Nearest Items and Clustering

Πίνακας Περιεχομένων

Στοιχεία

ΝΤΑΡΟΥΙΣ ΝΙΖΑΡ 1115201800286
ΤΣΙΛΙΦΩΝΗΣ ΑΡΗΣ 1115201700170

Οδηγίες μεταγλώττισης και εκτέλεσης του προγράμματος

Για την μεταγλώττιση έχει φτιαχτεί ένα makefile το οποίο φτιάχνει τα απαραίτητα εκτελέσιμα για όλα τα προγράμματα.
Η εκτέλεση των προγραμμάτων γίνεται ακριβώς όπως ζητήθηκε στην εκφώνηση.
Οι παράμετροι γράφονται με οποιαδήποτε σειρά.

Περιγραφή

Η εργασία υλοποιεί τους αλγορίθμους αναζήτησης κοντινότερου γείτονα, LSH και Hypercube χρησιμοποιώντας την μετρική L2 (euclidean distance). Καθώς και τους αλγορίθμους συσταδοποίησης διανυσμάτων στον χώρο, Lloyds και reverse Range search χρησιμοποιώντας τον LSH και Hypercube, με την ίδια μετρική. Τέλως, η αρχικοποίηση των διανυσμάτων που θα χρησιμοποιηθούν από τους αλγόριθμους συσταδοποίησης πραγματοποιείται με την τεχνική k-Means++.
Όλα τα ζητούμε της άσκησης έχουν υλοποιηθεί.
Το πρόγραμμα δεν αποθηκεύει διπλότυπα δεδομένα, στα hash tables αποθηκεύονται τα Item_id.

Κατάλογος των αρχείων

Unit tests Αρχείο και Περιγραφή

Τα LSH αρχεία θα τα βρείτε στον κατάλογο “Algorithmic-Problems-Project1\algo\test_cases”. Στο αρχείο “Unitest.cpp” θα βρείτε 3 test cases:

- Test για το Continuous Frechet Distance
- Test για το Discrete Frechet Distance
- Test για το Curve Filtering
- Test για το Mean Curve

Για μεταγλώττιση έχει φτιαχτεί ένα makefile στο αρχείο “test_cases”.

Mean Curves Αρχεία και Περιγραφή

Τα Mean Curves αρχεία τα βρείτε στον κατάλογο
“Algorithmic-Problems-Project1\algo\curves”.
Χρησιμοποιούμε τον αλγόριθμο των διαφανειών, στην συνάρτηση mean curve tree, για την ανανέωση των centroids. Αν και η συνάρτηση καλείται με τον ίδιο τρόπο που καλούνταν και η προηγούμενη συνάρτηση για την ανανέωση των centroids(συγκεκριμένα calculate_mean), είναι εντελώς διαφορετική. Συγκεκριμένα με τα στοιχεία του cluster δημιουργούνται τα leaf nodes του δέντρου με τυχαία σειρά. Έπειτα ανά δυάδες υπολογίζουμε το father node κάνοντας build up την ιεραρχία του δέντρου μέχρι την ρίζα. Αν κάποιος κόμβος είναι περιττός τον προσθέτουμε στην ιεραρχία. Όσον αφορά τον υπολογισμό της εκάστοτε τιμής, πρώτα βρίσκουμε το optimal path του ζευγαριού και στην συνέχεια για κάθε συντεταγμένη , υπολογίζουμε τον μέσο όρο τους και τον βάζουμε στον vector της mean curve.Τόσο το optimal path όσο και η discrete υπολογίζεται όπως στις διαφάνειες με κάποια παραπάνω corner cases για την optimal.Επειδή παρατηρήσαμε πως δεν έχει κάποια σημασία να αποθηκεύουμε κόμβους πέρα από το προηγούμενο επίπεδο, δεν δημιουργήσαμε δομή δέντρου .Απλά κρατάμε ένα διάνυσμα με τιμές για κάθε επίπεδο και όταν τελειώσει ο υπολογισμός του επόμενου επιπέδου , διαγράφουμε το παλιό διάνυσμα και βάζουμε τα νέα στοιχεία. Τέλος με την παράμετρο LSH_Frechet στο command line, όπου έχουμε euclidean distance υπολογίζουμε το discrete.

Για την γραμμική αύξηση των διαστάσεων του mean curve, το αντιμετωπίζουμε βγάζοντας στοιχεία από την αρχή και το τέλος της καμπύλης μέχρι το πλήθος των στοιχείων της καμπύλης να είναι ίδιος με το πλήθος στοιχείων των καμπυλών από το input data (ίδιες διαστάσεις δηλαδή).

