Software Development for Algorithmic Projects - Project 1

Στοιχεία Φοιτητών

Λάζαρος Αυγερίδης - 1115201600013
Θέμης Βαρβέρης - 1115201600015

Github: https://github.com/themisvr/Vector-Clustering-Algorithms

Σύντομη Περιγραφή Προγράμματος

Στο μέρος (**A**) της εργασίας υλοποιήσαμε αναζήτηση σε χώρους μεγάλων διαστάσεων $\mathbb{R}^d$ μέσω των αλγορίθμων LSH και Randomized Projection (σε υπερκύβο διάστασης d' όπου d' < d) . Και στις 2 περιπτώσεις, αρχικά διαβάζονται από το input file τα δεδομένα "εκπαίδευσης" (training set), που στη περίπτωσή μας είναι διανύσματα 784 διαστάσεων. Στη συνέχεια, υπολογίζεται η μέση απόσταση του πλησιέστερου γείτονα δειγματοληπτικά (χρησιμοποιείται στον υπολογισμό του *w*) και έπειτα δημιουργούνται οι δομές αναζήτησης, στην μία περίπτωση τα *L* hash tables (LSH) και στην άλλη περίπτωση το 1 και μοναδικό hash table (Randomized Projection – Hypercube). Αφού λοιπόν, τα σημεία / διανύσματα του training set αποθηκευθούν στις δομές αναζήτησης, γίνεται η ανάγνωση του query file, στο οποίο επίσης περιέχονται δεδομένα διάστασης 784, αλλά είναι διαφορετικά από αυτά που οι αλγόριθμοι είχαν συναντήσει μέχρι τώρα. Μετά, περνάμε στο στάδιο της αναζήτησης, όπου είτε για ολόκληρο το σύνολο των queries, είτε για κάποιο υποσύνολο αυτών, τα 2 εκτελέσιμα βρίσκουν για κάθε τέτοιο query:

1. τους N πλησιέστερους γείτονές του, καθώς και τις αποστάσεις τους από αυτό (προσεγγιστικός υπολογισμός)
2. τις αποστάσεις των N αληθινών πλησιέστερων γειτόνων από αυτό (ακριβής υπολογισμός)
3. όλους του πλησιέστερους γείτονες του εντός ακτίνας *R* – range search (προσεγγιστικός υπολογισμός)

Τέλος, τα αποτελέσματα γίνονται write στο αρχείο εξόδου, το οποίο έχει προσδιοριστεί από το χρήστη είτε όταν εκτέλεσε το πρόγραμμα από τη γραμμή εντολής, ή δυναμικά κατά τη διάρκεια της εκτέλεσης του προγράμματος. (Πριν τερματίσει το πρόγραμμα, ο χρήστης μπορεί να επιλέξει να επαναλάβει αναζήτηση με κάποιο άλλο υποσύνολο των queries, ή να προσδιορίσει διαφορετικό query file.)
Τα εκτελέσιμα που παράγονται στο μέρος (A) είναι τα:

- ➔ **lsh**
- ➔ **cube**

Software Development for Algorithmic Projects - Project 1

Στο μέρος (**B**) της εργασίας υλοποιήσαμε συσταδοποίηση διανυσμάτων στο χώρο $\mathbb{R}^d$. Ο αλγόριθμος που χρησιμοποιήθηκε εδώ είναι μία βελτιωμένη σε κάθε βήμα (αρχικοποίηση – ανάθεση - ανανέωση) εκδοχή του k-Means, που ονομάζεται k-Medians++. Αρχικά, γίνεται η ανάγνωση των δεδομένων από το input file, και εφόσον στον αλγόριθμο αυτόν δεν κατασκευάζεται κάποια δομή αναζήτησης, ξεκινάμε τη διαδικασία του clustering. Τα βήματα είναι τα εξής:

1. Initialization++: αντί να επιλέγονται τα αρχικά κέντρα των clusters τυχαία, επιλέγονται με βάση την απόστασή τους από τα υπόλοιπα κέντρα. Όσο μεγαλύτερη είναι η απόσταση ενός σημείου από το κοντινότερό του κέντρο, τόσο μεγαλύτερη η πιθανότητα να επιλεχθεί το σημείο αυτό ως ένα από τα αρχικά k κέντρα. Με άλλα λόγια, στην αρχικοποίηση αυτή, εξασφαλίζουμε ότι τα αρχικά κέντρα δεν θα είναι κοντά το ένα με το άλλο.
2. Assignment: Στο βήμα αυτό χρησιμοποιήθηκαν 3 διαφορετικές προσεγγίσεις (Lloyd's, LSH Reverse Assignment, Hypercube Reverse Assignment) οι οποίες θα αναλυθούν στη συνέχεια.
3. Median Update: Σε κάθε επανάληψη τα clusters αποκτούν νέο κέντρο και αυτό επιτυγχάνεται με τον υπολογισμό του διάμεσου διανύσματος.

