Μάθημα: Διαχείριση μη Παραδοσιακών Δεδομένων

Εργασία 3: Συνολοδεδομένα

Ονοματεπώνυμο: Κωνσταντίνος Βαρδάκας

ΑΜ: 522

Email: pcs0522@uoi.gr

Μέρος 1: Containment Queries

Στο πρώτο μέρος της εργασίας υλοποιήθηκαν και αξιολογήθηκαν διάφορες μεθοδολογίες για την αποτίμηση containment queries. Ο σχετικός κώδικας έχει οργανωθεί σε δύο αρχεία, το αρχείο containment_functions.py, το οποίο περιέχει τις βοηθητικές συναρτήσεις, και το αρχείο στο containment_main.py, το οποίο περιλαμβάνει το κύριο πρόγραμμα. Αρχικά τα αρχεία transactions.txt και queries.txt διαβάζονται μέσω της συνάρτησης read_file(filename), η οποία επιστρέφει μία λίστα από σύνολα. Κάθε σύνολο περιλαμβάνει ακέραιες τιμές που αντιστοιχούν στα id αντικειμένων που βρίσκονται είτε σε δεδομένο transaction, είτε σε δεδομένο query.

Naive method

Η πρώτη μέθοδος containment query που υλοποιήθηκε ήταν μια απλή μέθοδος αναφοράς, η naive_method(), για τη σύγκριση της απόδοσης ως προς τις υπόλοιπες. Κατά τη συγκεκριμένη συνάρτηση, για κάθε transaction ελέγχεται αν το query αποτελεί υποσύνολο των αντικειμένων της συναλλαγής. Εφόσον είναι, προστίθεται το id της συναλλαγής στην λίστα των αποτελεσμάτων.

Exact signature file

Η δεύτερη μέθοδος που υλοποιήθηκε βασίζεται στη χρήση bitmaps (signature files). Για κάθε συναλλαγή δημιουργείται ένας bitvector (ακέραιος αριθμός), ο οποίος αρχικοποιείται με την τιμή 0. Στη συνέχεια, για κάθε αντικείμενο που περιέχεται στη συναλλαγή, ενεργοποιείται το αντίστοιχο bit του vector χρησιμοποιώντας bitwise

τελεστές. Η αρίθμηση των bits γίνεται από τα δεξιά προς τα αριστερά (δηλαδή από το least significant bit προς το most significant), εφαρμόζοντας τον τελεστή μετατόπισης (1 << item). Παράλληλα ο τελεστής OR, χρησιμοποιείται για να διατηρούνται όλα τα bits που έχουν ενεργοποιηθεί σε προηγούμενες επαναλήψεις της λούπας.

Στη συνέχεια, η συνάρτηση to_sigfile(transactions) χρησιμοποιεί την συνάρτηση bitmap, για όλα τα transactions δημιουργώντας το signature file, το οποίο αποθηκεύεται χρησιμοποιώντας την συνάρτηση save_file(list_to_save, save_name). Το containment query πραγματοποιείται με τη συνάρτηση signature_file(query_bin, transactions_bin), όπου για κάθε transaction, ελέγχεται εάν τα bits που είναι ενεργοποιημένα στο query είναι επίσης ενεργά και στο αντίστοιχο bitvector της συναλλαγής. Αυτός ο έλεγχος πραγματοποιείται με το bitwise AND μεταξύ του query και του transaction, και σύγκριση του αποτελέσματος του AND με το αρχικό query.

