ΑΜ : 1115201400024
Δημήτριος Γάγγας

Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

Project 2
Clustering

Υλοποιήθηκε το Clustering επιτυχώς
σε C++11 με STL library σε λειτουργικό Linux.

- **[Optimality Note]** Σχεδόν όλες οι παράμετροι στις συναρτήσεις γίνονται pass by reference για εξοικονόμηση χώρου και βελτιστοποίηση της ταχύτητας.
- **[Optimality Note2]** Χρησιμοποιήθηκε διαφορετική δομή για την ανάθεση των διανυσμάτων στα clusters στο Reverse Assignment (lsh,hc) απ' ότι υποδείχθηκε στις διαφάνειες.Παρακάτω οι λεπτομέρειες.

- Ελέγχθηκαν επιτυχώς με Valgrind .Συνεπώς δεν υπάρχουν memory leaks.

- Το input dataset μπορει να διαβαστεί είτε αν είναι με tab-separated είτε comma-separated.

- Χρησιμοποιήθηκε το git και το gitkraken για οπτικοποίηση των εκάστοτε αλλαγών αλλά και για την ευκολία επιστροφής σε προηγούμενα στάδια ανάπτυξης του κώδικα.

- Χρησιμοποιήθηκε ο κώδικας για το lsh και το hypercube που είχαν υλοποιηθεί επιτυχώς.

- Περιέχεται makefile που μεταγλωττίζεται με make.

- Έχουν υλοποιηθεί όλες οι απαιτήσεις της εκφώνησης για τα ορίσματα με όποια σειρά και να δοθούν τα flags. Μπορώντας φυσικά να παραληφθούν και κάποια.
  $./cluster –i <input file> –c <configuration file> -o <output file> -d <metric>.
- Επίσης μπορεί να δοθεί και το flag -complete που δίνει την επιπλέον πληροφορία για το ποια itemIds ανήκουν στο εκάστοτε Cluster.

- Περιέχεται στο φάκελο και ένα ενδεικτικό αρχείο output με αποτέλεσματα που έβγαλε το πρόγραμμα.

- Για την καλύτερο δυνατό έλεγχο του κώδικα,διασπάστηκε σε πλήθος αρχείων .cpp/.h ,κλάσεων και συναρτήσεων. Αναλύονται διεξοδικότερα παρακάτω.

Τα αρχεία της προηγούμενης προηγούμενης άσκησης δε θα αναλυθούν διεξοδικά σ'αυτήν

Παρόλ' αυτά θα αναλυθούν κάποιες αλλαγές που έγιναν.
Συγκεκριμένα στη

Class MyVector :

Έγιναν overload οι συγκεκριμένοι operators: + , == .

Για το HyperCube

Δημιουργήθηκε ειδική δομη **struct** CubeTable για την αναπαράσταση του .

Το lsh της 1ης ασκησης έσπασε σε 2 αρχεία/classes

Το lshSimple και το LSH .

Το LSH περιέχει πλέον δείκτη στη δομή HashTables(MHT) .

Αντίστοιχα , το HyperCube περιέχει επίσης δείκτη σε δομή CubeTable.

Αυτό έγινε με στόχο τη καλύτερη δυνατή απόκρυψη της πληροφορίας.

Έπειτα δημιουργήθηκε μια abstract class [**class** AbstractLshCube] η οποία περιέχει τη virtual συνάρτηση **getSuperSet()** , που αποτελει την επικοινωνια της δομης lsh ή hypercube με το υπολοιπο προγραμμα.

Η συνάρτηση αυτή συγκεκριμένα μας επιστρέφει για συγκεκριμένο query (στη περίπτωση μας Centroid/Medoid) ένα set από όλα τα string vector_ids που βρισκόντουσαν στα buckets και πληρούν τους περιορισμούς .

Για παράδειγμα, για δομη lsh μας επιστρέφει ολα τα vectorIds απο τα L hashTables στα οποία υπάρχει collision με το query/Centroid μας.