Όταν οι αλλαγές στις αναθέσεις των clusters είναι πλέον "μικρές", τότε ο αλγόριθμος σταματάει και περνάμε στον υπολογισμό της σιλουέτας. Τέλος, τα αποτελέσματα γίνονται write στο αρχείο εξόδου, το οποίο έχει προσδιοριστεί από το χρήστη όταν εκτέλεσε το πρόγραμμα από τη γραμμή εντολής.

Το εκτελέσιμο που παράγεται στο μέρος (B) είναι το:

➔ **cluster**

Κατάλογος Αρχείων Κώδικα / Επικεφαλίδων

Η εργασία, χωρίζεται σε 3 μεγάλους καταλόγους.

1. **src:** Ο κατάλογος αυτός περιέχει όλα τα αρχεία κώδικα (.cpp αρχεία).
2. **include:** Ο κατάλογος αυτός περιέχει όλα τα αρχεία επικεφαλίδων (.h αρχεία).
3. **datasets:** Ο κατάλογος αυτός περιέχει ένα αρχείο με δεδομένα που πρέπει να γίνουν **train** και ένα αρχείο με τα **queries** πάνω στα οποία θα ελεγχθούν οι αλγόριθμοι μας.

Υποκατάλογοι src:

- **lsh:** Περιέχει το lsh_app.cpp αρχείο το οποίο είναι και το αρχείο για την εκτέλεση της εφαρμογής της προσομοίωσης για τον αλγόριθμο LSH. Επίσης περιέχει το Makefile και την μεταγλώττιση αυτού.

Software Development for Algorithmic Projects - Project 1

- **cube:** Περιέχει το cube_app.cpp αρχείο το οποίο είναι και το αρχείο για την εκτέλεση της εφαρμογής της προσομοίωσης για τον υπερκύβο. Επίσης περιέχει και το Makefile για την μεταγλώττιση αυτού.
- **cluster:** Περιέχει τα αρχεία cluster_app.cpp και cluster_utils.cpp. Το πρώτο είναι το αρχείο κώδικα για την εκτέλεση της προσομοίωσης για το clustering και το δεύτερο αρχείο αφορά την υλοποίηση συναρτήσεων, χρήσιμων για τους αλγορίθμους που ζητούνται σχετικά με το clustering. Επίσης περιέχει και το Makefile για την μεταγλώττιση των αρχείων αυτών.
- **common:** Περιέχει το αρχείο io_utils.cpp το οποίο είναι το αρχείο κώδικα που υλοποιεί συναρτήσεις σχετικά με το I/O όλων των εφαρμογών.

Υποκατάλογοι include:

