Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα– Χειμερινό ’23

1η Άσκηση - Readme

1. **Αναζήτηση και συσταδοποίηση διανυσμάτων σε C++.** Το πρόγραμμα υλοποιεί την εύρεση κοντινότερων γειτόνων με δύο διαφορετικούς τρόπους : μέσω της μεθόδου του LSH και μέσω της μεθόδου της προβολής στον υπερκύβο. Ακόμη υλοποιείται και η συσταδοποίηση με αρχικοποίηση μέσω της μεθόδου $Kmeans++$, ενημέρωση μέσω MacQueen και 3 διαφορετικούς τρόπους ανάθεσης Lloyds, Range search with LSH, Range search with Projection in Hypercube.

2. **Λίστα με αρχεία κώδικα και κεφαλίδων και περιγραφή τους**. Η οργάνωση των αρχείων είναι η εξής : το κάθε αρχείο κώδικα στο οποίο δεν υπάρχει κάποια main , συνοδεύεται από το αντίστοιχο αρχείο κεφαλίδας που παρέχει ένα interface μεταξύ άλλων αρχείων και των μεθόδων που υλοποιούνται στο αρχείο κώδικα παρέχοντας επίσης απόκρυψη πληροφορίας. Οπότε, η κεφαλίδες δεν έχουν τίποτα άλλο παρά δηλώσεις και έτσι θα επικεντρωθούμε στην περιγραφή των αρχείων με κώδικα. Παρακάτω είναι μια λίστα με την περιγραφή του κάθε αρχείου και της κεφαλίδας που του αντιστοιχεί σε σειρά που θεωρούμε ότι βοηθάει την καλύτερη κατανόηση του προγράμματος:

   - *StoreTrainData.cpp* : Στην κεφαλίδα του αρχείο υπάρχουν δύο κλάσεις : η κλάση για τις εικόνες και η κλάση που είναι υπεύθυνη για την αποθήκευση των εικόνων που ανήκουν στο training set . Η αναπαράσταση τις εικόνας γίνεται μέσω ενός πίνακα από bytes όπου το κάθε pixel είναι και ένα byte , ενώ υπάρχουν και δύο έξτρα πεδία checked, cluster με τους αντίστοιχους setters, getters που θα μας χρησιμεύσουν στην αναζήτηση και την συσταδοποίησης. Η κάθε εικόνα δημιουργείται byte per byte μέσω της συνάρτησης Insert , όπου κάθε φορά εισάγει ένα byte στον πίνακα των bytes της αναπαράστασης, και τον μεγαλώνει κατά 1. Όσον αφορά την αποθήκευση των εικόνων χρησιμοποιείται η κλάση TrainStore όπου στην ουσία οι εικόνες αποθηκεύονται μέσω ενός πίνακα δεικτών σε εικόνες. Η λογική είναι ότι οι εικόνες εισάγονται μια προς μια μέσω της συνάρτησης Insert . Έπειτα, μπορεί να γίνει προσπέλαση της κάθε εικόνας μέσω ενός integer index , όπου αντιστοιχεί στην θέση της εικόνας κατά την εισαγωγή ξεκινώντας την αρίθμηση από το 0. Δηλαδή, η πρώτη εικόνα που θα εισαχθεί θα έχει $index$ 0, η δεύτερη 1 κ.ο.κ. Στην πραγματικότητα είναι η θέση στον πίνακα των δεικτών, καθώς αυτός μεγαλώνει με την εισαγωγή της κάθε εικόνας. Οπότε μέσω αυτού του index , παρέχεται πρόσβαση στους setters, getters της κλάσης της εικόνας.Σημειώνουμε εδώ, ότι ο χρήστης δεν έχει πρόσβαση σε αυτές τις κλάσεις, το interface που του παρέχεται βρίσκεται στην RedTrainData.cpp .
   - *ReadTrainData.cpp* : Εδώ βρίσκεται η υλοποίηση της συνάρτησης που είναι υπεύθυνη για το άνοιγμα του κατάλληλου αρχείου και διάβασμα του train set, την δημιουργία της κλάσης που περιγράφηκε παραπάνω για την αποθήκευση του, την εισαγωγή των εικόνων στην εν λόγων κλάση και την παροχή του interface προς τον χρήστη. Για το διάβασμα και την αποθήκευση, χρησιμοποιείται η συνάρτηση ReadTrainData που σαν όρισμα παίρνει το μονοπάτι, ως string, στο αρχείο που περιλαμβάνει το dataset . Αφού αρχικά διαβάσει με κατάλληλη τρόπο την κεφαλίδα του αρχείο, δηλαδή τα πρώτα, 16 bytes και τα αποθηκεύσει σε κατάλληλες μεταβλητές όπως αυτές ορίζονται στο MNIST manual , συνεχίζει με το διάβασμα των εικόνων. Για το διάβασμα των εικόνων αρχικά δημιουργείται μια νέα εικόνα, και έπειτα byte per byte διαβάζονται τα pixels της και αποθηκεύονται καταλλήλως μέσω συνάρτησης που παρέχει η κλάση της εικόνας, το σύνολο των pixels προκύπτει από τον πολλαπλασιασμό $columns \cdot rows$, η διαδικασία αυτή επαναλμβάνεται τόσες φορές όσο και το πλήθος των εικόνων στο αρχείο. Το interface με κάθε εικόνα από τον χρήστη γίνεται μέσω του index που υπολογίζεται όπως περιγράφηκε παραπάνω, από εδώ και στο εξής πρόσβαση και οιονεί αναφορά σε κάθε εικόνα γίνεται μέσω index το οποίο παραμένει σταθερό καθ΄ όλη την εκτέλεση του προγράμματος.

