ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ
2020-2021

ΤΟ PROJECT ΥΛΟΠΟΙΗΘΗΚΕ ΑΠΟ ΤΟΥΣ ΕΞΗΣ:

ΑΘΑΝΑΣΙΟΣ ΤΣΟΦΥΛΑΣ: 1115201600186
ΔΙΟΝΥΣΙΟΣ-ΑΓΓΕΛΟΣ ΑΘΑΝΑΣΟΠΟΥΛΟΣ: 1115201600001

ΧΡΗΣΤΟΣ ΣΚΑΝΔΑΛΗΣ: 1115201600154

ΜΕΡΟΣ 1ο:

Στο πρώτο μέρος του Project ,έπρεπε να υλοποιηθούν οι θετικές κλίκες μεταξύ των δεδομένων
Αυτό πραγματοποιήθηκε με τον εξής τρόπο:

Αρχικά κάθε json αρχείο αποθηκεύτηκε ως μία δομή vertex ,οπού προσδιορίζει την κορυφή μίας κλίκας και έπειτα αποθηκεύτηκε σε μία δομή hash table.

Το κάθε vertex έχει έναν pointer σε λίστα η οποία περιέχει όλες τις κορυφές της κλίκας που ανήκει το συγκεκριμένο vertex

ΜΕΡΟΣ 2ο:

Στο δεύτερο μέρος του Project , πέρα από τις θετικές , έπρεπε να υλοποιηθούν και οι αρνητικές συσχετίσεις
Αυτό πραγματοποιήθηκε με τον εξής τρόπο:

Η κάθε λίστα που παριστάνει την κλίκα, με όλα τα vertex που ανήκουν σε αυτήν, έχει έναν pointer σε μία άλλη λίστα η οποία περιέχει όλες εκείνες τις κλίκες-λίστες οι οποίες είναι διαφορετικές με την συγκεκριμένη κλίκα-λίστα.

Στη συνέχεια δημιουργείται ένα μοντέλο μηχανικής μάθησης με το 60% των δεδομένων και με τον εξής τρόπο:

Αρχικά δημιουργείται ένα λεξικό όπου περιέχει όλες τις λέξεις που βρίσκονται στα json files.
(Μία φορά την κάθε λέξη)
Το λεξικό αυτό είναι μία δομή vector η οποία υλοποιήθηκε απο εμάς και παράλληλα χρησιμοποιήθηκε μια δομή φίλτρου (Bloom Filter) για αποτελεσματικότερη και ταχύτερη δημιουργία του.

Στη συνέχεια μετατρέψαμε κάθε json file σε διανύσματα σταθερού μήκους μετρώντας πόσες φορές εμφανίζεται η

κάθε λέξη στο κείμενο(Bag-Of-Words)
Πρίν γίνει αυτό όμως, διπλότυπες λέξεις με πεζά και κεφαλαία έγιναν με πεζά και τα σημεία στίξης και λέξεις που επαναλαμβανόντουσαν σε όλα τα json , απορρίφθηκαν.
Αυτή η δομή είναι sparse vector.

Έπειτα χρησιμοποιήθηκε η τεχνική TF-IDF .

(Text Frequency - Inverse Document Frequency)

Δημιουργήθηκε και για αυτή την μέθοδο ενα sparse vector για κάθε Json, όπου αποθηκεύτηκαν οι τιμές tf-idf για κάθε λεξη και δόθηκε μια συνολική τιμή για το json .

Τέλος , έχοντας πλέον τα απαραίτητα διανύσματα προχωρήσαμε στην λογιστική παλινδρόμηση

(Logistic Regression)

Παίρνοντας όλες τις θετικές-αρνητικές συσχετίσεις (από το 60% του dataset W) ανά 2 Json-vertex, ανάλογα με το σφάλμα σε κάθε επανάληψη(stochastic gradient descent) , άλλαζαν και τα βάρη , κρατόντας στο τέλος τα βάρη με το μικρότερο σφάλμα.

Χρησιμοποιόντας αυτά τα βάρη δοκιμάσαμε την ακρίβεια του μοντέλου μας ,με το υπόλοιπο 40% του dataset W και καταλήξαμε ότι η μεγαλύτερη δυνατή ακρίβεια επιτυγχάνεται με τις εξής τιμές:

h=0.01 (learning rate)

threshold=0.5 (τιμή απόφασης για την εκτίμηση θετικής-αρνητικής

συσχέτισης (prediction))

ΜΕΡΟΣ 3ο:

Στο τελευταίο μέρος του project σκοπός ήταν η βελτίωση χρόνου και η επίτευξη μεγαλύτερης ακρίβειας.