Αντίστοιχα ,για δομή hypercube μας επιστρέφει όλα τα vectorIds απο την/τις N probes κορυφές πάντα σε συμμόρφωση με το περιορισμό max M επιτρεπτών στοιχείων.

Όσον αφορά τα καινούργια αρχεία

1. **Initialization.cpp/h**
2. **Assignment.cpp/h**
3. **Update.cpp/.h**

Στα παραπάνω αρχεία περιέχονται υλοποιημένες οι συναρτήσεις που αναφέρονται στην εκφώνηση .
Μάλιστα στόχος ήταν η αποφυγή του spaghetti code και η όσο το δυνατόν καλύτερη διαμέριση των ενεργειών.
Αυτό είχε ως αποτέλεσμα να χρησιμοποιηθούν functors πολυμορφικά.
Ουσιαστικά είναι κλάσεις στις οποίες έχει γίνει overload ο operator ()
Με αποτέλεσμα να μπορούν να χρησιμοποιηθούν σα συναρτήσεις κρατώντας όμως και state λόγω των attributes.
Αυτο εξυπηρετεί στο να υπάρχουν ακριβώς ίδια ορίσματα με αποτέλεσμα να είναι δυνατή η εικονικοποίηση τους.
Για παράδειγμα ο PAM updater χρειαζοταν να γνωρίζει τη μετρικη απόστασης ενω ο απλός kmeans όχι.

Όσον αφορά το ReverseAssignment

Όπως προανέφερα υπάρχει μια συνάρτηση **getSuperSet()** η οποία επιστρέφει σετ απο βεκτορς ανεξαρτήτου δομής lsh ή υπερκύβου.
Αυτή καλείται στη συνάρτηση RevAssign().
Συνεπώς , για κάθε Centroid του Cluster επιστρέφεται αυτό το SuperSet.
Πάνω εκεί έχει δημιουργηθεί ένα UnorderedMap τρείδας TripletTable.
Ουσιαστικά αποτελείται απο τη 3αδα < (vectorItemId , clusterIndex , distance) >

Το uMap έχει ώς κλειδι τα διανύσματα του dataset ή όσα υπάρχουν αθροιστικά στα dataset των cluster (από μετρήσεις ~4800) και ως values
το pair:<(clusterIndex , distance)> .

Κατέληξα σε αυτη τη δομή γιατί το unordered map είναι υλοποιημένο με hash functions και hash table συνεπώς μπορεί να δώσει σε πολυπλοκότητα O(1) την αναζήτηση του διανύσματος που ψάχνω (βέβαια στη χειρότερη O(n) αλλά σχετικά απίθανο).
Και στη συγκεκριμένη περίπτωση το κατατάσσει ιδανική δομη για τη πολλαπλη αναζήτηση που θα χρειαστεί στον αλγόριθμο.

Η βασική ιδέα του αλγορίθμου είναι

Για κάθε βεκτορ στο dataset του εκάστοτε Cluster

Αν υπάρχει τοτε:

Αναζήτησε το στο TripletTable [Average O(1)].

Αν δεν έχει υπολογιστεί η απόσταση τοτε:

υπολόγισε και τις 2 αποστάσεις και βάλε αυτό με τη μικρότερη απόσταση.

Αλλιώς αν υπάρχει τοτε:

Υπολόγισε την απόσταση του τρέχοντος και Συγκρινε την αποσταση αυτου στου TripletTable και βάλε αυτό με τη μικρότερη απόσταση.

Αλλιώς Αν δεν υπάρχει ήδη :

βαλτο στο TripletTable **χωρις όμως να υπολογίσεις απόσταση!**

- Επίσης, στα αρχεία EvaluationMetric.cpp /.h έχει υλοποιηθεί και ο δείκτης εσωτερικής αξιολόγησης της συσταδοποίησης Silhouette.

