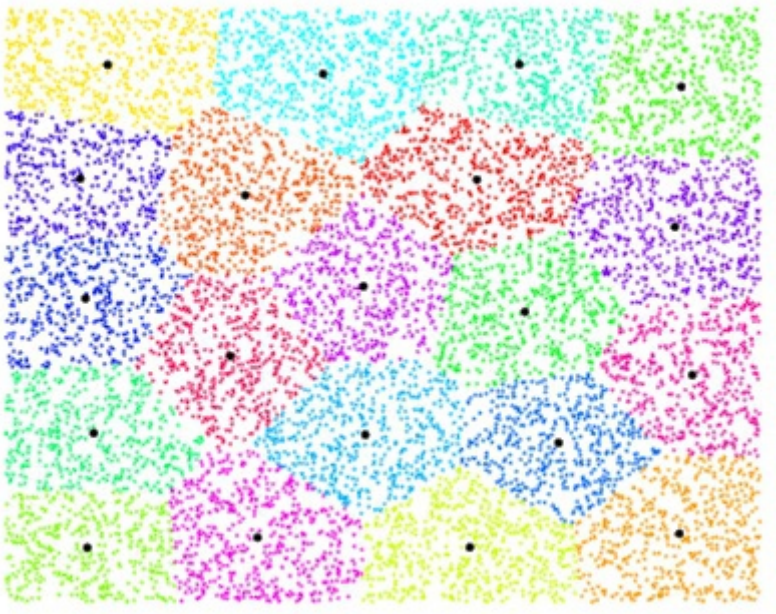
Ανάπτυξη Λογισμικού για Δυσεπίλυτα Αλγοριθμικά Προβλήματα

2019-2020

Ανάπτυξη Λογισμικού για Clustering

Υλοποίηση των Αλγορίθμου Συσταδοποίησης k-means & k-medoids για διανύσματα & πολυγωνικές καμπύλες

των φοιτητών
Δημήτρη Αλεξανδρή & Ιωάννας Ζαπαλίδη
1115201400006 & 1115201400044



Περιεχόμενα

Ανάπτυξη και Δομή Κώδικα

Χρησιμοποιησαμε git και GitHub για το version control κατά την εκπόνηση της εργασίας μας.

Τα αρχεία μας:

1. `Makefile` – *με οδηγίες μεταγλώττισης και εκτέλεσης*
2. `cluster.cpp` – η main μας
3. `inits.hpp` – *οι συναρτήσεις αρχικοποίησης των κέντρων των clusters*
4. `assign.hpp` – *οι συναρτήσεις assignment των στοιχείων σε κάθε cluster*
5. `update.hpp` – *οι συναρτήσεις ενημέρωσης των κέντρων των clusters*
6. `cluster_object.h/cluster_object.cpp` – *η κλάση των clusters*
7. `h_funs.h/h_funs.cpp` – *οι συναρτήσεις* h_i
8. `g_funs.h/g_funs.cpp` – *οι συναρτήσεις* g_i
9. `ht.h/ht.cpp` – *δική μας κλάση hash table (για LSH - διανύσματα)*
10. `my_vector.h/my_vector.cpp` – *δική μας κλάση διανυσμάτων*
11. `NNpair.h/NNpair.cpp` – *ζεύγη διανυσμάτων που είναι κοντινότεροι γείτονες & η απόστασή τους*
12. `curve_point.h/curve_point.cpp` – *κλάση με τα σημεία των καμπυλών*
13. `curve.h/curve.cpp` – *κλάση των καμπυλών*
14. `grid.h/grid.cpp` – *δομή grid*
15. `curve_ht.h/curve_ht.cpp` – *δική μας κλάση hash table για καμπύλες*
16. `readme.pdf` – *αυτό το αρχείο*
17. `utils.h` – *συμπληρωματικές συναρτήσεις που χρησιμοποιούμε*
18. `catch.hpp` – *unit testing module (Catch)*
19. `ourunit.cpp` – *unit testing module (δικό μας)*

Οδηγίες μεταγλωττισης:
Στον φάκελο της εργασίας εκτελούμε τα εξής:

- `$make`
  Παράγει το εκτελέσιμο αρχείο (και τα .o αρχεία)
- `$make clean`
  Διαγράφει το εκτελέσιμο αρχείο (και τα .o αρχεία)
- `$make test`
  Παράγει το εκτελέσιμο αρχείο για ελέγχους
- `$make runtest`
  Εκτελεί το αρχείο ελέγχων

Καθολικές Σχεδιαστικές Επιλογές:

- Κάθε διαδικασία που ζητήθηκε να υλοποιηθεί, αποτελεί μία δική της συνάρτηση (σε ορισμένες περιπτώσεις χρησιμοποιήθηκαν και βοηθητικές συναρτήσεις).