Αρχίκα προστέθηκαν νήματα-threads .

Δημιουργήθηκε μία δομή Εργασία(Job) η οποία είναι μια ρουτίνα κώδικα όπου θέλουμε να εκτελεστεί από κάποιο νήμα, παράλληλα με κάποια άλλη.

Για την διαχείρηση και ανάθεση των jobs και threads υλοποιήθηκε μία δομή προγραμματισμού ενεργειών (Job Scheduler).

Ο Job Scheduler αποτελείται από μία ουρά τύπου FIFO στην οποία εισάγουμε τα διάφορα Jobs, όποτε ο χρήστης κρίνει απαραίτητο.

Κατά τη δημιουργία του Scheduler, ο χρήστης εισάγει τον αριθμό των Threads, τα οποία και δημιουργούνται και ξεκινούν να τρέχουν σε άπειρη επανάληψη,

προσπαθώντας να αναλάβουν το αντίστοιχο Job μέσω της λειτουργίας

"pop".

Αν η pop μέθοδος επιστρέψει κάποιο Job, και όχι null, τότε το εκάστοτε thread αναλαμβάνει την εκτέλεσή του.

Με αυτό τον τρόπο γνωρίζουμε ότι τα Jobs θα τρέξουν με τη σείρα την οποία εισήχθησαν.

Σε περίπτωση που ο χρήστης επιθυμεί να σιγουρέψει ότι όλα τα jobs που έχει εισάγει στο scheduler έχουν τελειώσει, μπορεί να τρέξει τη μέθοδο

"wait_all_tasks_finish", η οποία θα επιστρέψει μόνο όταν έχουν τελειώσει όλα τα jobs.

Για να σταματήσουν τα threads πρέπει να τρέξει η "destroy_scheduler", στην οποία γίνονται και "detach", καθώς τα threads δεν τερματίζουν μόνα τους επειδή δε γνωρίζουν αν ο χρήστης θα θελήσει κάποια στιγμή στο μέλλον να εισάγει και κάποιο άλλο "job".

Όσον αφορά το μοντέλο μηχανικής μάθησης , αρχικά σημειώθηκε μείωση του διανύσματος του κάθε vertex , στις 1000 σημαντικότερες λέξεις με βάση την tf-idf τιμή τους.

Στη συνέχεια το μοντέλο τροποποιήθηκε ώστε να λειτουργεί με βάση το mini-batch training , δηλαδή την αλλαγή των βαρών μετά από την επεξεργασία μίας δέσμης ζευγαρίων vertex.Η επεξεργασία του κάθε

batch δηλώνεται ως Job και εκτελείται από τα threads παράλληλα με άλλα batches.

Οι μεταβλητές που ορίζουν τα βάρη προκειμένου να έχουν πρόσβαση ολα τα threads έχουν δηλωθεί ως extern και για να αποκτήσει πρόσβαση οποιοδήποτε thread σε κάποια από αυτές τις μεταβλητές ,κλειδώνει και έπειτα ξεκλειδώνει ένα mutex ώστε να μην ύπαρχει σύγχυση μεταξύ τους.

Στην συνέχεια το μοντέλο δοκιμάζεται(testing) με διαφορετικό τρόπο απ'οτι στο ΜΕΡΟΣ 2ο.
Πλέον χρησιμοποιείται μία επαναληπτική μέθοδος με τον εξής τρόπο:

Ορίζουμε ένα threshold=0.1

Ελέγχουμε τα ζευγάρια vertex από το υπόλοιπο 20% του dataset W, αν το prediction<threshold ή prediction>1-threshold και τα εντάσουμε στις δομές του 60% .

Έπειτα ξαναεκπαιδεύουμε το μοντέλο μας με το νέο training set.

Αυτό επαναλαμβάνετε για 5 φορες ,αυξάνοντας κάθε φορά το threshold κατα 0.1.

Τέλος με το τελευταίο 20% του dataset W κάνουμε το validation του μοντέλου και εκτυπώνουμε τις εκτιμήσεις του.

Συνθήκες έκτελεσης:

Χρόνος: 7,55 λεπτα.

Cpu: 4 cores , 3.2GHz

RAM: 8GB

Σημειώσεις.

Δε χρησιμοποιήθηκε καμία έτοιμη δομή
Όλες οι δομές , όπως λίστες-ουρες-vector υλοποιήθηκαν από εμάς.