ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Project: Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Τελική αναφορά

Ομάδα Εργασίας

Φαραώ Γεώργιος **1115201700177**

Καζάκος Βασίλειος **1115201700040**

Στιβακτάς Εμμανουήλ **1115201700152**

Περιεχόμενα

Σκοπός της Εργασίας

Σκοπός του project αυτού είναι να βρούμε προϊόντα που ταιριάζουν (στην περίπτωσή μας json files) από ποικίλες εγγραφές που βρίσκονται σε πολλές ιστοσελίδες. Το πρόβλημα αυτό είναι γνωστό στην βιβλιογραφία ως entity resolution or disambiguation. Πιο συγκεκριμένα, μας δίνονται φάκελοι που περιέχουν αρχεία json. Κάθε φάκελος αντιπροσωπεύει μία ιστοσελίδα διαδικτυακών πωλήσεων (π.χ. www.ebay.com) και κάθε json αρχείο αντιπροσωπεύει ένα προϊόν που πωλείται από τις ιστοσελίδες αυτές. Επιπλέον μας δίνεται ένα αρχείο .csv που περιέχει id προϊόντων σε δυάδες. Τα id αυτά χρησιμοποιούνται για να ταιριάξουμε προϊόντα με βάση την μεταθετική ιδιότητα.

Μηχάνημα

Οι παρακάτω μετρήσεις έγιναν σε Ryzen 5 2600, 16 GB RAM.

Γενικές Δομές

Οι βασικές δομές που χρησιμοποιήθηκαν στα τρία μέρη της εργασίας ήταν τα Red Black Trees, τα Hash Tables και απλά συνδεδεμένες λίστες. Τα Red Black Trees προτιμήθηκαν για την αποθήκευση και την αναπαράσταση δεδομένων, όπως για παράδειγμα τα αρχεία json, καθώς πρόκειται για ισοζυγισμένα δέντρα που η προσπέλαση τους πραγματοποιείται σε χρόνο O(logn). Οι λίστες είχαν ως κύρια λειτουργία την αποθήκευση κάποιων επιπλέων στοιχείων όπως τα χαρακτηριστικά του κάθε αρχείου. Προτιμήσαμε μία τέτοια απλή δομή, όταν δεν υπήρχε η ανάγκη της ταξινόμησης-αναζήτησης, όπως για παράδειγμα οι κόμβοι μια κλίκας. Ακόμη, τα Hash Tables χρησιμοποιήθηκαν κατά κόρων, λειτουργώντας ώς ένα πρώτο επίπεδο indexing, με το κύριο προτέρημα την αναζήτηση ενός στοιχείου σε μέσο χρόνο O(1)*O(logn/M), με M τις θέσεις του Hash Table και την χρήση μια καλής συνάρτηση κατακερματισμού. Να τονιστεί, πως τα Collisions τα διαχειριζόμαστε με Red-Black-Trees. Συνεπώς, χρησιμοποιούμε δομές που θα προσφέρουν πολύ γρήγορη προσπέλαση των στοιχείων, μιας και αυτή είναι η πιο σημαντική λειτουργία και στα τρία μέρη, ειδικά όταν έχουμε σαν είσοδο, χιλιάδες δεδομένα. Για αυτό τον λόγο, προτιμήσαμε τα Hash Tables και τα Red Black δέντρα.

Μέρος 1

Η κύρια λειτουργία του αλγορίθμου για τις θετικές συσχετίσεις είναι κάθε στιγμή όταν έρχεται μία θετική συσχέτιση ανάμεσα σε 2 προϊόντα διαφορετικών κλικών, όλα τα στοιχεία από τις 2 κλίκες να γίνονται concatenate σε μία μεγαλύτερη κλίκα.

Χρόνος (seconds) (user+kernel)	**Συνολική Μνήμη (bytes)**	**Πλήθος θετικών συσχετίσεων**	**Αρχείο**
2.8	3.494.284.891	3582	Μικρό
3,01	3.494.298.835	42535	Μεγάλο

Μέρος 2

Στο δεύτερο μέρος του project αρχικά δημιουργήσαμε και κλίκες αρνητικών συσχετίσεων με σκοπό να χρησιμοποιηθούν στην εκπαίδευση του μοντέλου μηχανικής μάθησης.

Πραγματική αναπαράσταση μνήμης: Μία κλίκα αποτελείται από μία λίστα δεικτών προς τους κόμβους που ανήκουν στην ίδια κλίκα, και ένα δέντρο με δείκτες προς τις κλίκες που είναι διαφορετικές από την κλίκα μας.