- Στο αρχείο utils.h συμπεριλαμβάνονται οι συναρτήσεις για την μετρική Manhattan, την DTW, και δύο βοηθητικές συναρτήσεις που από τα εκάστοτε ευρεθέντα κέντρα, αρχικοποιούν τα clusters (μία για διανύσματα και μία για καμπύλες).
- Έχουμε στα αρχεία inits.hpp, assign.hpp & update.hpp εξίσου τις υλοποιημένες μεθόδους για διανύσματα και για καμπύλες.
- Στα αρχεία cluster_object.h/cluster_object.cpp έχουμε υλοποιήσει διαφορετικές δομές για τα clusters των διανυσμάτων και των καμπυλών λόγω διαφορετικών αναγκών των δομών.
- Για unit testing, χρησιμοποιήσαμε την βιβλιοθήκη Catch, και οι έλεγχοι γίνονται σε διαφορετικό αρχείο.
- Το πρόγραμμά μας, στην περίπτωση που συναντήσει άδειο cluster, βάζει σε αυτό το μακρυνότερο σημείο του μεγαλύτερου cluster από τα άλλα, ώστε να επιτευχθεί η εύρυθμη συνέχεια της εκτέλεσης, όπως προτάθηκε στο eclass.

Initialization 1: Random Selection

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `inits.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `cluster_object.h/cluster_object.cpp`

Η συνάρτηση `initialize_centers` στο αρχείο `inits.hpp` αφορά σε αυτήν την διεργασία. Αρχικά δημιουργούμε δομές αποθήκευσης των indexes των κέντρων στην δομή αποθήκευσης όλων των στοιχείων μας, και έπειτα βάσει της uniform κατανομής βρίσκουμε τυχαία indexes, κατάλληλου πλήθους, που όταν περνιούνται στην δομή αποθήκευσης όλων των στοιχείων μας, επιστρέφει τα τυχαία μας vectors ή curves που θα είναι τα κέντρα των clusters μας.

Σχεδιαστικές επιλογές:

- Μετατρέπουμε το unordered map των dataset στοιχείων μας σε vector για ευκολότερο χειρισμό των στοιχείων.
- Αξιοποιούμε την uniform real distribution της C++ όπως και στην προηγούμενη άσκηση για την εύρεση τυχαίων τιμών, ομοιόμορφα κατανεμημένων. Αυτές για να μετατραπούν σε int, γίνονται floor.

Initialization 2: K means ++

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `inits.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `cluster_object.h/cluster_object.cpp`

Η συνάρτηση `initialize_centers_plus` στο αρχείο `inits.hpp` αφορά σε αυτήν την διεργασία. Αρχικά δημιουργούμε δομές αποθήκευσης των indexes των κέντρων στην δομή αποθήκευσης όλων των στοιχείων μας, και έπειτα βάσει της uniform κατανομής βρίσκουμε τυχαία το 1ο index μας. Έπειτα, μέχρι τα κέντρα να φτάσουν το κατάλληλο πλήθος, για κάθε σημείο που δεν είναι ήδη κέντρο, βρίσκουμε τις αποστάσεις του από κάθε κέντρο, κρατάμε σε έναν πίνακα αυτές τις αποστάσεις (υψωμένες στο τετράγωνο & διαιρεμένες με την μέγιστη για κανονικοποίηση, όπως λένε οι σημειώσεις του μαθήματος). Τέλος, με δυαδική αναζήτηση βρίσκουμε (με την κατάλληλη πιθανότητα) το επόμενο κέντρο.

Σχεδιαστικές επιλογές:

- Χρησιμοποιούμε την βοηθητική συνάρτηση δυαδικής αναζήτησης (που βρίσκεται στο ίδιο αρχείο) για την εύρεση στο array P του κατάλληλου σημείου, όπως λέει η εκφώνηση.
- Αξιοποιούμε την δομή distance matrix όπου αποθηκεύονται οι αποστάσεις όλων των στοιχείων αναμεταξύ τους. Για να μην υπολογίζονται μη χρησιμοποιούμενες αποστάσεις, κάθε φορά ελέγχεται βάσει index αν έχει υπολογιστεί κάποια απόσταση: αν ναι, με απλό access του κατάλληλου σημείου του πίνακα βρίσκουμε την τιμή της, αν όχι, υπολογίζεται & έπειτα αποθηκεύεται στην κατάλληλη θέση.

