Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Ιωάννης Ιάκωβος Λάμης 1115201600087

Κωνσταντίνος Δούμπας 1115201600044

Παραδοτέο 1

Στο πρώτο παραδοτέο καλούμασταν να προσομοιώσουμε την δομή ενός γράφου και να εξάγουμε τις πιθανές κλίκες οι οποίες προέκυπταν από τις θετικές συσχετίσεις μεταξύ των κόμβων του γράφου. Οι πληροφορίες του γράφου όσον αφορά τις συσχετίσεις των κόμβων δίνονταν μέσα από αρχείο csv. Για την αποθήκευση των πληροφοριών από το csv χρησιμοποιήθηκαν ένα struct (Pair) το οποίο αποτελούνταν από τα δυο id και ένα αριθμό που αναπαριστά την σχέση των δυο id και ένα vector (CustomVector) στο οποίο αποθηκεύονταν τα εκάστοτε Pair. Εδώ να αναφερθεί ότι αντί για vector έγιναν πειραματισμοί και με λίστα (αρχικός σχεδιασμός) χωρίς ωστόσο να έχει σημαντική διαφορά στην αποδοτικότητα (μνήμη, χρόνος). Όσων αφορά την προσομοίωση των κλικών, εδώ χρησιμοποιήθηκε HashMap με key typeof string και value typeof HashMap. Ο συγκεκριμένος σχεδιασμός έγινε τόσο λόγω της γρήγορης αναζήτησης σε HashMap O(1) όσο και του πλεονεκτήματος του στην αποφυγή duplicate values (το εμφολευμένο HashMap είχε για key και value string). Όσον αφορά την εξαγωγή αυτών των κλικών και την αποθήκευση των unique pairs σε csv, χρησιμοποιήθηκε η δομή της Λίστας. Το εύρος εκτέλεσης του προγράμματος κυμαίνεται στα 2.5 – 4sec με τα 4sec να προκύπτουν από την χρήση docker container και το overhead που δημιουργούνταν.

Παραδοτέο 2

Στο δεύτερο παραδοτέο εκτός από την προσθήκη των αρνητικών συσχετίσεων στην κλίκα υπήρχαν και τα ακόλουθα ζητούμενα:

- Διάβασμα και εξαγωγή πληροφοριών από φάκελο με κάμερες.
- Υλοποίηση tokenization στις παραπάνω πληροφορίες.
- Υλοποίηση Tf-idf Vectorization με input τα παραπάνω tokens
- Υλοποίηση Bow Vectorization με παρόμοια input
- Υλοποίηση Logistic Regression.

Για το διάβασμα και την εξαγωγή πληροφοριών από το φάκελο με τις κάμερες έγινε χρήση της ήδη υπάρχουσας δομής του HashMap.

Το ίδιο έγινε και στο tokenization στο οποίο χρησιμοποιήθηκε και μια παρόμοια δομή του vector που είχαμε υλοποιήσει στο πρώτο παραδοτέο που όμως ήταν πιο αποδοτική (FastVector).

Όσον αφορά τους Tf-Idf και Bow vectorizers από δομές χρησιμοποιήθηκαν τόσο HashMap όσο και FastVector. Και για τους δυο υπάρχουν οι συναρτήσεις fit και

transform. Το fit έπαιρνε σαν είσοδο τα παραχθέντα στοιχεία από το tokenization και δημιουργούσε στο αντίστοιχο vocabulary (στον Tf-Idf εκεί υπολογιζόταν και το idf).

Ενώ κατά το transform υπολογιζόταν το frequency κάθε λέξης μέσα στην πρόταση και παραγόταν ένας πίνακας δυο διαστάσεων typeof float και int για τον tfidf και bow αντίστοιχα.

Ακόμα για το μοντέλο, υλοποιήθηκε πλήθος συναρτήσεων:

- Fit στο οποίο δίνεται το train dataset και κατά το οποίο γίνεται η εκπαίδευση του μοντέλου
- Predict στο οποίο δίνεται το dataset για το οποίο θέλουμε να κάνει προβλέψεις το μοντέλο
- Update weights η οποία ανανέωνε τα βάρη του μοντέλου με νέες τιμές.
- Cost function η οποία υπολόγιζε το loss του μοντέλου σε τυχόν predict

Να σημειωθεί πως το dataset του μοντέλου δεν είναι της μορφής array από bow ή tfidf vectors καθώς αυτή η σχεδίαση κατανάλωνε υπερβολική μνήμη.