Χρησιμοποιούμε το μεγάλο αρχείο εισόδου και κρατάμε όλες τις στήλες λέξεων που δημιουργούνται μετά την προεπεξεργασία (Μέγεθος vocabulary: περίπου 29500 λέξεις).

Είναι φυσικό ο πίνακας TF-IDF ενός αρχείου να περιέχει πολλά μηδενικά καθώς τα αρχεία κατα μέσο όρο έχουν 100 διαφορετικές λέξεις. Έχει γίνει κατάλληλο optimization στο μοντέλο ώστε να διαχειρίζεται τα μηδενικά αυτά. Πιο συγκεκριμένα

Χρησιμοποιήθηκε το μεγάλο αρχείο.
Επειδή το valgrind αδυνατούσε να μετρήσει τη μνήμη, η μέτρηση της έγινε αντικαθιστώντας όλες τις malloc() με τη παρακάτω συνάρτηση.

```
size_t globalmemory;
void * myMalloc(size_t size)
{
   globalmemory = globalmemory +size;
   return malloc(size);

}
```

με το globalmemory να αποτελεί global μεταβλητή.

Epochs	**Χρόνος (seconds) (user+kernel)**	**Συνολική Μνήμη**	**Τελική απόδοση**
4	27.68	436.734.253.554	80,77%
6	33.34	648.624.476.263	81.05%
8	38.82	860.443.478.119	82.05%
10	44.96 seconds	1.072.452.223.655	81.4 %

Ζευγάρια ίδιων προϊόντων: 42535
Ζευγαρια διαφορετικών προϊόντων: 299394

Μέρος 3

Στο μέρος αυτό υλοποιήθηκε το κομμάτι του πολυνηματισμού τόσο για το training του μοντέλου όσο και για το testing αυτού. Επίσης υλοποιήθηκε η διαδικασία αλλαγής βαρών του μοντέλου από τα conflicts των κλικών που δημιουργήθηκαν από τις προβλέψεις του validation.

Για να δημιουργήσουμε Jobs για νήματα χρησιμοποιούμε την συνάρτηση CreateJobs, που δημιουργεί ανάλογα με την περίσταση διαφορετικά είδη jobs (training ή testing). Τα jobs για το training δημιουργούνται με βάση τα batches. Έπειτα κάθε νήμα εκτελεί συγκεκριμένο αριθμό από jobs.

Για το πολυνηματισμό χρησιμοποιούμε δύο συναρτήσεις, την Reader και την Writer. Η Writer είναι η συνάρτηση που χρησιμοποιούν τα threads για να κάνουν training ή testing το μοντέλο. Το τι από τα δύο θα κάνουν εξαρτάται από τον τύπο του Job που παίρνουν από την ουρά με jobs που δημιουργούμε. Η Reader λειτουργεί ως συντονιστής για τα νήματα. Αρχικά δημιουργεί jobs για το training και μετά μαζεύει για κάθε batch τις τιμές βαρών που υπολόγισαν οι Writers. Επίσης δημιουργεί jobs για την αλλαγή βαρών που γίνονται από τα conflicts.

Διαχείριση Conflicts:

Ελέγχουμε τις κλίκες που δημιουργήθηκαν από τις προβλέψεις του validation για conflicts. Αν βρούμε conflict, δηλαδή στοιχείο που υπάρχει στην κλίκα υπάρχει και στην κλίκα των αρνητικών συσχετίσεων, εργαζόμαστε ως εξής:

I. Υπολογίζουμε για κάθε συνδυασμό δύο στοιχείων την P[f(x)] δηλαδή τη πιθανότητα τα στοιχεία να είναι ίδια.
II. Υπολογίζουμε το μέσο όρο των παραπάνω τιμών
III. Αν ο μέσος όρος είναι μεγαλύτερος του 0.5 αποφασίζουμε ότι η κλίκα δεν έχει conflict οπότε για όλες τις τιμές του P[f(x)] που ήταν μικρότερες του 0.5 αλλάζουμε τα βάρη με τιμη y=1.
IV. Αν ο μέσος όρος είναι μικρότερος του 0.5 αποφασίζουμε ότι η κλίκα έχει conflict. Τότε για κάθε στοιχείο υπολογίζουμε την P[f(x)] με όλα τα άλλα στοιχεία. Αν η μέση P[f(x)] για κάποιο στοιχείο είναι μικρότερη του 0.5 αποφασίζουμε ότι το στοιχείο δεν ανήκει στη κλίκα, και αλλάζουμε τα βάρη για όλες τις δυάδες με αυτό το στοιχείο δίνοντας τιμή y=0.