Exact bitslice signature file

Η τρίτη μέθοδος που υλοποιήθηκε είναι η bitslice signature file, όπου αντί να δημιουργείται ένας bitvector για κάθε συναλλαγή, δημιουργείται για κάθε αντικείμενο. Κάθε τέτοιος bitmap έχει θέσεις ισάριθμες με το πλήθος των συναλλαγών και έχει τιμή 1 στις θέσεις που αντιστοιχούν σε συναλλαγές που υπάρχει αυτό το αντικείμενο. Η δημιουργία της δομής bitslice υλοποιείται με τη συνάρτηση bitslice(transactions) η οποία αρχικοποιεί μία λίστα με μηδενικά, τα οποία είναι τα αρχικά bitvector κάθε αντικειμένου. Για κάθε αντικείμενο, ενεργοποιείται το bit των συναλλαγών που το περιέχουν, με τον ίδιο τρόπο όπως και προηγουμένως στην συνάρτηση bitmap με χρήση του τελεστή μετατόπισης (1 << item) για την είσοδο της μονάδας από δεξιά, και χρήση του τελεστής OR με σκοπό να διατηρούνται τα bit που έχουν εισαχθεί σε προηγούμενη επανάληψη της λούπας. Έτσι κατασκευάζεται μία λίστα, όπου κάθε θέση αντιστοιχεί σε ένα αντικείμενο και περιέχει έναν ακέραιο αριθμό που αναπαριστά το bitvector (bitslice) του αντικειμένου. Η λίστα αυτή αποθηκεύεται στο αρχείο bitslice.txt μέσω της συνάρτησης save_enu_file, με κάθε γραμμή να περιλαμβάνει το ID του αντικειμένου και τον αντίστοιχο ακέραιο bitvector που το περιγράφει.

Στη συνέχεια για την αποτίμηση containment queries, χρησιμοποιήθηκε η συνάρτηση bitsliced_signature_file(query, bitslice_trans) η οποία δέχεται το output της

προηγούμενης συνάρτησης, δηλαδή τα bitmaps των αντικειμένων και το query και πραγματοποιεί το λογικό AND μεταξύ των αντικειμένων του query. Αυτό πραγματοποιείται συγκρίνοντας τον bitvector του πρώτου αντικειμένου με τα υπόλοιπα αντικείμενα του query, και το τελικό αποτέλεσμα, διαθέτει μονάδα στις θέσεις των συναλλαγών που περιέχουν όλα τα αντικείμενα του query. Τέλος, εξάγονται αυτές οι θέσεις των συναλλαγών σε μία λίστα η οποία και επιστρέφει από την συνάρτηση.

Inverted file

Η τέταρτη και τελευταία μέθοδος βασίζεται στη δημιουργία ενός inverted file. Για κάθε αντικείμενο που εμφανίζεται στις συναλλαγές, κατασκευάζεται μια λίστα που περιέχει τα IDs των συναλλαγών στις οποίες το αντικείμενο αυτό συμμετέχει. Οι λίστες αυτές είναι ταξινομημένες και αποθηκεύονται σε ένα λεξικό, όπου το κλειδί είναι το ID του αντικειμένου και η τιμή είναι η αντίστοιχη λίστα των συναλλαγών. Η δομή αυτή δημιουργείται με τη συνάρτηση build_inverted_file().

Για την αποτίμηση ενός containment query, χρησιμοποιείται η συνάρτηση inverted_file(query, inverted_index), η οποία για κάθε αντικείμενο του ερωτήματος προσθέτει τις λίστες συναλλαγών από το λεξικό inverted index που δημιουργήθηκε προηγουμένως. Οι λίστες αυτές συγκεντρώνονται και στη συνέχεια υπολογίζεται η τομή τους με χρήση του αλγορίθμου συγχώνευσης για ταξινομημένες λίστες με τη συνάρτηση merge_intersection. Στην συνάρτηση merge_intersection, αρχικά συγκρίνονται γραμμικά (δεδομένου ότι είναι sorted) οι δύο πρώτες λίστες που αντιστοιχούν στις συναλλαγές που συμμετέχουν τα δύο πρώτα αντικείμενα, όμοια με τον αλγόριθμο intersection sorted. Στη συνέχεια οι κοινές συναλλαγές παραμένουν στη λίστα result που αρχικά ήταν το πρώτο αντικείμενο και, συγκρίνονται με το τρίτο αντικείμενο. Αυτή η διαδικασία επαναλαμβάνεται για κάθε αντικείμενο και διατηρούνται, όμοια με την προηγούμενη μέθοδο, η συναλλαγές που είναι κοινές για κάθε αντικέιμενο. Εν τέλει, τα αποτελέσματα αποθηκεύονται με την χρήση της συνάρτησης save_inverted_file, με ταξινόμηση των keys του λεξικού.

