**ΜΥΕ047: Αλγόριθμοι για δεδομένα ευρείας κλίμακας**

1ο Σετ Ασκήσεων: Ανίχνευση Κοντινότερων Γειτόνων σε Συλλογή Εγγράφων

Στοιχεία ομάδας:

Αχιλλέας Παππάς, 4774, cs04774@uoi.gr

Χαράλαμπος Θεοδωρίδης, 4674, cs04674@uoi.gr

Περιεχόμενα:

1ο Μέρος: Περιγραφή των μεθόδων που υλοποιήθηκαν

* Ανάλυση μεθόδων υπολογισμού
* Ανάλυση main() μεθόδου και δοκιμές εκτέλεσης

2ο Μέρος: Πειραματική αξιολόγηση προγράμματος

* Ανάλυση δοκιμαστικής μεθόδου
* Ανάλυση δεδομένων που προκύπτουν

**1ο Μέρος: Περιγραφή των μεθόδων που υλοποιήθηκαν**

Ανάλυση μεθόδων υπολογισμού

Βάσει των οδηγιών της εκφώνησης, υλοποιήσαμε όλες τις μεθόδους δοκιμάζοντας σταδιακά την σωστή λειτουργία τους. Ακολουθούν όλες με τη σειρά που παρουσιάζονται στην εκφώνηση ενώ σχολιάζεται η λειτουργία τους, πιθανόν παραδοχές που έγιναν και το API που προσφέρει η κάθε μέθοδος για την κλήση της.

1) Ανάγνωση και επεξεργασία δεδομένων εισόδου

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΗ μέθοδος που υλοποιήθηκε ήταν η MyReadDataRoutine(filename, numDocuments);

Η μέθοδος αυτή ανοίγει το αρχείο με την χρήση των μεθόδων της python και διαβάζει τις δυο πρώτες γραμμές, δηλαδή το πλήθος των αρχείων και το πλήθος των λέξεων. Αν το numDocuments ξεπερνάει τον μέγιστο αριθμό αρχείων που διαβάσαμε, το πρόγραμμα επιστρέφει χωρίς αποτέλεσμα. Έπειτα αποθηκεύουμε το πλήθος των λέξεων στην σφαιρική μεταβλητή dictSize ώστε να την χρησιμοποιήσει όποια μέθοδος χρειαστεί. Tέλος, αφού προσπεράσουμε την τρίτη γραμμή του αρχείου, διαβάζουμε σταδιακά τον αριθμό εγγράφων που καθορίζει το numDocuments, ενώ αποθηκεύουμε κάθε λέξη σε ένα σετ. Στο τέλος κάθε επανάληψης, το σετ αναπαριστά ένα από τα numDocuments αρχεία, οπότε το μετατρέπουμε σε frozenList για να είναι iteratable και το αποθηκεύουμε στην θέση i του πίνακα output. Ο πίνακας αυτός περιέχει κάθε αρχείο που διάβασε από την είσοδο με index i+1 στην θέση i.

2) Συναρτήσεις για Jaccard ομοιότητα

Κατά την καθοδήγηση της εκφώνησης, αναπτύξαμε τις μεθόδους:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαMyJacSimWithSets(doc1,doc2);

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματακαι MyJacSimWithOrderedLists(doc1,doc2);

οι οποίες δέχονται ως ορίσματα δυο iterable σύνολα από λέξεις (συγκεκριμένα τα frozenset που δημιουργήσαμε κατά την ανάγνωση του εγγράφου) και επιστρέφουν την τομή των δύο συνόλων προς την ένωσή τους (δηλαδή την ομοιότητα κατά jaccard).

Ο τρόπος υπολογισμού είναι διαφορετικός σε κάθε μέθοδο: Η πρώτη χρησιμοποιεί έναν αφελή αλγόριθμο ωμής βίας με τετραγωνική πολυπλοκότητα ενώ η δεύτερη διατρέχει διαδοχικά τα σύνολα και συγκρίνει παράλληλα τα στοιχεία. Συγκεκριμένα αφού κάνει sort τα σετ σε γραμμικό χρόνο, διατηρεί ένα counter για κάθε σετ που αυξάνει σταδιακά αναλόγως με το μικρότερο στοιχείο που βρει. Έτσι καταφέρνει με ένα πέρασμα να βρει όλα τα κοινά στοιχεία χωρίς την πολυπλοκότητα της πρώτης μεθόδου.

Όπως είναι εμφανές και από την εκφώνηση, η δεύτερη μέθοδος είναι πολύ πιο γρήγορη και θα χρησιμοποιηθεί για όλες τις επόμενες συγκρίσεις κατά jaccard. Ο χρήστης έχει την δυνατότητα να παρατηρήσει αυτή την διαφορά μεταξύ των μεθόδων αν κατά την εκτέλεση του προγράμματος επιλέξει να συγκρίνει δυο έγγραφα μεταξύ τους (αφού δοκιμάσει και τις υπόλοιπες μεθόδους).

3) Δημιουργία υπογραφών με την MinHash

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΓια άλλη μια φορά, ακολουθήσαμε τις οδηγίες της εκφώνησης. Η μέθοδος που προκύπτει είναι η MyMinHash(docList, K); που φαίνεται παρακάτω:

Η μέθοδος παίρνει ως ορίσματα την λίστα από set που αναπαριστούν το έγγραφο που διαβάσαμε και τον αριθμό των τυχαίων μεταθέσεων Κ που συμπίπτει με το μήκος της κάθε υπογραφής. Επιστρέφει το μητρώο υπογραφών (Κ\*numOfDocuments) που δημιουργήθηκε.