Assignment 1: Lloyd’s Assignment

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `assign.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `cluster_object.h/cluster_object.cpp`

Βάσει του input dataset & των clusters που παίρνει ως όρισμα, η `lloyd_ass` βρίσκει τις ελάχιστες αποστάσεις κάθε στοιχείου του dataset με τα κέντρα των clusters αυτών και έτσι ενημερώνει τις δομές αυτές, που ως τότε δεν περιέχουν παρά μόνο τα κέντρα τους.

Σχεδιαστικές επιλογές:

- Εφόσον οι υπολογισμοί αποστάσεων είναι λιγότεροι, δεν χρησιμοποιήσαμε δομή distance matrix, αλλά κάθε φορά υπολογίζουμε τις manhattan αποστάσεις από τα κέντρα.

Assignment 2: Range Search with LSH (inversive assignment)

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `assign.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `h_funs.h/h_funs.cpp`
- `g_funs.h/g_funs.cpp`
- `ht.h/ht.cpp`
- `cluster_object.h/cluster_object.cpp`

Η συνάρτηση `LSH_range_ass` χρησιμοποιεί τις δομές `NN_pairs, h_funs, g_funs & ht` από την προηγούμενη εργασια μας. Αρχικά υπολογίζεται το w, όπως στην 1η εργασία, έπειτα το μέγεθος του table των ht του LSH μας. Έπειτα ακολουθείται η ίδια διαδικασία με την πρώτη εργασία, μόνο που μέσα σε ένα bucket, δεν κρατάμε μόνο τον κοντινότερο γείτονα κάθε στοιχείου αλλά όσους έχουν απόσταση μέχρι radius. Εδώ, τα input στοιχεία μας hash-άρονται στη δομή μας, και τα “query” στοιχεία είναι τα κέντρα των clusters. Σε έναν μετρητή κρατάμε το πλήθος των unassigned στοιχείων, δηλαδή πόσα από τα input στοιχεία μας δεν έχουν βρεθεί σε κάποιο cluster. Επιπλέον, έχουμε έναν μετρητή kill_countdown, ο οποίος βοηθάει στο να μην προκύψει ατέρμον loop, δηλαδή όταν δεν παρατηρηθεί νέα ανάθεση για κάποιες συνεχόμενες φορές, λήγει την διαδικασία. Τα υπόλοιπα σημεία ανατίθενται στο κοντινότερό τους κέντρο.

Σχεδιαστικές επιλογές:

- Χρησιμοποιήσαμε την βοηθητική συνάρτηση initialize_radius, η οποία υπολογίζει την μέση απόσταση των κέντρων των clusters, και επιστρέφει το μισό αυτής, που γίνεται το όρισμα της ranged search.

Update 1: PAM a la Lloyds

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `update.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `h_funs.h/h_funs.cpp`
- `g_funs.h/g_funs.cpp`
- `ht.h/ht.cpp`
- `cluster_object.h/cluster_object.cpp`

Η συνάρτηση `update_pam` παίρνει ως όρισμα δείκτη στα υπάρχοντα clusters. Για κάθε cluster, εκτελεί επαναλήψεις για κάθε στοιχείο αυτού: για κάθε πιθανό νέο κέντρο, διατηρεί την μέση απόστασή του από κάθε σημείο του cluster, παίρνει την ελάχιστη τιμή, και το σημείο στο οποίο αντιστοιχεί αυτή είναι το νέο κέντρο του cluster αυτού.

Σχεδιαστικές επιλογές:

- Αγνοούμε το υπάρχον κέντρο στην λούπα των x, γλιτώνοντας πράξεις, αλλά δεν το αγνοούμε στην εσωτερική λούπα των y, καθώς συνεισφέρει στην υπολογιζόμενη μέση απόσταση.