Κύριο πρόγραμμα

Στο κύριο πρόγραμμα χρησιμοποιείται η συνάρτηση time_method για μορφοποίηση και χρονομέτρηση των αποτελεσμάτων κάθε μεθοδολογίας. Η χρονομέτρηση γίνεται με την συνάρτηση perf_counter() αντί της time() λόγω μεγαλύτερης ακρίβειας και η χρονομέτρηση σε κάθε περίπτωση αφορά το χρόνο του που χρειάζεται για να εξαχθούν τα αποτελέσματα του query, και όχι η προεπεξεργασία που χρειάζεται να πραγματοποιηθεί στα δεδομένα (π.χ. εξαγωγή αρχείων sigfile, bitslice και invfile). Σε κάθε περίπτωση αρχικά δημιουργούνται και αποθηκεύονται αυτά τα αρχεία και το query πραγματοποιείται με την μεθοδολογία που επιλέγεται βάση της τιμής του argument method. Επίσης το query πραγματοποιείται με τη συνάρτηση map, καθώς κάθε συνάρτηση query που υλοποιήθηκε αφορά ένα μεμονωμένο query. Έτσι, εφόσον το query είναι ένα, μετατρέπεται σε λίστα για να εξαχθούν τα αποτελέσματα με map.

Μέρος 2: Relevance Queries

Στο δεύτερο μέρος της εργασίας υλοποιήθηκαν μέθοδοι για την αποτίμηση ερωτημάτων σχετικότητας (relevance queries), λαμβάνοντας υπόψη τη συχνότητα εμφάνισης των αντικειμένων σε κάθε συναλλαγή (bag semantics) και τη σπανιότητα των αντικειμένων στο σύνολο των συναλλαγών. Ο κώδικας οργανώθηκε όμοια, στο αρχείο containment_functions.py, το οποίο περιλαμβάνει τις βοηθητικές συναρτήσεις, και στο αρχείο containment_main.py, το οποίο περιλαμβάνει το κύριο πρόγραμμα. Το αρχείο transactions.txt διαβάζεται μέσω της συνάρτησης read_file(filename), η οποία επιστρέφει μία λίστα από λίστες διατηρώντας τις πολλαπλές εμφανίσεις των αντικειμένων.

Δημιουργία Inverted File με Occurrences

Για την υλοποίηση των relevance queries, βασική απαίτηση αποτελεί η κατασκευή μιας τροποποιημένης δομής inverted file λαμβάνοντας πλέον υπόψη και τον

αριθμό εμφανίσεων κάθε αντικειμένου σε κάθε συναλλαγή καθώς επίσης και την κατασκευή μίας δομής που να διατηρεί τη σχετική σπανιότητά κάθε αντικειμένου στο σύνολο των συναλλαγών. Η συνάρτηση build_inverted_file_occ είναι υπεύθυνη για τη δημιουργία των δύο αυτών δομών σε μορφή λεξικών, όπου αρχικοποιούνται ως κενά λεξικά.

Η διαδικασία ξεκινά με την επεξεργασία κάθε συναλλαγής ξεχωριστά. Για κάθε συναλλαγή διατηρείται αρχικά ένα προσωρινό λεξικό συχνοτήτων, το occ_dict, το οποίο καταγράφει πόσες φορές εμφανίζεται κάθε αντικείμενο μέσα σε αυτή τη συναλλαγή. Μόλις υπολογιστεί το occ_dict, τα περιεχόμενά του προστίθονται στο λεξικό inverted_index. Συγκεκριμένα, για κάθε αντικείμενο που υπάρχει στο occ_dict, προστίθεται στο inverted_index ένα ζεύγος της μορφής (τ, occ(i,τ)), όπου τ είναι το transaction ID και occ(i,τ) είναι ο αριθμός εμφανίσεων του αντικειμένου i σε αυτό το transaction.