Για περαιτέρω κατανόηση δείτε τη συνάρτηση fixConflict στο αρχείο logistic_regression.c.

Ο παραπάνω αλγόριθμος αποδείχτηκε πειραματικά ότι ήταν ο αποδοτικότερος για την μείωση των conflict.

Πειράματα

Epochs αρχικού training: 1
Batches: 512
Threads: 8
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
783 477 292 212 181 161 143 123 110 105 97 87 85 80 77 74 73 73 72 71 67 66 63 60 60 58 55 55 55 55 54 52 50 49 48 47 46 45 45 43 43 43 42 41 40 38 37 36 34 33 33 32 32 32 32 32 29 29 29 29 29 29 29 29 29 29 29 28 28 26 25 24 24 26 24 25 27 25 26 27 26 27 26 26 29 26 27 26 27 26 27 26 27 26 27 26 27 26 27
Χρόνος user+system: 88 seconds
Χρόνος real: 125 seconds
Accuracy στο τέλος: 87.45%

Epochs αρχικού training: 1
Batches: 512
Threads: 16
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
783 477 292 212 181 161 143 123 110 105 97 87 85 80 77 74 73 73 72 71 67 66 63 60 60 58 55 55 55 55 54 52 50 49 48 47 46 45 45 43 43 43 42 41 40 38 37 36 34 33 33 32 32 32 32 32 29 29 29 29 29 29 29 29 29 29 29 28 28 26 25 24 24 26 24 25 27 25 26 27 26 27 26 26 29 26 27 26 27 26 27 26 27 26 27 26 27 26 27
Χρόνος user+system: 123 seconds
Χρόνος real: 124 seconds
Accuracy στο τέλος: 87.451868 %

Epochs αρχικού training: 1
Batches: 1024
Threads: 8
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
783 476 294 213 181 161 143 123 111 105 95 87 85 80 77 74 73 73 72 71 67 66 63 60 60 58 55 55 55 55 54 52 50 48 48 47 46 45 45 43 43 42 42 41 38 38 38 36 34 33 32 32 32 32 32 32 31 31 29 29 29 29 29 29 29 29 29 28 28 28 26 24 25 27 24 26 28 26 27 26 26 28 26 26 29 26 28 26 27 26 27 26 27 26 27 26 27 26 27 26 27 26 27
Χρόνος user +system: 68 seconds
Χρόνος real: 90 seconds
Accuracy στο τέλος: 87.44 %

Epochs αρχικού training: 1
Batches: 1024
Threads: 16
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
783 476 294 213 180 161 143 124 111 105 96 87 85 80 77 74 73 73 72 71 67 66 62 60 60 58 55 55 55 55 54 52 50 48 48 47 47 45 45 43 43 42 42 41 38 38 37 36 34 33 32 32 32 32 32 32 31 29 29 29 29 29 29 29 29 29 28 28 28 25 26 25 26 25 26 28 26 27 26 26 28 26 27 26 26 28 26 28 26 27 26 27 26 27 26 27 26 27 27 26 27 26
Χρόνος user+system: 99 seconds
Χρόνος real: 68 seconds
Accuracy στο τέλος: 87.4489 %

Epochs αρχικού training: 5
Batches: 512
Threads: 8
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w
786 252 152 135 124 111 107 101 96 86 80 75 70 67 65 62 60 60 58 55 55 55 54 53 53 52 52 52 50 49 49 49 49 48 47 44 42 41 40 38 37 37 37 37 37 36 36 36 36 36 36 36 36 36 35
Χρόνος user+system: 92 seconds
Χρόνος real: 88 seconds
Accuracy στο τέλος: 87.522668 %

Epochs αρχικού training: 5
Batches: 512
Threads: 16
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w
786 252 152 135 124 111 107 101 96 86 80 75 70 67 65 62 60 60 58 55 55 55 54 53 53 52 52 52 50 49 49 49 49 48 47 44 42 41 40 38 37 37 37 37 37 36 36 36 36 36 36 36 36 36 35
Χρόνος user+system: 102 seconds
Χρόνος real: 64 seconds
Accuracy στο τέλος: 87.522668 %

Epochs αρχικού training: 5
Batches: 1024
Threads: 8
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
785 248 150 134 123 112 107 101 96 86 81 75 70 67 64 62 60 60 58 56 54 54 54 53 53 52 52 52 50 49 49 49 49 48 47 44 42 41 40 39 37 37 37 37 37 36 36 36 36 36 36 36 36 36 35 35
Χρόνος user+system: 59 seconds
Χρόνος real: 91 seconds
Accuracy στο τέλος: 87.422668%