Η υλοποίηση που χρησιμοποιήθηκε είναι η χρήση ενός FastVector στο οποίο είναι αποθηκευμένες οι γραμμές που προκύπταν από τις κλίκες, η χρήση ενός array δυο διαστάσεων στο οποίο είναι αποθηκευμένες οι πληροφορίες για κάθε αρχείο με την μορφή tfidf και ένα hashmap με key το id του αρχείου και value τη θέση των πληροφοριών του αρχείου στο δυσδιάστατο πίνακα.

Γενικά η διαδικασία του προγράμματος που ακολουθήθηκε είναι η εξής:

1. Διάβασμα w dataset και δημιουργία τελικού dataset.
2. Διάβασμα φακέλου με κάμερες, tokenization, vectorization
3. Εκπαίδευση αξιολόγηση μοντέλου.

Ακόμα, παρατηρήσεις που προέκυψαν από την εκπαίδευση και αξιολόγηση του μοντέλου είναι, πως ενώ το μοντέλο έφτανε accuracy 87% όλες οι άλλες τιμές (f1, recall, precision) ήταν αρκετά χαμηλές πράγμα που δείχνει πως το μοντέλο έκλινε και προς την μια τιμή του target πιο συγκεκριμένα την τιμή 0. Αυτό συμβαίνει κυρίως λόγω του ότι το dataset ήταν unbalanced και αποτελούνταν κυρίως από γραμμές που σαν target είχαν το 0 (περίπου το 70% αν όχι παραπάνω). Σε μια προσπάθεια να γίνει το dataset πιο balanced (αφαιρώντας κάποιες γραμμές με target 0) το μοντέλο έδειχνε να εκπαιδεύεται καλυτέρα και να έχει καλυτέρα αποτελέσματα.

Τέλος, ενδεικτικός χρόνος εκτέλεσης του προγράμματος είναι τα 2 λεπτά (το μοντέλο τρέχει με 5 epochs).

Παραδοτέο 3

Στο τελευταίο παραδοτέο του μαθήματος είχε τις εξής απαιτήσεις:

- Επιλογή των 1000 καλύτερων μέσων Tf-Idf

- Εισαγωγή multithreading στο μοντέλο (τόσο στο fit όσο και στο predict)
- Υλοποίηση mini-batch gradient descent στο μοντέλο
- Υλοποίηση επαναληπτικής μάθησης του μοντέλου

Mini Batch gradient descent:

Πριν την εκπαίδευση του μοντέλου, το dataset χωρίζεται σε batches μεγέθους 1024 και υστέρα το μοντέλο εκπαιδεύεται με βάση αυτά. Η ενημέρωση των συντελεστών του μοντέλου γίνεται μετρά από το κάθε batch.

Multithreading:

Οι δυνατότητες multithreading έχουν γίνει abstract στην κλάση JobScheduler που χρησιμοποιεί το singleton pattern ώστε να είναι βέβαιο ότι δεν θα υπάρχουν πολλαπλά instances στο πρόγραμμα. Με το instantiation της κλάσης δημιουργούνται τόσα threads όσοι πυρήνες έχει το μηχάνημα σε ένα thread pool. Μέσω της μεθόδου addJob προστίθεται στο queue το job και εκτελείται μόλις κάποιο thread είναι διαθέσιμο. Το job queue είναι διπλάσιο από τα threads ώστε να μην σπαταλάται χρόνος όπου τα threads περιμένουν κάποιο νέο job και για να μην υπάρχουν θέματα μνήμης. Αν γεμίσει το pool, τότε η addJob μπλοκάρει μέχρι να αδειάσει κάποια θέση. Για την επίτευξη της επικοινωνίας και συγχρονισμού των threads χρησιμοποιούνται semaphores και mutexes. Τέλος, η waitAllJobs μπλοκάρει μέχρι να έχουν τελειώσει τα threads όλα τα jobs και το pool να έχει αδειάσει.

Επίσης, τα jobs που τρέχουν γίνονται abstract από το struct Job το οποίο έχει το lambda function που θέλουμε να τρέξει και την μέθοδο run η οποία το εκτελεί. Το lambda είναι απλώς syntactic sugar και τελικά είναι απλώς ένα struct αφού γίνει compile με τα captured variables να είναι μέλη του. Τα lambdas είναι compiler feature της C++11.

