Verslag Practicum 1

Martijn Koenis (3770214) Jordi Vermeulen (3835634)

Samenvatting

In dit verslag presenteren we onze bevindingen bij het maken van het eerste practicum van Data-analyse en retrieval, waarvoor een systeem geïmplementeerd moest worden dat de resultaten van een query automatisch rankt en de beste k resultaten teruggeeft. Wij beschrijven de exacte aanpak die wij gebruikt hebben en lichten toe waarom we die manier gekozen hebben.

1 Aanpak

We hebben ervoor gekozen om geavanceerde technieken te gebruiken voor het berekenen van de gelijkenis tussen waarden en voor het berekenen van de top-k. Voor het berekenen van *inverse document frequencies* en *query frequencies* hebben we gebruik gemaakt van de technieken beschreven in het artikel 'Automated ranking of database query results' van Agrawal, Chaudhuri e.a.

1.1 IDF

Voor de berekening van de IDF (*inverse document frequency*) van categorische waarden gebruiken we de volgende formule:

$$IDF_k(t) = log\left(\frac{N}{df_k(t)}\right)$$

met $df_k(t)$ het aantal voorkomens van waarde t voor een attribuut k in de database en N het totaal aantal rijen in de database. De IDF's hebben wij berekend door naar alle rijen in de database te kijken en voor elk attribuut bij te houden hoe vaak elke waarde voorkomt. Als je alle rijen hebt bekeken is het eenvoudig uit te rekenen wat de IDF voor elke waarde van elk attribuut is en deze op te slaan in de metadatabase. Voor de definitie van de tabel waarin de IDF's worden opgeslagen, zie metadb.txt.

Voor de inverse document frequency van numerieke waarden gebruiken we de volgende formule:

$$IDF_k(t) = log\left(\frac{n}{\sum_i^n e^{-\frac{1}{2}\left(\frac{t_i - t}{h}\right)^2}}\right)$$

met n het aantal rijen in de database en h de *bandwidth parameter*, welke berekend kan worden met de volgende formule:

$$h = 1.06\,\sigma\,n^{-\frac{1}{5}}$$

waarin σ de standaardafwijking tussen alle waarden van het attribuut is.

Deze formules hebben wij geïmplementeerd door voor elk numeriek attribuut een aantal samples te nemen. Zo berekenen we voor *displacement* de reeks 50, 60, 70 ... 490, 500. Vervolgens bereken we eerst h en daarna voor elke getal in de reeks de IDF. Zowel de h als de IDF worden opgeslagen in de metadatabase.

1.2 QF

Voor de berekeningen van de QF (*query frequencies*) gaan we eerst de hele workload doorlopen. Hierbij slaan we per attribuut op hoe vaak welke waarde gevraagd is. Ook slaan we meteen op hoe vaak waarden van attributen samen in een IN-statement voorkomen om later te gebruiken voor het bereken van de Jaccard-coëfficiënt.

Voor het berekenen van de QF van categorische attributen gebruiken we de volgende formule:

$$QF_k(q) = \frac{RQF_k(q)}{RQFMax_k}$$

hierin is $RQF_k(q)$ het aantal voorkomens van attribuut k met waarde q in de workload en $RQFMax_k$ het maximum van alle RQF_k's.

Voor het berekenen van de QF-waarden voor numerieke attributen gebruiken we dezelfde formule, alleen met een andere definitie voor $RQF_k(q)$:

$$RQF_k(q) = \sum_{i}^{n} e^{-\frac{1}{2}\left(\frac{q_i - q}{h}\right)^2}$$

Hierin is n het totaal aantal queries in de workload. q_i is de waarde van het attribuut k in query i en q is de gevraagde waarde voor het attribuut. h wordt berekend met dezelfde formule als bij numerieke IDF, alleen wordt nu de standaardafwijking genomen over de queries in de workload in plaats van over de rijen in de database. Wederom worden zowel de h-waarde als de QF opgeslagen in de metadatabase.

1.3 Jaccard-coëfficiënt

Het berekenen van de Jaccard-coëfficiënt wordt de volgende formule gebruikt:

$$J\left(W(t), W(q)\right) = \frac{|W(t) \cap W(q)|}{|W(t) \cup W(q)|}$$

Hierin is $W(x)$ de set IN statements uit de workload waar waarde x in staat. De functie is dus het aantal IN-statements waarin t en q samen voorkomen gedeeld door het aantal IN-statements waarin t of q voorkomt. Voor de implementatie van de berekening hebben we de formule omgeschreven naar de equivalente formule:

$$J\left(W(t), W(q)\right) = \frac{|W(t) \cap W(q)|}{|W(t)| + |W(q)| - |W(t) \cap W(q)|}$$

Ook deze waarden worden opgeslagen in de metadatabase.

1.4 Scores

Tijdens het opvragen van een query worden scores berekend die de gelijkenis tussen de opgevraagde query en een rij in de database uitdrukken. Omdat we voor de numerieke waarden samples hebben genomen wordt de invoer van numerieke waarden eerst afgerond naar de dichtstbijzijnde sample. De IDF-score voor gelijkenis van categorische waarden wordt berekend met de volgende formule:

$$IDFSim_k(q,t) = \begin{cases} IDF_k(q), & \text{if } q = t \\ 0, & \text{otherwise} \end{cases}$$

Hierin wordt $IDF_k(q)$ gewoon uit de metadatabse opgevraagd.