- **cluster:** Περιέχει τα εξής 3 αρχεία. Πρώτο αρχείο, το αρχείο cluster.conf το οποίο είναι το αρχείο με τα configurations του clustering. Δεύτερο αρχείο είναι το cluster_utils.h το οποίο περιέχει πρότυπα συναρτήσεων σχετικά με την υλοποίηση των αλγορίθμων για το clustering. Τελευταία αρχείο είναι το αρχείο cluster.h το οποίο υλοποιεί ολόκληρη την κλάση Cluster (template) και τις μεθόδους της (initialization, update, assignment etc.) όπως υλοποιούνται το καθένα ξεχωριστά.
- **hash_function:** Περιέχει το αρχείο hash_function.h το οποίο υλοποιεί 2 templated κλάσεις. Την κλάση HashFunction η οποία χρησιμοποιείται για την δημιουργία των h hash functions και την κλάση AmplifiedHashFunction η οποία χρησιμοποιείται για την δημιουργία των g hash functions.
- **io_utils:** Περιέχει 2 αρχεία. Το πρώτο αρχείο είναι το cmd_args.h το οποίο περιέχει κλάσεις σχετικα με τα arguments του κάθε εκτελέσιμου (π.χ. cube_args, lsh_args). Το δεύτερο αρχείο είναι το io_utils.h το οποίο περιέχει πρότυπα συναρτήσεων σχετικά με λειτουργίες I/O και την υλοποίηση μιας templated συνάρτησης read_file η οποία είναι υπεύθυνη για το διάβασμα των αρχείων που μας δίνονται με την συγκεκριμένη δομή.
- **math_utils:** Περιέχει το αρχείο math_utils.h το οποίο είναι υπεύθυνο για πρότυπα συναρτήσεων σχετικών με μαθηματικές πράξεις όπως το exponential modulo.
- **metric:** Περιέχει το αρχείο metric.h το οποίο υλοποιεί την μετρική συνάρτηση Manhattan στον d-διάστατο χώρο.
- **modules:** Ο κατάλογος αυτός περιέχει επιπλέον 3 καταλόγους οι οποίοι είναι οι παρακάτω:
    1. **exact_nn:** Ο κατάλογος αυτός περιέχει το αρχείο exact_nn.h, το ο οποίο υλοποιεί την αντίστοιχη συνάρτηση (templated), δηλαδή την εξαντλητική αναζήτηση του πλησιέστερου γείτονα.
    2. **hypercube:** Περιέχει το αρχείο hypercube.h, το οποίο υλοποιεί την κλάση Hypercube και τις απαραίτητες μεθόδους της για την υλοποίηση των αλγορίθμων στον υπερκύβο.

3. **lsh:** Περιέχει το αρχείο lsh.h, το οποίο υλοποιεί την κλάση LSH και τις απαραίτητες μεθόδους της για την υλοποίηση των αλγορίθμων αναζήτησης.

Όλες οι παραπάνω υλοποιήσεις κλάσεων / συναρτήσεων που βρίσκονται σε αρχεία επικεφαλίδων (.h) είναι υλοποιημένες με templates (generic) (ορισμένες συναρτήσεις είναι inline) για να πετύχουμε όσο περισσότερη γενίκευση στην διαχείριση των δεδομένων μας.

Οδηγίες Μεταγλώττισης

Στους υποκαταλόγους **cluster, lsh, cube** του καταλόγου **src** βρίσκονται τα αρχεία που προσομοιώνουν τους αλγορίθμους που ζητούνται. Σε αυτούς τους φακέλους, μαζί με τα αντίστοιχα αρχεία αυτά (main συναρτήσεις), υπάρχουν και τα αντίστοιχα Makefiles που μεταγλωττίζουν τα αρχεία με τα απαραίτητα includes, flags etc. Οπότε η εντολή μεταγλώττισης είναι η εξής:
$ make
Δημιουργείται το εκτελέσιμο με τα αντίστοιχα ονόματα καταλόγων (δηλ, lsh, cube, cluster) και ταυτόχρονα ένας φάκελος με όνομα **obj** ο οποίος κατά την μεταγλώττιση τοποθετεί τα αντικείμενα αρχεία. Τέλος για να καθαριστεί ο φάκελος obj αλλά και να διαγραφεί το εκτελέσιμο εκτελούμε την παρακάτω εντολή:
$ make clean

Σημείωση: Στην μεταγλώττιση χρησιμοποιούμε -Ofast για να μπορέσετε να δείτε αποτελέσματα για διάφορες παραμέτρους, οι οποίες μπορεί να αποτελούν παράγοντα αύξησης χρόνου.

Οδηγίες Χρήσης

LSH

Το εκτελέσιμο καλείται από τη γραμμή εντολής ως εξής:
$ ./lsh -d [path_to_input_file] -q [path_to_query_file] -k [number_of_h_hash_functions] -L [number_of_hash_tables] -o [path_to_output_file] -N [number_of_nearest_neighbors] -R [range_search_radius]

Ωστόσο, ο χρήστης μπορεί να τρέξει το πρόγραμμα από τη γραμμή εντολής χωρίς να προσδιορίσει καμία επιπλέον παράμετρο, δηλαδή απλά γράφοντας:
$ ./lsh
Σε αυτήν την περίπτωση, ο χρήστης δυναμικά θα πρέπει να δώσει μέσω του standard input τα paths για τα input, query και output file, και οι παράμετροι της

δομής του lsh (k, L) καθώς και οι παράμετροι της αναζήτησης (N, R) παίρνουν τις default τιμές.

