Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα: Project 3

Σταματόπουλος Βασίλειος 1115201400188

3/12/2018

1 Εισαγωγή

Στην εργασία αυτή υλοποιήθηκαν 4 διαφορετικοί τρόποι recoommendation, σε χρήστες του twitter για κρυπτονομίσματα τα οποία δεν έχουν ξαναδεί, όπως ακριβώς ζητήθηκε και στην εκφώνηση. Δηλαδή, 2 τρόπους για την **πρόταση 5 κρυπτονομισμάτων** μέσω του διανύσματος συναισθήματος, προς κάθε κρυπτονόμισμα -για κάθε χρήστη και 2 τρόπους για την **πρόταση 2 κρυπτονομισμάτων** μέσω του διανύσματος συναισθήματος που προκύπτει από την συσταδοποίηση των τιτιβισμάτων στην **Εργασία 2**. Η παρούσα εργασία, ακολουθεί τις σαφείς οδηγίες που δώθηκαν στην εκφώνηση, ενώ έχουν υλοποιηθεί όλα τα ζητουμενα πλην του 10-fold cv λόγω αδυναμίας χρόνου.

2 Αρχέια

Ο φάκελος περιέχει τα κάτωθι αρχεία στα οποία έγινε η υλοποίηση του κώδικα.

1. **recommendation.c:** Περιέχει την main function στην οποία με τη σειρά διαβάζουμε τα arguments. ύστερα διαβάζουμε το αρχείο εισόδου και στη συνέχεια αρχικοποιούμε τις μεταβλητές που θα χρειαστούμε. Δημιουργούνται αρκετές δομές δεδομένων, οι οποίες αποτελούν την βάση των δεδομένων μας, και αυτές είναι:

```
typedef struct tweet{
int id;
int user_id;
int no_of_words;
node_t * coin_list;
double * coordinates;
char ** words;
int cluster_id;
double score;
} *tweet;
```

```
typedef struct user{
int id;
int no_of_tweets;
struct user * next;
tweet * tweets;
score_array * score_vector;

//centroid stuff
int centroid_id;
int centroid2_id;
double dist; //distance from centroids
double dist_as_centroid;
double silhouette;
} *user;

typedef struct cluster{
tweet* tweets;
int size;
} * cluster;
```

Γενικότερα, χρησιμοποιήθηκαν αρκετές δομές δεδομένων για την εργασία αυτή, οι οποίες βρίσκονται στο αρχείο **hashtable.h** και γίνονται κατανοητές από την υπογραφή τους. Πιο συγκεκριμένα, χρησιμοποιήθηκαν δομές για την φύλαξη, ανάγνωση και επεξεργασία των κρυπτονομισμάτων καθώς και μια δομή λεξικού η οποία μετατρέπει έναν αριθμό (id) σε όνομα κρυπτονομίσματος. Στην main, καλούνται οι 4 συναρτήσεις που κάνουν την εργασία (μία για κάθε ερώτημα) και είναι αντίστοιχα οι εξής

(a) cosine_lsh_implementation(users, no_of_users, p , 0, dict ,output);

(b) cosine_cluster_implementation(users2, no_of_users, 100, k, p, type, dict, output);

(c) clustering_lsh_implementation(users3, clusters, no_of_users, no_of_clusters, p, dict, output);

(d) clustering_cluster_implementation(users4, clusters, no_of_users, no_of_clusters, p, dict, output);

2. **functions.c:** Στο αρχείο αυτό βρίσκουμε όλες τις συναρτήσεις που χρησιμοποιούνται στην εργασία με μια μεγάλη πλειοψηφία εξάυτών να προέρχονται από τον φάκελο της πρώτης εργασίας και της δεύτερης εργασίας. Λόγω του μεγάλου όγκου του αρχείου, οι συναρτήσεις δεν θα περιγραφούν πλήρως, αλλά θα δωθεί η βασική δομή του δένδρου και της σειράς με την οποία καλούνται. Επιπλεόν, σε ολή την έκταση του αρχείου υπάρχουν τα απαραίτητα