Παράλληλα, για κάθε αντικείμενο που εμφανίζεται σε αυτή τη συναλλαγή, προστίθεται μόνο μία εμφάνιση στο λεξικό trf, ώστε να μετρηθεί σε πόσες διαφορετικές συναλλαγές συνολικά συμμετέχει το κάθε αντικείμενο. Σημαντικό είναι ότι η ενημέρωση του trf βασίζεται στο occ_dict, δηλαδή κάθε αντικείμενο μετριέται μόνο μία φορά ανά συναλλαγή, ανεξαρτήτως πόσες φορές εμφανίζεται μέσα σε αυτή.

Αφού ολοκληρωθεί η επεξεργασία όλων των συναλλαγών και έχουν ενημερωθεί τα δύο λεξικά (inverted_index και trf), υπολογίζεται η τελική τιμή της σχετικής σπανιότητας κάθε αντικειμένου μέσω του λεξικού idf. Για κάθε αντικείμενο εφαρμόζεται ο τύπος |T| / trf(i,T), όπου |T| είναι το πλήθος των συναλλαγών. Το τελικό αποτέλεσμα της συνάρτησης είναι το inverted_index, που περιέχει για κάθε αντικείμενο τις σχετικές συναλλαγές και τις εμφανίσεις του, και το idf, που εκφράζει πόσο σπάνιο είναι το κάθε αντικείμενο με βάση τη συμμετοχή του στο σύνολο των συναλλαγών.

Inverted file

Πλέον είναι δυνατή η υλοποίηση της διαδικασίας αποτίμησης ερωτημάτων relevance, με τη χρήση των δύο λεξικών που δημιουργηθηκαν. Για δεδομένο query, για κάθε αντικείμενο, αναζητούνται οι αντίστοιχες λίστες από το λεξικό inverted_index.

Κάθε τέτοια λίστα, όπως αναφέρθηκε, περιέχει ζεύγη της μορφής [transaction_id, occ(i, τ)]. Για τον υπολογισμό της ένωσης των συναλλαγών που περιλαμβάνουν τουλάχιστον ένα από τα αντικείμενα του ερωτήματος, χρησιμοποιείται η συνάρτηση merge_union. Αυτή η συνάρτηση, εφαρμόζει έναν merge αλγόριθμο σε ταξινομημένες λίστες ζευγών [tid, occ]. Κατά την επεξεργασία, διατηρείται ένας δείκτης (pointer) για κάθε λίστα και συγκρίνεται το τρέχον transaction_id που δείχνει κάθε δείκτης. Σε κάθε βήμα, εντοπίζεται το μικρότερο tid μεταξύ όλων των δεικτών και συλλέγονται όλες οι εμφανίσεις (occ) των αντικειμένων για αυτό το tid, από όσες λίστες περιέχουν το συγκεκριμένο tid. Αν ένα αντικείμενο δεν εμφανίζεται στο tid, προστίθεται 0 στη θέση του. Έτσι, παράγεται μία λίστα από ζεύγη της μορφής [tid, occ_list], όπου occ_list είναι μια λίστα με τη συχνότητα εμφάνισης του κάθε αντικειμένου στο tid.