- Τέλος το το lsh και το hypercube reverse assign συνήθως σε πολλές περιπτώσεις δεν επιστρέφει ολους τα vector Ids με αποτέλεσμα actual dataset > datasetInClusters.
  Συνεπώς για να μην έχουμε σε κάθε επανάληψη το φόρτο της εύρεσης και ανάθεσης των υπολειπόμενων σημείων στο τέλος, αφού γίνει το update και διαγραφούν ολα τα dataset για καθε cluster , κάνουμε την τελευταία ανάθεση με Lloyd για να κάνει “σκουπα” και τα σημεία που δεν υπήρχαν στα clusters.

Μετρήσεις-Πειράματα-Σχόλια

Στο dataset το Γενικά έγιναν διάφοροι πειραματισμοι.
Κυρίως στις περιπτώσεις I1A1U1 - I1A1U2 - I1A2U2 - I1A3U2 και μεταξυ cosine
Και euclidean distance metric

Γενικες παρατηρήσεις που μπορούν να εξαχθούν απ' αυτά τα πειράματα είναι τα εξής:

- Το ευκλείδιο μέτρο σε τρεξίματα με 50 ή 200 cluster φαίνεται να αργεί αρκετά περισσότερο από ότι το cosine.
  Για παράδειγμα στο κλασσικό 1-1-1 χρειαζόταν της τάξης των 20 seconds
  Παραπάνω απο οτι το αντίστοιχο του cosine.
- Παραταύτα το euclidean σχεδόν σε όλα τα τρεξίματα του Silhouette φαίνεται να έχει αρκετά καλύτερες αποτιμήσεις απο το cosine για τις αντίστοιχες περιπτώσεις.
- Κατι που πιθανως περιμέναμε είναι ότι το PAM είναι αρκετά πιο αργό από το kmeans update αλλά κατι που μάλλον δεν περιμέναμε είναι το PAM είχε αρκετά καλύτερες αποτιμήσεις σε σχέση με το απλο kmeans update.
- Στο lsh είναι ακόμη περισσότερο τυχαιοκρατικά τα ενδεχόμενα με αποτέλεσμα να μην μπορούμε να βγάλουμε κάποια καθολικά συμπεράσματα.
  Παρολ 'αυτα συγκρίνοντας το χρόνο και την αποτίμηση του euclidean lsh σε σχέση με τους υπόλοιπους αλγόριθμους παρατηρούμε οτι τα cluster στο lsh "διαχέονται" αρκετά καλα στο dataset σε μάλιστα εξαιρετικά μικρότερο χρόνο.
  Συγκεκριμένα , στο 1-1-2 το ευκλείδο σε k = 100 χρειαζόταν 200 sec ενω στο αντίστοιχο lsh μόλις 22 sec.
- Εδώ οφείλουμε να επισημάνουμε και τη δομή hyperCube η οποία δινει εξίσου καλά αποτελέσματα σε ακόμη λιγότερο χρόνο για (probes =2/3 , M =inf , k_hash_function = 7).
  Επίσης μια άλλη παρατήρηση που έγινε πάνω σ'αυτό είναι το γεγονός οτι χρειάζεται λιγότερες επαναλήψεις σε σχέση με τις υπόλοιπες δομές.
  Με συνέπεια τα κεντρα να σταθεροποιούνται αρκετα γρηγορα (+PAM).

Αποτίμηση Silhouette για euclidean:

	1-1-2	1-2-2	1-3-2
K = 50	0.257	0.275	0.370
K = 100	0.357	0.334	0.41
K = 200	0.412	0.388	0.39

Αποτίμηση Silhouette για cosine:

	1-1-2	1-2-2	1-3-2
K = 50	0.116	0.164	0.152
K = 100	0.176	0.180	0.175
K = 200	0.233	0.242	0.227

Για τις εξής παραμέτρους:
number_of_hash_functions: 7
number_of_hash_tables: 5
number_of_probes: 3
M_cube = inf
W =1