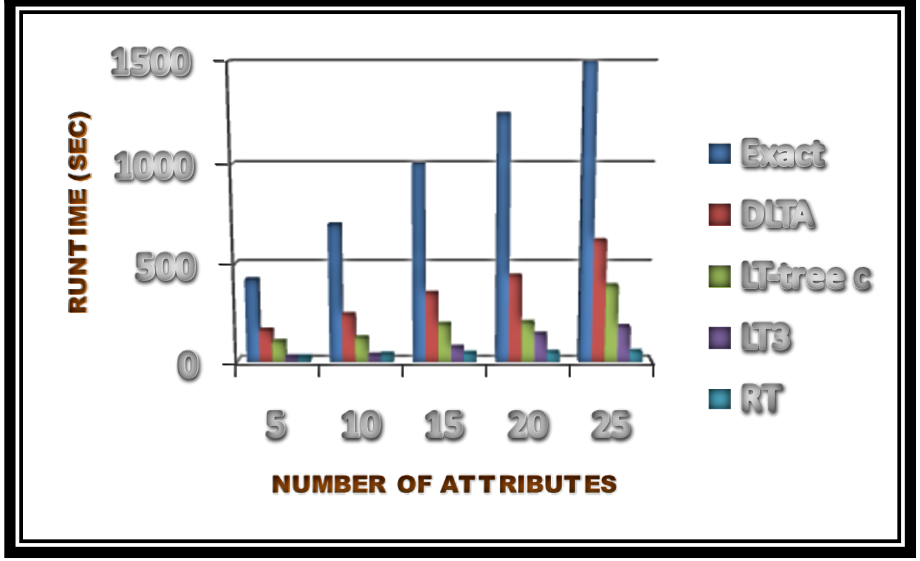
Risultati – Variazione di k



- Al variare di k, DLTA e LT3 mantengono prestazioni pressoché invariate e simili. Questo perché si "riciclano" tornei già eseguiti.
- RT, nonostante sia il più veloce ad eseguire, è anche il metodo col più alto tasso d'errore.

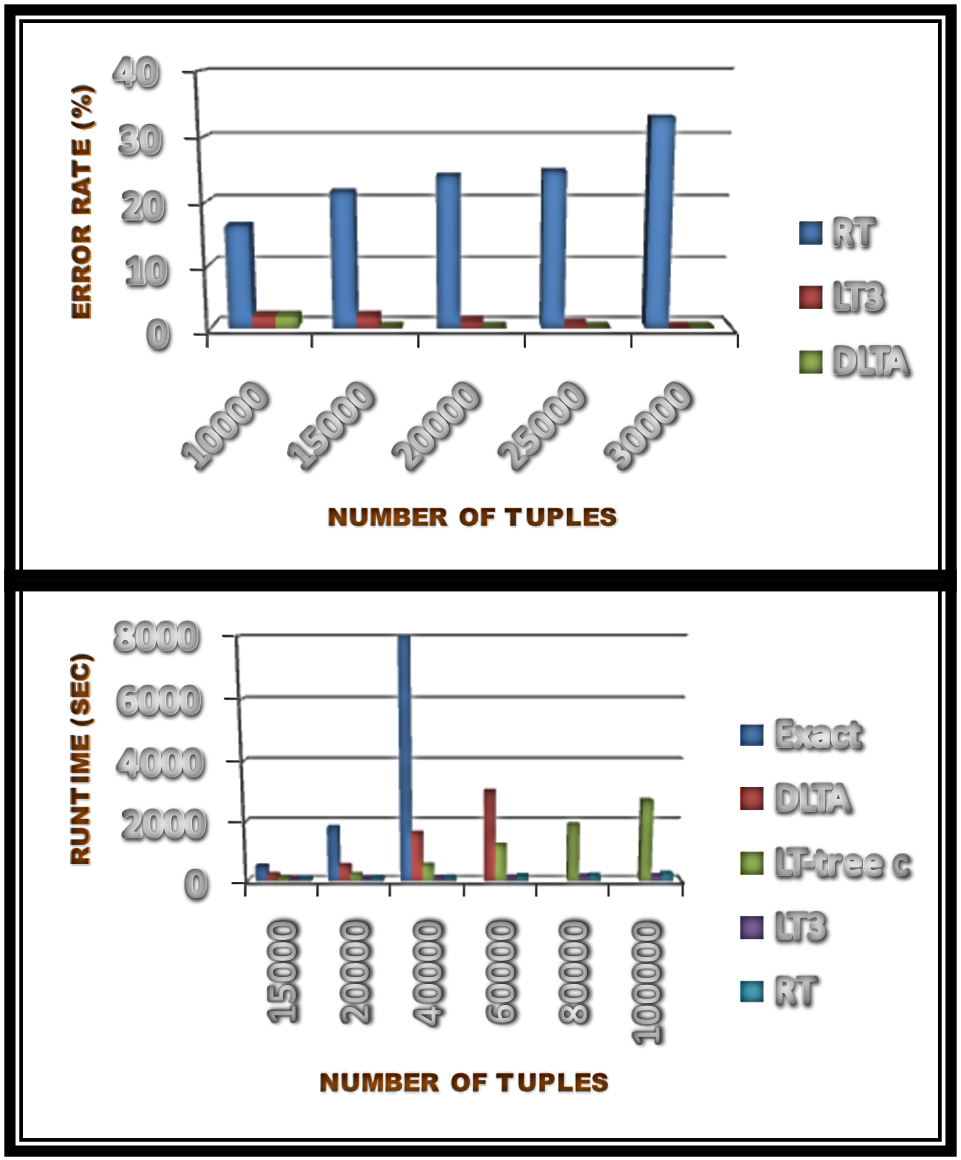
Risultati – Variazione numero attributi





- Aumentando il numero di attributi, le prestazioni in termini di runtime di DLTA ed LT3 degradano lentamente fino ad avere un costo sempre più consistente per la costruzione del LT3.
- Aumentando la dimensionalità oltre i 25 attributi, si potrebbe preferire DLTA a LT3.

Risultati – Variazione numero tuple



- Aumentando il numero di tuple oltre le 80000 unità, il metodo LT3 risulta l'unico ragionevolmente applicabile al data set.
- Il tasso di errore di LT3 cala perché i dati, essendo più numerosi, sono anche più densi, rendendo più efficiente la tipicità locale.

Risultati - Conclusioni

- Dato che RT ha complessità lineare, esso dovrebbe essere scelto quando il tempo di esecuzione è vitale.
- Nonostante DLTA e LT3 siano molto più scalabili dell'algoritmo esatto e più accurati di RT, essi sono ottimali per situazioni nelle quali è richiesto un certo compromesso tra accuratezza e prestazioni.
- LT3 ha efficienza e scalabilità mogliori di DLTA, raggiungendo, in alcuni casi, un'accuratezza paragonabile a quest'ultimo.