- *ReadQueryData.cpp* : Ίδια λογική με το ReadTrainData.cpp μόνο που αφορά του test set, και δεν χρειάζεται interface για τους setters,getters των πεδίων checked, cluster των αποθηκευμένων εικόνων.
- *StoreQueryData.cpp* : Ίδια λογική με το StoreTrainData.cpp
- *hFunc.cpp* : Εδώ βρίσκεται η υλοποίηση της συνάρτησης H του LSH καθώς και η συνάρτηση που υπολογίζει την Ευκλείδεια Απόσταση μεταξύ δύο διανυσμάτων byte . Απλά τα πράγματα , ακολουθούνται ακριβώς οι ορισμοί των διαφανειών. Η συνάρτηση H παίρνει σαν όρισμα το w και το μέγεθος των διανυσμάτων. Σημειώνουμε εδώ, ότι επειδή θέλουμε οι επιστρεφόμενες τιμές της g να είναι θετικές, γίνεται μια κανονικοποίηση στο εσωτερικό γινόμενο, για να είναι πάντα θετικό, μέσω τετραγωνισμού. Ορίζεται για κάθε H ένας τυχαίος πίνακας v και μέσω της συνάρτησης operator παίρνεις το αποτέλεσμα του εσωτερικού γινομένου με αυτόν τον πίανακα.
- *lsh.cpp* : Στην κεφαλίδα του αρχείο υπάρχουν δύο κλάσεις LSH και Hashtable .Η μία αφορά τους πίνακες κατακερματισμού ενώ η άλλη είναι υπεύθυνη για την κατασκευή των πινάκων κατακερματισμού και αποθήκευσης των σημείων σε buckets αλλά και γενικά για όλη την διαδικασία εύρεσης πλησιέστερων γειτόνων. Κατά την δημιουργία ενός αντικειμένου LSH , κατασκευάζεται ένας πίνακας L θέσεων από πίνακες κατακερματισμού (class Hashbucket). Κάθε πίνακας κατακερματισμού δημιουργεί με την σειρά του ένα πίνακα από buckets (class Bucket) και μία g function (class gFunction) η οποία κατασκευάζει K h functions και τυχαία r και μέσω της συνάρτησης FindPosition παίρνεις το αποτέλεσμα της g όπως ακριβώς περιγράφεται στις διαφάνειες του μαθήματος. Ουσιαστικά κάθε hashtable είναι ένας πίνακας απο pointers σε linked list , οπότε καθε bucket λειτουργεί σαν ένα node αυτής της λίστας όπου περιέχει ένα σήμειο. Με την κλήση της συνάρτησης Train , χρησιμοποιείται η συνάρτηση GetRepresentation όπου επιστρέφει τα pixels-bytes κάθε εικόνας.Έπειτα κάθε σήμειο αποθηκεύεται για κάθε hastable σε κάποιο bucket ανάλογα με την g function . Η συνάρτηση KNN επιστέφει ένα πίνακα ο οποίος περιέχει τους κατα προσέγγιση πλησιέστερους γείτονες του query και τις αποστάσεις τους απο αυτο. Αυτό επιδιώκεται με την κλήση μίας συνάρτησης NearestNeighbour της κλάσης LSH έπειτα της κλάσης Hashtables κ.ο.κ ώσπου να φτάσει σε κάποιο bucket και μέσω της ευκλείδιας απόστασης να πάρει το επιθυμητό αποτέλεσμα. Για κάθε γείτονα που επιλέγεται υπάρχει η συνάρτηση SetChecked ώστε να μην λαμβάνεται ο γείτονας αυτός υπόψη στις επόμενες επαναλήψεις . Με παρόμοιο τρόπο έχουν κατασκευαστεί και οι συναρτήσεις AKNN και RangeSearch.
- *lshmain.cpp* : Εδώ βρίσκεται η main συνάρτηση για τον αλγόριθμο LSH.Ο κώδικας αρχικοποιεί με προεπιλεγμένες τιμές τις μεταβλητές K, L, N, R και στη συνέχεια πραγματοποιεί loop για τα ορίσματα της γραμμής εντολών (εάν υπάρχουν) και τα επεξεργάζεται. Αναζητά συγκεκριμένα flags όπως -d, -q, -k, -L, -o, -N και -R και προσαρμόζει τις αντίστοιχες μεταβλητές ανάλογα με βάση τις τιμές που παρέχονται στα ορίσματα της γραμμής εντολών. Στην συνέχεια καλείται η συνάρτηση ReadTrainData για το inputfile και η συνάρτηση ReadQueryData για το queryfile όπου αναλύθηκαν παραπάνω. Έπειτα δημιουργείται ενα αντικείμενο LSH και εκτελείται η συνάρτηση Train όπως αναλύθηκε παραπανω. Για κάθε query εκτελούνται οι συναρτήσεις KNN και AccurateKNN καταγράφοντας τον χρόνο που απαιτείται και γράφοντας τα αποτελέσματα σε ένα αρχείο εξόδου outputfileName. Με βάση αυτά που επέστρεψαν οι δύο αυτές συναρτήσεις (αριθμός του N-οστού προσεγγιστικά πλησιέστερου γείτονα που βρέθηκε και την απόπσταση του απο το query , απόσταση από το αληθινά N-οστό πλησιέστερο γείτονα αντίστοιχα) γίνονται οι ανάλογες εκτύπωσεις στο αρχείο εξόδου. Καλείται η συνάρτηση RangeSearch βρίσκοντας και εκτυπόνωντας τους γείτονες εντός μιας ακτίνας R . Τέλος το πρόγραμμα ρωτά τον χρήστη εάν θέλει να τερματίσει το πρόγραμμα και αν όχι, ζητά από τον χρήστη ένα νέο queryfile . Αυτός ο βρόχος συνεχίζεται όσο ο χρήστης δεν εισάγει "y" για τερματισμό.
- *RandomProjection.cpp* : Εδώ βρίσκεται η υλοποίηση της προβολής στον υπερκύβο, το interface προς τον χρήστη είναι το ίδιο με αυτό του LSH , δηλαδή πρώτα αρχικοποιεί ένα αντικείμενο τύπου