LSH Αρχεία και Περιγραφή

Τα LSH αρχεία θα τα βρείτε στον κατάλογο
“Algorithmic-Problems-Project1\algo\LSH_Folder” και το πρόγραμμα (main) που εκτελεί τους μεθόδους και τον αλγόριθμο βρίσκεται στον κατάλογο
“Algorithmic-Problems-Project1\algo”.
Στο αρχείο “LSH/LSH.hpp” θα βρείτε τις δομές, τις μεταβλητές και τις μεθόδους που χρησιμοποιούνται στον αλγόριθμο, καθώς και σχόλια εξηγόντας τι κάνει η κάθε μέθοδο, δομή και σε τι χρησιμεύει η κάθε μεταβλητή.
Η συνάρτηση κατακερματισμού, και η τιμή της, υπολογίζεται στην συνάρτηση “Specific_Hash_Value” και έχει υλοποιηθεί ακριβώς έτσι όπως έλεγε η θεωρεία στις διαφάνειες. Το w ορίζεται με τον υπολογισμό της μέσης ευκλείδειας απόστασης μεταξύ των 10% των δεδομένων εισόδου και για το continuous είναι το 2%, εάν η είσοδος είναι μικρή και επομένως το 5% είναι μικρότερο από το μηδέν, παίρνουμε το 50% των δεδομένων. Η τιμή που παίρνει το w πρέπει να εξαρτάται από τα δεδομένα.

Καταλήξαμε στο 5% των δεδομένων πειραματίζοντας με διάφορα ποσοστά και παρατηρώντας τα αποτελέσματα.
Όσον αφορά το πλήθος των buckets παρατηρήσαμε πως το n/4 είναι αρκετά καλή τιμή καθώς και είναι πιο σκορπισμένα τα δεδομένα με αποτέλεσμα ο αλγόριθμος να είναι πιο γρήγορος.
Οι αλγόριθμοι search by range και approximate nearest neighbors έχουν υλοποιηθεί στο αρχείο "LSH_Folder/LSH.cpp" με πλήρη περιγραφή για την λειτουργία τους, πάνω από τις συναρτήσεις "Search_by_range" και "Nearest_N_search".

Η συνάρτηση που κάνει Filter της καμπύλες βρίσκεται στο αρχείο "Algorithmic-Problems-Project1\algo\functions"
Η Grid συνάρτηση βρίσκεται στο αρχείο "LSH_Folder/LSH.cpp". Όσο πιο μεγάλο είναι το δ τόσο πιο γρήγορος είναι ο αλγόριθμος, με χειρότερα αποτελέσματα.

Για το Filtering αποφασίσαμε πως η τιμή ε θα είναι το μέσο βήμα από σημείο σε σημείο.

Hypercube Αρχεία και Περιγραφή

Τα Hypercube αρχεία θα τα βρείτε στον κατάλογο "Algorithmic-Problems-Project1\algo\Hypercube" και το πρόγραμμα (main) που εκτελεί τους μεθόδους και τον αλγόριθμο βρίσκεται στον κατάλογο "Algorithmic-Problems-Project1\algo".
Στο αρχείο "Hypercube/Hypercube.hpp" θα βρείτε τις δομές, τις μεταβλητές και τις μεθόδους που χρησιμοποιούνται στον αλγόριθμο, καθώς και σχόλια εξηγόντας τι κάνει η κάθε μέθοδο, δομή και σε τι χρησιμεύει η κάθε μεταβλητή.
Αρχικά έχω δημιουργήσει ένα πίνακα όπου για κάθε θέση έχω ζευγάρια (hi,fi) . Έτσι μπορώ να αποθηκεύω για κάθε τιμή του H[i] την αντίστοιχη τιμή fi (το H[1] έχει πολλά {h1,f1}). Δηλαδή για h1(point_1) = 3 και f1(h1) = 0 σώζω το ζευγάρι {h1,f1} ώστε άμα ξανατύχει κάποιο point_x να έχει h1(point_x) = 3 να μην χρειαστεί να ξαναυπολογίσω το f1(h1).Έτσι δημιουργείται μια δομή hashtable όπου για κάθε H[i] βρίσκω ή υπολογίζω την αντίστοιχη τιμή fi. Με αυτόν τον τρόπο δημιουργείται για κάθε σημείο p η εξής συμβολοσειρά : [f1(h1(p)),f2(h2(p)) . . . , fd'(hd'(p))] , d' = k στην υλοποίηση. Στην συνέχεια, αυτή η συμβολοσειρά που προκύπτει (πχ. κ = 4 -> 0110) είναι η δυαδική αναπαράσταση ένος δεκαδικού αριθμού , που αντιστοιχεί στην θέση του bucket στο hypercube hashtable.Οπότε 0110 = 7 , συνεπώς το σημείο μπαίνει στον bucket 7. Όσον αφορά την αναζήτηση των queries, πρώτα βρίσκω το bucket που βάζει ο αλγόριθμος hypercube το query. Έπειτα βρίσκω με hamming distance τους γειτονικούς bucket σε αυτόν τον bucket. Δηλαδή για κάθε bucket βρίσκω το hamming distance του bucket με το bucket που bucket που έπεσε το query. Όπως γνωρίζουμε το hamming distance βρίσκει πόσους διαφορετικούς άσσους και μηδενικά έχουν δύο ακέραιοι. (σαν να σου λέει πόσα διαφορετικά fi έχουν).Αφού ταξινομήσω τα γειτονικά

