Hierarchical hash tree

Luigi Foscari

Si vuole creare un dizionario che permette le operazioni di inserimento, cancellazione e ricerca. I valori da inserire nella struttura appartengono ad un insieme $\mathcal{X}$ per cui esistono k feature map $f_i : \mathcal{X} \mapsto \mathcal{H} \quad i = 1, \dots, k$ ed esistono t funzioni di hash $h_{i,b} : \mathcal{H} \mapsto \{0, \dots, b-1\} \quad i = 1, \dots, t$.

La struttura è organizzata come un albero, in cui ad ogni nodo corrisponde una lista di collisione o una tabella di dimensione b, una delle mappe di feature e una funzione di hash, nelle celle della tabella sono presenti puntatori a nodi figli. La struttura è inizializzata con un singolo nodo composto da solo un bucket.

La procedura di inserimento per un elemento $x \in \mathcal{X}$ in un noto y avviene nel seguente modo

- Se y è una lista di collisione x è aggiunto in testa.
- Se y è una tabella con feature map f e funzione di hash h, si ripete l'inserimento ricorsivamente nel nodo puntato dalla posizione $h(f(x))$ della tabella.

Se un bucket in un nodo t raggiunge un limite massimo[1] viene **scisso**, l'operazione di scissione controlla tutti gli elementi nel bucket e trova la funzione di hash che distribuisce più equamente i valori; scelta una funzione h_b e una feature map f, viene poi creata una tabella di taglia b e i valori del bucket sono inseriti in questa tabella.

L'intera struttura dati può fare a meno delle feature map se si utilizzano funzioni di hash che hanno come dominio $\mathcal{U}$, ma questa opzione è meno interessante perché l'estrazione di feature permette di operare in modo più espressivo sui dati. Esempio pratico: l'universo contiene informazioni su libri, una feature che si è scelto di estrarre è la data di pubblicazione, si possono quindi cercare tutti i testi usciti nel medesimo anno in tempo efficiente.

Senza scendere nei dettagli la struttura dati dell'albero è espressa ricorsivamente in questo modo

```
type 'a tree =
| Leaf of 'a list
| Node of 'a tree array
```

[1] Bisogna fare ulteriori considerazioni su questo valore

Dato un bucket di elementi bisogna trovare la funzione di hash tra quelle a disposizione che ottiene i valori più sparsi, è possibile utilizzare l'indice di Gini normalizzato, indichiamo con x i valori nel campione e con ϕ le frequenze relative:

$$\{x_1, \ldots, x_n\} \to \{\phi_1, \ldots, \phi_k\} \qquad I = \left(1 - \sum_{i=0}^{k} \phi_i\right) \frac{k}{k-1}$$

Un algoritmo ingenuo che utilizza questo indice è il seguente, assumiamo per semplicità che ad ogni mappa di feature sia associata una e una sola funzione di hash

Algorithm 1: Algoritmo per la scelta della funzione di hash

Input: B bucket, $f_1, \ldots, f_k$ feature maps, $h_1, \ldots, h_k$ funzioni di hash
Output: Una feature map

Sia M una matrice $k \times |B|$
for $i = 0, \ldots, k-1$ **do**
  for $j = 0, \ldots, |B|-1$ **do**
    $M[i][j] \leftarrow h_i(f_i(B_j))$
  end
end
Sia G un array di taglia k
for $i = 0, \ldots, k-1$ **do**
  $G[i] \leftarrow$ indice di Gini normalizzato della i-esima riga di M
end
$r \leftarrow$ indice dell'elemento più grande di G
return f_r

Sarebbe anche possibile definire le feature map in modo che abbiamo ognuna un codominio distinto, ma questo renderebbe necessario definire almeno una funzione di hash per ognuna di essere, rendendo più complicato il procedimento di scelta delle feature.

Si può rendere l'algoritmo più efficiente? Anche nel caso in cui si rimuova il rilassamento.