Στη συνέχεια, στη συνάρτηση inverted_file_method, για κάθε tid που προέκυψε από την ένωση, υπολογίζεται η συνάρτηση σχετικότητας rel(τ, q), η οποία ορίζεται ως:

$$rel(\tau, q) = \sum_{i \in q} \left(occ(i, \tau) \cdot \frac{|T|}{\text{trf}(i, T)} \right)$$

Δηλαδή, για κάθε αντικείμενο i του ερωτήματος που εμφανίζεται στη συναλλαγή τ, υπολογίζεται το γινόμενο του πλήθους εμφανίσεών του στη συναλλαγή (occ(i, τ)) με τη σπανιότητά του στο σύνολο των συναλλαγών (η οποία έχει υπολογιστεί εκ των προτέρων ως idf[i] = |T| / trf(i,T)). Το συνολικό rel προστίθεται στα αποτελέσματα μόνο αν υπάρχει κάποια σχετικότητα (rel > 0), και καταχωρείται μαζί με το transaction_id στα αποτελέσματα. Τέλος, τα αποτελέσματα ταξινομούνται κατά φθίνουσα τιμή σχετικότητας και επιστρέφονται τα k πιο σχετικά transaction_id, ή όλα όσα υπάρχουν εφόσον είναι λιγότερα από k.

Naive method

Η συνάρτηση naive_relevance_query υλοποιεί μια απλή προσέγγιση για την αποτίμηση της σχετικότητας ενός ερωτήματος ως προς το σύνολο των συναλλαγών, χωρίς την χρήση του λεξικού inverted_index, αλλά εξετάζοντας μία προς μία όλες τις συναλλαγές που περιέχονται στη λίστα transactions.

Στην αρχή της συνάρτησης, δημιουργείται ένα κενό λεξικό relevance_scores, το οποίο χρησιμοποιείται για να αποθηκεύσει τη σχετικότητα κάθε συναλλαγής που περιέχει τουλάχιστον ένα αντικείμενο του query. Για κάθε συναλλαγή στη λίστα transactions, κατασκευάζεται ένα λεξικό occ_dict, το οποίο καταγράφει πόσες φορές εμφανίζεται κάθε αντικείμενο μέσα στη συγκεκριμένη συναλλαγή, όμοια με το occ_dict της συνάρτησης build_inverted_file_occ. Αφού υπολογιστεί το πλήθος εμφανίσεων των αντικειμένων στη συναλλαγή, ξεκινά ο υπολογισμός της συνάρτησης σχετικότητας rel(τ, q) για το συγκεκριμένο query. Η μεταβλητή rel αρχικοποιείται ως μηδέν και για κάθε αντικείμενο του query ελέγχεται εάν υπάρχει στο occ_dict και επίσης αν υπάρχει στο λεξικό idf. Αν και οι δύο συνθήκες ισχύουν, τότε προστίθεται στο score το γινόμενο occ_dict[item] * idf[item].

Εφόσον η σχετικότητα για τη συναλλαγή δεν είναι μηδενική, καταχωρείται στο λεξικό relevance_scores με κλειδί το tid της συναλλαγής και τιμή τη σχετικότητα. Αφού υπολογιστεί η σχετικότητα για όλες τις συναλλαγές, τα αποτελέσματα μετατρέπονται σε λίστα ζευγών [score, tid] και ταξινομούνται κατά φθίνουσα σειρά ως προς τη σχετικότητα. Τέλος, η συνάρτηση επιστρέφει τις k συναλλαγές με τη μεγαλύτερη σχετικότητα ή όλες όσες υπάρχουν αν είναι λιγότερες από k.

Κύριο πρόγραμμα

Όμοια με το πρώτο μέρος χρησιμοποιείται η μέθοδος time_method, το query πραγματοποιείται με τη συνάρτηση map, οπότε εάν το query είναι συγκεκριμένο, πρώτα μετατρέπεται σε λίστα. Σε κάθε περίπτωση αρχικά δημιουργείται και αποθηκεύεται το αρχείο invfileocc.txt και το query πραγματοποιείται με την μεθοδολογία που επιλέγεται βάση της τιμής του argument method.

Οδηγίες εκτέλεσης

Εκτέλεση πρώτου μέρους:

python containment_main.py <transactions file> <queries file> <qnum> <method>

Εκτέλεση δευτέρου μέρους:

python relevance_main.py <transactions file> <queries file> <qnum> <method> <k>

Τα transactions και queries files μπορεί να δίνεται είτε με κατάληξη .txt είτε χωρίς αυτή.