buckets ανάλογα με την απόσταση του query bucket, ψάχνω τόσους buckets όσους μου λέει η παράμετρος probes.Για κάθε bucket , ψάχνω όλα τα σημεία του και αναζητώ πόσα είναι μέσα στο radius που έχω δώσει σαν παράμετρο στην συνάρτηση.Όσα είναι εντός ,επιστρέφω το γειτονικό σημείο σε μια λίστα και την απόσταση του στην δεύτερη λίστα. Αντίστοιχα στο N-nearst neighbours απλά καθώς ψάχνω τα probes (buckets) έχω έναν vector όπου τον γεμίζω με γείτονες και τον ταξινομώ ώστε ο μακρινότερος να είναι στο τέλος. Αν ο vector με τα nearest neighbors γεμίσει και έχω βρει σημείο για υποψήφιο κοντινό , κοιτάω τον τελευταίο γείτονα .Σε περίπτωση που είναι πιο κοντά, πετάω τον παλιό και βάζω στην θέση του τον καινούριο.Έπειτα ταξινομώ και πάλι το ίδιο.(Κάτι σαν min heap).

Cluster Αρχεία και Περιγραφή

Τα Cluster αρχεία θα τα βρείτε στον κατάλογο “Algorithmic-Problems-Project1\algo\kMeans++” και το πρόγραμμα (Cluster_main.cpp) που εκτελεί τις μεθόδους και τον αλγόριθμο βρίσκεται στον κατάλογο “Algorithmic-Problems-Project1\algo”.
Στο αρχείο “kMeans++/Cluster.hpp” θα βρείτε τις δομές, τις μεταβλητές και τις μεθόδους που χρησιμοποιούνται στον αλγόριθμο, καθώς και σχόλια εξηγόντας τι κάνει η κάθε μέθοδο, δομή και σε τι χρησιμεύει η κάθε μεταβλητή.
Στο αρχείο “kMeans++/Cluster.cpp” θα βρείτε την υλοποίηση των μεθόδων kMeans++ (συνάρτηση kMeanspp_Initialization), Lloyd (συνάρτηση Lloyd_method), Silhouette (συνάρτηση Silhouette) και Reverse search (συνάρτηση reverse_assignment), καθώς και την πλήρη περιγραφή τους πάνω από κάθε συνάρτηση.
Όλα υλοποιήθηκαν όπως στις διαφάνειες.

k-Means++ Αρχικοποίηση

Η μέθοδος έχει υλοποιηθεί ως εξής:

- Το πρώτο κεντροειδές επιλέγεται ομοιόμορφα
- Τα υπόλοιπα κεντροειδή επιλέγονται με βάση την πιθανότητα (Πιθανότητα = μικρότερη απόσταση/άθροισμα μικρότερων αποστάσεων)
- μικρότερη απόσταση = η απόσταση του σημείου σε σχέση με το πλησιέστερο μικρότερων κεντροειδή
- Οι υπολογισμοί γίνονται με την ευκλείδεια απόσταση
- Το σημείο με την μεγαλύτερη πιθανότητα (δηλαδή το πιο μακρινό από όλα τα είδη υπάρχοντα κεντροειδή) θα είναι το επόμενο κεντροειδή.

Συνθήκες Σύγκλισης

Ο αλγόριθμος Lloyd για να τερματίσει συγκρίνει όλα τα νέα κεντροειδή με τα προηγούμενα. Συγκεκριμένα συγκρίνει τα στοιχεία τους και αν έχουν αλλάξει λιγότερο από 1% (όλων των κεντροειδών), τότε τερματίζει.
Ο αλγόριθμος reverse τερματίζει όταν η απόσταση μεταξύ του νέου και του προηγούμενου κέντρου είναι μικρότερη από 10 τουλάχιστον στα μισά κεντροειδή.

LSH_Frechet

Παρατήρηση: Στην εντολή 511 στο αρχείο “kMeans++/Cluster.hpp” βάλαμε δ = 10.0 καθώς και παρατηρήσαμε πως αυτή η τιμή είναι αρκετά καλή για χρόνο και απόδοση, αλλά μπορείτε να την αλλάξετε.

Τα αποτελέσματα αλλάζουν ανάλογα με το W και την μεταβλητή δ

Βοηθητικές Συναρτήσεις

Στο αρχείο “Algorithmic-Problems-Project1\algo\functions” υπαρχουν βοηθητικές συναρτήσεις που επαναχρησιμοποιούνται καθόλη την διάρκεια εκτέλεσης των προγραμμάτων.