Συνεπώς, όταν το πρόγραμμα αρχίσει να εκτελείται, συμβαίνουν τα εξής:

1. Ανάγνωση του input file και αποθήκευση των δεδομένων (στην περίπτωσή μας τα δεδομένα είναι δυαδικής μορφής: 60.000 εικόνες διάστασης 784) .
2. Υπολογισμός της μέσης απόστασης του πλησιέστερου γείτονα υπολογισμένη με ακρίβεια, αλλά δειγματοληπτικά χρησιμοποιώντας το 1 / x των δεδομένων του input file. Η απόσταση αυτή χρησιμοποιείται στη συνέχεια στον υπολογισμό της παραμέτρου *w* του αλγορίθμου.
3. Δημιουργία της δομής του lsh, δηλαδή όλα τα σημεία / διανύσματα του input file αποθηκεύονται σε L hash tables. Το κάθε hash table έχει αριθμό buckets ίσο με τον αριθμό dataset.size() / N, όπου στην περίπτωση μας το N είναι ο αριθμός 16 (συνήθως 8, 16).
4. Ανάγνωση του query file και αποθήκευση των δεδομένων, τα οποία είναι ίδιας διάστασης με τα δεδομένα που υπάρχουν στο input file, αλλά προφανώς είναι διαφορετικά, για να μπορούμε να ελέγξουμε την δυνατότητα γενίκευσης του αλγορίθμου (στην περίπτωσή μας το query file περιέχει 10.000 queries) .
5. Για κάθε query του query file, εκτελείται αναζήτηση N πλησιέστερων γειτόνων προσεγγιστικά και με ακρίβεια, καθώς και αναζήτηση πλησιέστερων γειτόνων εντός ακτίνας R.
6. Στη συνέχεια, το πρόγραμμα γράφει τα αποτελέσματα στο output file και ρωτά το χρήστη αν επιθυμεί να επαναλάβει την διαδικασία με διαφορετικό query file.

Hypercube

Το εκτελέσιμο καλείται από τη γραμμή εντολής ως εξής:
$ ./cube -d [path_to_input_file] -q [path_to_query_file] -k [hypercube_dimension] -M [max_number_of_points_to_check] -probes [max_number_of_vertices_to_check] -o [path_to_output_file] -N [number_of_nearest_neighbors] -R [range_search_radius]

Ωστόσο, ο χρήστης μπορεί να τρέξει το πρόγραμμα από τη γραμμή εντολής χωρίς να προσδιορίσει καμία επιπλέον παράμετρο, δηλαδή απλά γράφοντας:
$ ./cube
Σε αυτήν την περίπτωση, ο χρήστης δυναμικά θα πρέπει να δώσει μέσω του standard input τα paths για τα input, query και output file, και οι παράμετροι της δομής του hypercube (k, M, probes) καθώς και οι παράμετροι της αναζήτησης (N, R) παίρνουν τις default τιμές.

Συνεπώς, όταν το πρόγραμμα αρχίσει να εκτελείται, συμβαίνουν τα εξής:

7. Ανάγνωση του input file και αποθήκευση των δεδομένων (στην περίπτωσή μας τα δεδομένα είναι δυαδικής μορφής: 60.000 εικόνες διάστασης 784) .