σχόλια για την κατανόησή του από τρίτους.
Καλώντας την **cosine_lsh_implementation**, ο αλγόριθμος εκτελεί την ίδια ακριβώς διαδικασία που εκτελούσε στις προηγούμενες εργασίες, απλά αντί για **σημεία** εφαρμόζεται επάνω στη δομή **user**.Ουσιαστικά, η βασική διαφορά που έχουν οι δύο πρώτοι αλγόριθμοι από τους αντίστοιχους τους στις δύο προηγούμενες εργασίες, είναι οτι δεν υπάρχει κάποιο συγκεκριμένο **query**, αλλά στη θέση του έχουμε τον κάθε χρήστη ενάντια τον υπολοίπον, για να βρούμε τους γειτόνους του. Αυτό επιτυγχάνεται με την ταξινόμηση των **άγνωστων κρυπτονομισμάτων** κάθε χρήστη, με χρήση της **qsort**, και την επιλογή των 5 πρώτων από τον πίνακα.
Η διαδικασία για την cosine_cluster_implementation είναι παρόμοια.
Οι πιο περίπλοκες συναρτήσεις ήταν αυτές του Β ερωτήματος στις οποίες έπρεπε να δημιουργήσουμε τα επιπλέον διανύσματα c_j και να τα αντιπαραβάλουμε με τους χρήστες που ανοίκουν σε κάθε συστάδα προσέχωντας πάντα να μην τους προτίνουμε κάποιο νόμισμα που γνωρίζουν ήδη.
Αυτό επιτυγχάνεται αφού πάρουμε τον πίνακα που περιέχει τα προτινόμενα νομίσματα κάθε χρήστη και εφαρμόζοντας τον παρακάτω αλγόριθμο. Ο οποίος για κάθε χρήστη και κάθε συστάδα, ψάχνει τα νομίσματα μέχρι να βρει δύο των οποίων η τιμή είναι μηδενική (δηλαδή ο χρήστης δεν έχει αναφερθεί σε αυτά ακόμη).

```
for(i = 0; i < no_of_users;  i++){
fprintf(o ,"<%d> ", users[i]->id);
for(j = 0; j < no_of_clusters; j++){
int count = 0; //count till 2
for(z = 0; z < 8; z++){
if(count < 2){
int k;
int flag = 0;
for(k = 0; k < users[i]->no_of_tweets; k++){
if(users[i]->score_vector[coins[j][z]]->value != 0){ // filter here
flag = 1;
break;
}

}
if(flag == 0){
fprintf(o, "%s ", dict[coins[j][z]]);
count++;
}
}
else break;
}
if(count >= 2){
break;
}
```

```
}
fprintf(o, "\n");
```

3. **hashtable.c hashtable.h** Τα αρχεία αυτά περιέχουν τους τύπους των στοιχείων που χρησιμοποιήθηαν καθώς και τις απαραίτητες συναρτήσεις για να λειτουργήσει ένας πίνακας κατακερματισμού, όπως οιinsert, print. Όπως τα δύο προηγούμενα, είναι κατάλληλα σχολιασμένα για τρίτους.

3 Μεταγλώττιση

Το πρόγραμμα χρησιμοποιεί makefile, συνεπώς μπορεί να μεταγλωττιστεί με την εντολή make.

4 Εκτέλεση

Η εκτέλεση του αρχείου ακολουθεί το πρότυπο που δώθηκε στην εκφώνηση

5 Version Control

Για το πρόγραμμα χρησιμοποιήθηκε το github για να γίνει το απαραίτητο version control.[1]

6 Αποτελέσματα

Τα αποτελέσματα βρίσκονται στο αρχείο **output.txt**

6.1 Πορίσματα

Καταρχήν, όταν τρέχουμε τον αλγόριθμο και με τις 4 συναρτήσεις, η 2η εμφανίζει σφάλμα ότι κάποια συστάδα είναι άδεια. Αυτό μάλλον σχετίζεται με το γεγονός ότι υπάρχει τυχαιότητα στη διαδικασία, και η μόνη λύση που βρήκα είναι να τρέξω το A2 ερώτημα ξεχωριστά από τα υπόλοιπα 3 για το output. Παρ΄όλα αυτά, το πρόγραμμα έδωσε αρκετά καλά και αναμενόμενα αποτελέσματα. Υπάρχει ομοιότητα σε κάποια σημεία μεταξύ των 4 αλγορίθμων, τουλάχιστον όσον αφορά μια γρήγορη ματιά.

Tested on Linux Ubunut 18.03

References

[1] Vasileios Stamatopoulos, Github
`https://github.com/billstam12/Project/tree/master/2`