Update 2: Υπολογισμός Mean Vector/DTW Centroid Curve

Χρησιμοποιούμε τα αρχεία:

- `Makefile`
- `cluster.cpp`
- `update.hpp`
- `my_vector.h/my_vector.cpp`
- `NNpair.h/NNpair.cpp`
- `utils.h`
- `h_funs.h/h_funs.cpp`
- `g_funs.h/g_funs.cpp`
- `ht.h/ht.cpp`
- `cluster_object.h/cluster_object.cpp`

Η συνάρτηση `update_mean` παίρνει ως όρισμα δείκτη στα υπάρχοντα clusters, καθώς και τον αριθμό των διαστάσεων του διανυσματικού χώρου. Για κάθε cluster, βρίσκει τον γεωμετρικό μέσο των διανυσμάτων του, ο οποίος γίνεται το νέο κέντρο του cluster αυτού.

Σχεδιαστικές επιλογές:

- Στα διανύσματα που δημιουργούνται από αυτήν την μέθοδο θέτουμε ως όνομα τον στηλοθέτη (“`\t`”), όνομα που αποκλείεται να είναι ήδη δωθέν στο input dataset μας, αλλά και που δεν δημιουργεί πρόβλημα στις assign & update συναρτήσεις μας.
- Δεν υλοποιήθηκε για καμπύλες.

Μετρήσεις & Συγκρίσεις

Παρακάτω παραθέτουμε τις τιμές Silhouette που παρατηρήθηκαν κατά μέσο όρο στην εκτέλεση των αρχείων μας.

Για διανύσματα:

Συνδυασμός Μεθόδων	Πλήθος Cluster	Μέση τιμή Silhouette
I1 A1 U1	5	0.696291
I1 A1 U2	5	0.904918
I1 A2 U1	5	0.739129
I1 A2 U2	5	0.904915
I2 A1 U1	5	0.654969
I2 A1 U2	5	0.798243
I2 A2 U1	5	0.921908
I2 A2 U2	5	0.798731
I1 A1 U1	10	0.546828
I1 A1 U2	10	0.906466
I1 A2 U1	10	0.581839
I1 A2 U2	10	0.904408
I2 A1 U1	10	0.534323
I2 A1 U2	10	0.909043
I2 A2 U1	10	0.351964
I2 A2 U2	10	0.907207
I1 A1 U1	15	0.676645
I1 A1 U2	15	0.789617
I1 A2 U1	15	0.620748
I1 A2 U2	15	0.885241
I2 A1 U1	15	0.655665
I2 A1 U2	15	0.826117
I2 A2 U1	15	0.601245
I2 A2 U2	15	0.824363

Συνδυασμός Μεθόδων	Μέση τιμή Silhouette
I1 A1 U1	0.639921
I1 A1 U2	0.867001
I1 A2 U1	0.647238
I1 A2 U2	0.898188
I2 A1 U1	0.614985
I2 A1 U2	0.844467
I2 A2 U1	0.625039
I2 A2 U2	0.843433

Γενικά, καλύτερα αποτελέσματα είχαν οι μέθοδοι με την update (2) – εύρεση μέσου διανύσματος ως νέο κέντρο. Επιπλέον, η αρχικοποίηση (2) (initialize kmeans++) οδηγεί σε σύγκλιση γρηγορότερα κατά μέσο όρο. Στην περίπτωση που εμφανιστεί άδειο cluster, ο αλγόριθμός μας το χειρίζεται με επιτυχία, ωστόσο πρέπει να σημειωθεί ότι αυτό έχει επίπτωση στην τιμή του Silhouette.

Για καμπύλες:

Συνδυασμός Μεθόδων	Πλήθος Cluster	Μέση τιμή Silhouette
I1 A1 U1	5	0.379100
I1 A2 U1	5	0.332768
I2 A1 U1	5	0.430642
I2 A2 U1	5	0.374967
I1 A1 U1	10	0.417986
I1 A2 U1	10	0.447294
I2 A1 U1	10	0.454282
I2 A2 U1	10	0.534805
I1 A1 U1	15	0.406678
I1 A2 U1	15	0.441970
I2 A1 U1	15	0.451835
I2 A2 U1	15	0.443765

Συνδυασμός Μεθόδων	Μέση τιμή Silhouette
I1 A1 U1	0.401255
I1 A2 U1	0.407344
I2 A1 U1	0.445586
I2 A2 U1	0.451179

Βιβλιογραφία

- http://www.cplusplus.com/
- https://en.cppreference.com/
- Σημειώσεις του μαθήματος
- https://www.geeksforgeeks.org/
- Τα αρχεία της προηγούμενης εργασίας μας
- Catch