8. Υπολογισμός της μέσης απόστασης του πλησιέστερου γείτονα υπολογισμένη με ακρίβεια, αλλά δειγματοληπτικά χρησιμοποιώντας το 1 / x των δεδομένων του input file. Η απόσταση αυτή χρησιμοποιείται στη συνέχεια στον υπολογισμό της παραμέτρου *w* του αλγορίθμου.
9. Δημιουργία της δομής του hypercube, δηλαδή όλα τα σημεία / διανύσματα του input file αποθηκεύονται στο ένα και μοναδικό hash table της δομής. Το hash table αυτό έχει αριθμό buckets ίσο με τον αριθμό των δεδομένων στο input file, δηλαδή $2^{\log 60.000} = 60.000$.
10. Ανάγνωση του query file και αποθήκευση των δεδομένων, τα οποία είναι ίδιας διάστασης με τα δεδομένα που υπάρχουν στο input file, αλλά προφανώς είναι διαφορετικά, για να μπορούμε να ελέγξουμε την δυνατότητα γενίκευσης του αλγορίθμου (στην περίπτωσή μας το query file περιέχει 10.000 queries) .
11. Για κάποιο υποσύνολο του query file, καθώς για όλα τα queries το πρόγραμμα χρειάζεται πολύ ώρα για να τερματίσει, και για κάθε query αυτού του υποσυνόλου εκτελείται αναζήτηση N πλησιέστερων γειτόνων προσεγγιστικά και με ακρίβεια, καθώς και αναζήτηση πλησιέστερων γειτόνων εντός ακτίνας R.
12. Στη συνέχεια, το πρόγραμμα γράφει τα αποτελέσματα στο output file και ρωτά το χρήστη αν επιθυμεί να επαναλάβει την διαδικασία με διαφορετικό υποσύνολο του query file, ή με κάποιο διαφορετικό query file.

Clustering

Το εκτελέσιμο καλείται από τη γραμμή εντολής ως εξής:
$ ./cluster -i [path_to_input_file] -c [path_to_config_file] -o [path_to_output_file] --complete [optional] -m [Classic_or_LSH_or_Hypercube]

Όμοια με πριν, το πρόγραμμα όταν κληθεί από το χρήστη, αρχικά κάνει το parsing των command line arguments και διαβάζει το configuration file. Στη συνέχεια, διαβάζει το input file, αποθηκεύει τα δεδομένα και ξεκινάει την διαδικασία του clustering, χωρίς την κατασκευή κάποιας επιπλέον δομής.
Αρχικά, εκτελείται η αρχικοποίηση των κέντρων με την τεχνική initialization++ που αποσκοπεί ουσιαστικά στο να “διασκορπίσει” τα κέντρα με τυχαίο τρόπο. Στη συνέχεια επαναλαμβάνονται τα παρακάτω βήματα μέχρι να οδηγηθούμε σε σύγκλιση του αλγορίθμου:

1. Ανάθεση των σημείων στα μέχρι τώρα κέντρα. Χρησιμοποιείται είτε η κλασική μέθοδος του Lloyd’s ή μέθοδος Reverse Assignment με LSH και Hypercube. Στη μέθοδο Lloyd’s κάθε σημείο του συνόλου δεδομένων γίνεται assign στο κοντινότερό του κέντρο χρησιμοποιώντας ακριβείς υπολογισμούς. Στο Reverse Assignment, για κάθε κέντρο κάνουμε range search με ακτίνα που συνεχώς αυξάνεται και κάθε σημείο του dataset, το οποίο βρίσκεται εντός της ακτίνας αυτής, το κάνουμε assign στο cluster αυτού του κέντρου.

Software Development for Algorithmic Projects - Project 1

2. Ανανέωση των κέντρων υπολογίζοντας το διάμεσο διάνυσμα του κάθε cluster και ορίζοντας αυτό ως νέο κέντρο του κάθε cluster.

Στο πρόγραμμά μας, αυτή η επαναληπτική διαδικασία σταματά όταν η απόλυτη τιμή της διαφοράς της αντικειμενικής συνάρτησης του k-Medians των επαναλήψεων n - 1 και n είναι μικρότερη κάποιου αριθμού ε. Γενικά, για να επιτευχθεί σύγκλιση δεν απαιτούνται πολλές επαναλήψεις του αλγορίθμου. Όταν συμβεί αυτό, προχωράμε στον υπολογισμό της σιλουέτας, ο οποίος είναι αρκετά χρονοβόρος, και τέλος γίνεται write των αποτελεσμάτων στο αρχείο εξόδου.

Στον παρακάτω πίνακα σε κάθε κελί είναι η αντίστοιχη τιμή του s_{total}:

Assignment method	*k = 2*	*k = 3*	*k = 4*	*k = 5*	*k = 6*	*k = 7*	*k = 8*	*k = 9*	*k = 10*
Lloyd's	0.101	0.070	0.100	0.111	0.128	0.110	0.123	0.121	0.159
LSH	0.101	0.102	0.090	0.117	0.120	0.140	0.097	0.137	0.150
Hypercube	0.099	0.074	0.099	0.114	0.131	0.129	0.130	0.138	0.146