Epochs αρχικού training: 5
Batches: 1024
Threads: 16
Αρχείο εισόδου: μεγάλο
Conflicts που εμφανίζονται κάθε φορά αφού αφού διορθωθούν προηγούμενα conflicts και τροποποιηθούν τα βάρη w:
785 250 151 134 124 111 107 101 96 86 81 75 70 67 64 62 60 60 58 56 54 54 53 53 53 52 52 52 50 49 49 49 49 48 47 44 43 42 40 40 37 37 37 37 37 36 36 36 36 36 36 36 36 36 35 35
Χρόνος user+system: 69 seconds
Χρόνος real: 52 seconds
Accuracy στο τέλος: 87.422668 %

Συμπεράσματα

1. Οταν αυξάνουμε τα epochs του αρχικού training παρατηρούμε οτι η σύγκλιση επιτυγχάνεται πιο γρήγορα.

2. Οταν αυξάνουμε το batch size παρατηρούμε πως ο χρόνος (user + sys) μειώνεται σημαντικά. Πχ όταν για epoch=5, threads=8 το batch size το αυξήσαμε από 512 σε 1024 ο χρόνος μειώθηκε από 92 seconds σε 59 seconds.

3. Παρατηρούμε πως στο 3ο μέρος της εργασίας, η λογική των mini batches και των conflicts, που γίνονται συνεχώς resolve μέχρι να συγκλίνουν, βελτιώνει το μοντέλο, κάτι το οποίο φαίνεται και από το accuracy στο test data, που αυξάνεται περίπου κατά 7%.

Σημείωση

Επειδή η συνθήκη σύγκλισης είναι αρκετά αυστηρή μπορείτε στην γραμμή 214 στο thread.c να μειώσετε το 15, το οποίο αντιπροσωπεύει τον αριθμό των συνεχόμενων παραπλήσιων τιμών conflicts που πρέπει να βρούμε για να συγκλίνει το μοντέλο. Εναλλακτικά, μπορείτε στην γραμμή 220 να αυξήσετε το 2 που αντιπροσωπεύει το όριο της διαφοράς δύο conflicts ώστε να θεωρούνται παραπλήσια.

Εκτέλεση Προγράμματος

Υποστήριξη εντολών

1. make -> μεταγλωτίζει το πρόγραμμα και παράγει τα αντίστοιχα αντικειμενικά αρχεία και εκτελέσιμα, μαζί με το unit_test

2. make clean -> διαγράφει τα αντικειμενικά αρχεία, μαζί με τα εκτελέσιμα και τα output αρχεία που δημιουργήθηκαν

Μέρος 1

./find_commons -d “μονοπάτι για τον φάκελο με τα αρχεία” -f “αρχείο με συσχετίσεις” -h “μέγεθος του hash table”

Μέρος 2

./find_commons -d “μονοπάτι για τον φάκελο που αποτελείται απο υποφακέλους με τα json files” -f “αρχείο με συσχετίσεις” -h “μέγεθος του hash table”

Μέρος 3

./find_commons -d “μονοπάτι για τον φάκελο με τα αρχεία” -f “αρχείο με συσχετίσεις” -h “μέγεθος του hash table” -sw stopwords.txt

Αρχεία

1. **RBtree.c, RBtree.h**: Υλοποίηση Red black δέντρων
2. **HashTable.c, HashTable.h**: Συναρτήσεις για hashtable
3. **list.c, list.h**: Λίστα με matching products
4. **jsonParser.c, jsonParser.h**: Συναρτήσεις για να διαχειρίζονται τα json αρχεία, μαζί με λίστες και έναν json parser
5. **helpFunctions.c, helpFunctions.h**: Συναρτήσεις για διάβασμα των παραμέτρων και για την αρχικοποίηση των δομών
6. **unit_test.c**: Συναρτήσεις για unit testing
7. **main.c**: Η main
8. **logistic_regression.c, logistic_regression.h**: Υλοποίηση του logistic regression μοντέλου
9. **dataList.c, dataList.h**: Λίστες για αποθήκευση των train και test δεδομένων
10. **thread.c, thread.h**: Υλοποίηση του πολυνηματισμού
11. **stopwords.txt**: Αρχείο με stopwords
12. **Makefile**: Δημιουργεί τα αντικειμενικά αρχεία και τα εκτελέσιμα, μαζί με το unit_test