RandomProjection , και στην συνέχεια μπορεί να καλέσει τις συναρτήσεις που ζητούνται βάζοντας ως input τα queries . Τα ορίσματα για τον constructor είναι τα k, m, probes όπως περιγράφονται στην εκφώνηση. Επειδή ο υπερκύβος περιλαμβάνει ένα σύνολο F συναρτήσεων που η τιμή τους αποφασίζεται τυχαία αλλά παραμένει ίδια για όλο το πρόγραμμα, έχουμε δημιουργήσει μια κλάση FValues , η οποία έχει δύο πεδία, ένα για το input ένα για την τιμή του κάθε input (f(5) = 3)καθώς και τους κατάλληλους setters, getters . Με την σειρά της, η κλάση FunctionF , κρατά μια συνδεδεμένη λίστα από FValues , έτσι καθώς δέχεται σαν όρισμα ένα point , παίρνει την τιμή από την κατάλληλη LSH , και έπειτα ελέγχει αν αυτή η τιμή βρίσκεται σε κάποιο κόμβο της συνδεδεμένης λίστας, αν ναι επιστρέφει αν όχι δημιουργεί νέο κόμβο. Όλα τα υπόλοιπα είναι απλές υλοποιήσεις αυτών που ζητά η εργασία στην λογική του LSH