Επαναληπτική Μάθηση:

Αρχικά να επισημανθεί ότι λόγω του χρόνου που έπαιρνε το πρόγραμμα για να ελέγξει όλα τα πιθανά ζευγάρια αρχείων, αποφασίστηκε για κάθε τιμή threshold να επιλέγεται τυχαία ένα διάστημα το οποίο θα περιλαμβάνει τα αρχεία που θα ελεγχθούν (το διάστημα είναι μεγέθους 2000 αρχείων). Έτσι καταφέραμε τα εξής:

- το dataset με το οποίο το μοντέλο εκπαιδεύεται να αυξάνεται, με αποτέλεσμα το μοντέλο να γίνεται καλύτερο
- να μην απαιτείται ο ίδιος χρόνος προκειμένου να τελειώσει η μάθηση για τις εκάστοτε τιμές του threshold
- και τέλος να μπορεί η επαναληπτική μάθηση να τρέξει για διαφορετικές τιμές του threshold

Ακόμα επειδή θέλαμε να αποφύγουμε την χρήση υπερβολικής μνήμης για την αποθήκευση των ζευγαριών που περνάνε επιτυχώς από την συνθήκη του threshold επιλέξαμε να τα αποθηκεύουμε σε ένα temp αρχείο, το οποίο υστέρα να χρησιμοποιηθεί για να κάνουμε resolve το transitivity.

Η επιλογή μας όσων αφορά το transitivity είναι να εμπιστευόμαστε το μοντέλο. Αυτό σημαίνει πως αν παραδείγματος χάρη το μοντέλο προέβλεπε ότι a, b είναι διαφορετικά μεταξύ τους δηλαδή 0, τότε το περνάγαμε στον γραφώ ως αρνητική συσχέτιση. Τυχόν conflict τα έλυνε ο γραφτός, δηλαδή απλά δεν αποθήκευε το ζευγάρι a,b με τιμή 0 αν ήξερε ήδη ότι ισχύει a,b έχουν τιμή 1.

Τέλος προκειμένου να μπορέσουμε να κάνουμε αρκετές μετρήσεις, αποφασίσαμε το threshold να ξεκινά με τιμή 0. 01 και το βήμα το οποίο αύξανε το threshold να ξεκινά από 0.1 αλλά μετρά από κάθε αύξηση του threshold να διπλασιάζεται και το ίδιο έχοντας έτσι συνολικά 3 επαναλήψεις με το threshold.

Η maximum τιμή της ram ήταν 6 gb οπού αυτό γινόταν στη εύρεση των χίλιων καλυτέρων Tf-Idf. Μέσος χρόνος του προγράμματος άλλαζε ανάλογα με το μηχάνημα στο οποίο έτρεχε, ωστόσο κυμαίνεται στα 8 λεπτά.

Επισκόπηση δομών που χρησιμοποιήθηκαν:

1. List (κανονική δομή λίστας)
2. CustomVector (προσομοίωση δομής vector)
3. FastVector (παρόμοιο με το CustomVector πιο αποδοτικό από θέμα χρόνου και μνήμης και από το πλήθος παρεχόμενων features)
4. HashMap (δομή hash table)
5. Set (δομή hash table με παραπάνω features)
6. Clique (Hash table με std::string για key και pointer σε Set για value)

Οι δομές που χρησιμοποιήθηκαν εν τελεί πιο πολύ είναι:

1. FastVector
2. Set
3. HashMap

Επισκόπηση Vectorizer που υλοποιήθηκαν:

1. Tf-idf Vectorizer με την δυνατότητα να κρατήσεις τα 1000 καλυτέρα από το συνολικό vocabulary
2. Bow Vectorizer

Πληροφορίες μηχανήματων που χρησιμοποιήθηκαν

1. Installed OS: Windows 10 Education (used WSL 1)
   Processor: Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz 3.40 GHz
   Installed Ram: 32.0 GB DDR3
   System Type: 64-bit operating system, x86-64 processor architecture

2. Installed OS: Windows 10 Pro (used WSL 1)
   Processor: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
   Installed Ram: 16.0 GB
   System Type: 64-bit operating system, x86-64 processor architecture
3. Installed OS: MacOS 11
   Processor: i7-8750H 6 cores 12 threads @ 2.2GHz base clock
   Installed Ram: 16GB DDR4
   System type: 64-bit operating system, x86-64 processor architecture
4. Installed OS: Ubuntu Server 20.04
   Processor: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC
   Installed Ram: 8GB LPDDR4
   System type: 64-bit operating system, arm64 processor architecture