De IDF-score voor gelijkenis voor numerieke waarden wordt berekend met de volgende formule:

$$IDFSim_k(q,t) = e^{-\frac{1}{2}\left(\frac{t-q}{h}\right)^2} \times IDF_k(q)$$

Hierin worden h en $IDF_k(q)$ uit de metadatabse opgevraagd.

De QF-score voor categorische waarden, waarvoor een Jaccard-coëfficiënt berekend is, wordt berekend met de volgende formule:

$$QFSim_k(q,t) = J(W(q), W(t)) \times QF_k(q)$$

Waarin zowel $J(W(q), W(t))$ en $QF_k(q)$ uit de metadatabase worden opgevraagd. De Jaccard-coëfficiënt is altijd 1 als $q = t$, en 0 als er geen waarde bekend is.

Voor het berekenen van de QF-scores voor numerieke waarden wordt de volgende formule gebruikt:

$$QFSim_k(q,t) = e^{-\frac{1}{2}\left(\frac{t-q}{h}\right)^2} \times QF_k(q)$$

Waarin $QF_k(q)$ en h worden opgevraagd uit de metadatabase.

De eindscore van een query met waarden $Q = q_1 \dots q_n$ met een rij in de database met waarden $T = (t_1 \dots t_n)$ wordt berekend met:

$$S(Q,T) = \sum_i^n \big(IDFSim_k(q,t) + QFSim_k(q,t)\big)$$

1.5 Top-k

Voor het berekenen van de top-k resultaten gebruiken we Fagin's algoritme met *sorted* en *random access*. Het algoritme neemt als invoer twee lijsten en een getal k. De eerste lijst bevat voor ieder attribuut in de query twee lijsten van ID's uit de database, gesorteerd op de IDFSim, respectievelijk QFSim. De tweede lijst bevat *dictionaries* van ID naar score. Deze twee structuren worden gebruikt voor *sorted* respectievelijk *random access*. Het algoritme geeft vervolgens een lijst van lengte k terug (of n als $k > n$) met daarin tupels van ID en totale score. Deze lijst wordt gebruikt om voor ieder ID de volledige record uit de database op te vragen en weer te geven in het uiteindelijke resultaat.

2 Ontwerpkeuzes

We zijn in ons ontwerp weinig afgeweken van het proces zoals beschreven in het paper *Automated ranking of database query results* van Agrawal, Chaudhuri e.a. We hebben hiervoor gekozen omdat we ervan uitgaan dat de schrijvers van dat artikel weten waar ze het over hebben, en wij onszelf niet in staat achtten om iets beters te verzinnen. Ook is de aanpak vrij simpel en goed te implementeren, terwijl alternatieve manieren die in andere papers beschreven worden al snel vrij complex zijn.

We hebben binnen het redelijke zo veel mogelijk geprobeerd om de snelheid van de daadwerkelijke queries te verbeteren. Zo slaan we alle data die tussen meerdere queries gebruikt zou kunnen worden op in de metadatabase; niet alleen de IDF-, QF- en Jaccard-scores, maar bijvoorbeeld ook de *bandwidth parameter* h. Hierdoor komt het uitvoeren van een zoekopdracht simpelweg neer op het ophalen van de scores van de betreffende attributen uit de database, om daar vervolgens het top-k algoritme op toe te passen.

Een ander belangrijke ontwerpkeuze is het gebruiken van sampling om numerieke QF- en IDF-functies op te slaan. Het is namelijk onmogelijk om voor alle mogelijk waarden de resultaten van deze functies op te slaan, omdat ze continu zijn. In het artikel van Agrawal en Chaudhuri wordt verwezen naar de *WARPing* methode. Aangezien dit ook op een histogram gebaseerd is, lijkt het redelijk op onze aanpak, alleen zijn de *bins* in ons geval handmatig gekozen voor iedere parameter.

3 Ervaringen

We hebben slechts kleine aanpassingen aan ons ontwerp moeten aanbrengen om tot een werkend programma te komen. De grootste aanpassing was het opslaan van de *bandwidth parameters*, waar we in ons ontwerp helemaal geen rekening mee hadden gehouden. Daarnaast hebben we tegen ons ontwerp in geen aparte module gemaakt voor het afhandelen van de database-interacties. Het programma was misschien wat mooier geworden als we dit wel hadden gedaan, maar het implementeren van een *data abstraction layer* is weer een hele opgave op zich, en valt volgens ons niet echt binnen de scope van deze opgave.

We hadden ook nog enige moeite met het implementeren van de numerieke QF-functie. Deze wordt niet beschreven in het artikel, maar na enig gepuzzel en hulp van de practicumassistent is het toch gelukt om een aangepaste versie van de numerieke IDF-functie te gebruiken.

Als laatste hadden we nog wat problemen met het testen van ons programma. De waarden die uit de functies komen zijn niet echt inzichtelijk, wat het moeilijk maakt om te controleren of de resultaten correct zijn. Uiteindelijk hebben we dit opgelost door de kleinere berekeningen op een paar voorbeelden na te rekenen, en voor het programma als geheel een bepaalde query te geven en daar dan een extra attribuut aan toe te voegen. Het was dan redelijk goed in te schatten of de nieuwe resultaten ongeveer de verschillen lieten zien die je zou verwachten.