- *cube.cpp* : Τίποτα δύσκολο εδώ, ό,τι ζητάει η εκφώνηση πάλι στην λογική της LSH main.
- *kmeans.cpp* : Σε αυτό το αρχείο βρίσκεται η υλοποίηση του μέσω Kmeans. Η βασική ιδέα είναι ότι όλα γίνονται μέσω του Constructor για τον Kmeans, ο οποίος παίρνει σαν όρισμα όλες τις μεταβλητές για όλες τις μεθόδους, δηλαδή το K το LSH, Hypercube, το L κ.ο.κ. ανεξάρτητα από το αν θα τα χρησιμοποιήσει ή όχι, έτσι κι αλλιώς παίρνουν default τιμές. Οπότε το μόνο το οποίο παρέχεται στον χρήστη ως interface είναι η συνάρτηση του constuctor . Ο constructor της Kmeans αρχικοποιεί K το πλήθος clusters , όπου η κλάση cluster περιλαμβάνει το σύνολο των γειτόνων (ως indexes , όπως περιγράφηκαν παραπάνω) και του κέντρο. Παρέχονται ακόμη από τη κλάση cluster οι πλέον τετριμμένες κλάσεις που αφορούν setters, getters κτλπ. Αφού κληθεί ο constructor Kmeans , γίνεται η κοινή αρχικοποίηση, και μετά ανάλογα με την μέθοδο καλείται η κατάλληλη συνάρτηση για την ανάθεση : Lloyds ή Range Searchπου στην Range Search ανάλογα δημιουργείται LSH ή Hypercube ,και μετά προχωράμε σύμφωνα με τους αλγορίθμους των διαφανειών. Η σιλουέτα υπολογίζεται πάλι βάσει των διαφανειών.
- *KMeansMain.cpp* : Ό,τι ζητείται στην εκφώνηση, το μόνο για το οποίο είναι υπεύθυνη η main είναι η δημιουργία ενός αντικειμένου τύπου KMeans με τα κατάλληλα ορίσματα.

3. **Οδηγίες μεταγλώττισης του προγράμματος.** Υπάρχουν 3 κανόνες για την δημιουργία 3 ξεχωριστών εκτελέσιμων, (που ωστόσο χρησιμοποιούν κάποια κοινά αρχεία), μέσω seperate compilation , παρακάτω είναι τα εκτελέσιμα που δημιουργούνται με τους αντίστοιχους κανόνες :

   - *lsh* : μέσω της make lsh
   - *cube* : μέσω της make cube
   - *cluster* : μέσω της make cluster
   - *διαγραφή .o και εκτελέσιμων* : μέσω της make clean

4. **Οδηγίες χρήσης του προγράμματος** : Ο τρόπος χρήσης του προγράμματος είναι αυτός που ζητάει η εκφώνηση, αν κάποιο input είναι λάθος, π.χ αρνητικό K ή το αρχείο δεν βρεθεί, εμφανίζεται μήνυμα λάθους και το πρόγραμμα τερματίζει. Για τις LSH, Hypercube ο χρήστης μπορεί να επαναεισάγει αρχείο αναζήτησης - query , για τις ίδιες παραμέτρους και το ίδιο dataset , άμα θέλει να αλλάξει κάποιο από τα τελευταία δύο πρέπει να τερματίσει και να εκτελέσε εκ νέου το πρόγραμμα. Για το clustering , μόλις τελειώσει και εκτυπώσει το πρόγραμμα τερματίζει.

5. **Στοιχεία φοιτητών**

   - Απόστολος Κουκουβίνης, ΑΜ : 1115202000098
   - Γιώργος Μητρόπουλος, ΑΜ : 1115202000128