Ας αναλύσουμε την λειτουργία της. Κατ’ αρχάς, χρησιμοποιούμε το random.seed(10) στην γραμμή 75 ώστε να έχουμε σταθερότητα στα αποτελέσματά μας κατά την διαδικασία των δοκιμών. Αυτό, φυσικά, είναι προαιρετικό και η συγκεκριμένη γραμμή έχει αναιρεθεί με comment στο τελικό παραδοτέο.

Με την χρήση μιας for δημιουργούμε μια λίστα από Κ λεξικά, στο καθένα από τα οποία υπάρχει μια τυχαία μετάθεση dictSize αριθμών, χρήσιμων για την δημιουργία υπογραφών. Μέσω του randomHashFunction που δημιουργούμε από την δοθείσα μέθοδο, αρχικοποιούμε ένα λεξικό με τυχαίες τιμές αλλά αριθμημένα κλειδιά. Έπειτα κάνουμε sort βάσει των τιμών και μετατρέπουμε τις τιμές σε αριθμημένες θέσεις, βάσει της θέσης που έχουν καταλήξει τα ζεύγη μετά το sort. Προσθέτουμε την τυχαία μετάθεση που προκύπτει στην λίστα h που περιέχει όλα τα hash functions.

Στη συνέχεια, αρχικοποιούμε την λίστα από λίστες sig που θα περιέχει όλες τις τελικές μας υπογραφές. Ο αλγόριθμος υποδεικνύει να αρχικοποιηθεί κάθε θέση στο άπειρο, οπότε εμείς την αρχικοποιούμε στην μέγιστη δυνατή τιμή, δηλαδή το dictSize.

Τέλος, βάσει του αλγορίθμου διατρέχουμε το μητρώο υπογραφών και για κάθε λέξη σε κάθε αρχείο κρατάμε την ελάχιστη τιμή που μπορούν να μας δώσουν οι Κ τυχαίες μεταθέσεις που δημιουργήσαμε παραπάνω. Ως κλειδί στην κάθε τυχαία μετάθεση, χρησιμοποιούμε την ίδια την λέξη, αφού ο αριθμός της αντιστοιχεί στην θέση της στο μητρώο χαρακτηριστικών διανυσμάτων. Η διαδικασία αυτή, φυσικά, αποδίδει ακριβώς τις ίδιες υπογραφές με την μέθοδο του μητρώου χαρακτηριστικών διανυσμάτων (που δεν διατρέξαμε, ούτε δημιουργήσαμε εδώ λόγω χώρου).

Φυσικά, τυπώνουμε στατιστικά χρόνου για αργότερες συγκρίσεις.

4) Σύγκριση υπογραφών της MinHash

Μένει να δημιουργήσουμε έναν τρόπο σύγκρισης των υπογραφών. Αυτή την δουλειά εκτελεί η MySigSim(docSig1, docSig2, numPermutations);

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Η απλή αυτή μέθοδος δέχεται ως όρισμα τις υπογραφές δυο εγγράφων που θέλουμε να συγκρίνουμε και επιστρέφει την Signature Similarity που ορίζεται από τον αριθμό των κατά θέση κοινών στοιχείων προς τον αριθμό των τυχαίων μεταθέσεων.

Για μεγάλο αριθμό μεταθέσεων, το signature similarity τείνει στο jaccard similarity. Αυτό παρατηρείται αργότερα και στα τεστ που κάνουμε, δείχνοντας ότι οι μέθοδοι έχουν υλοποιηθεί σωστά.

5) Υπολογισμός γειτόνων ανά έγγραφό με μεθόδους ωμής βίας

Η μεθοδολογία της ωμής βίας υπαινίσσεται τον έλεγχο όλων των εγγράφων μεταξύ τους για την εύρεση ομοιοτήτων. Αυτό συνεπάγεται τετραγωνική πολυπλοκότητα και μεγάλο αριθμό συγκρίσεων. Αυτές οι συγκρίσεις θα γίνουν με δύο τρόπους, ένας με την χρήση Jaccard similarity και ένας με την χρήση Signature similarity των υπογραφών από την Minhash.

Αναπτύξαμε, έτσι, δύο μεθόδους, την bruteForceJacNeighbors(doclist, numDocuments, numNeighbors); και την bruteForceSigNeighbors(sig, numDocuments, numNeighbors);. Η πρώτη παίρνει ως πρώτο όρισμα την λίστα των εγγράφων που διαβάσαμε, ενώ η δεύτερη παίρνει το μητρώο υπογραφών που θα έχουμε ήδη παράξει.

Στην ουσία θέλουμε να βρούμε τους numNeighbors κοντινότερους γείτονες για κάθε έγγραφο μεταξύ των numDocuments εγγράφων. Αρχικά, λοιπόν, για κάθε έγγραφο υπολογίζουμε όλες τις ομοιότητες με τα υπόλοιπα έγγραφα, κάνουμε sort τα αποτελέσματα βάσει της dist απόστασης τους (dist=1-similarity) και κρατάμε τους numNeighbors πρώτους γείτονες που προκύπτουν.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΗ ομοιότητα υπολογίζεται με δυο τρόπους. Ο πρώτος είναι με την χρήση jaccard similarity:

Όπως είναι φανερό, γεμίζουμε σταδιακά την λίστα simList με τις ομοιότητες κάθε εγγράφου με κάθε άλλο έγγραφο. Κάθε στοιχείο της στην θέση i είναι ένα λεξικό με κλειδί το αναγνωριστικό j ενός άλλου εγγράφου και τιμή το jaccard similarity μεταξύ των εγγράφων i+1 και j+1. Αφού γεμίσουμε το simList, κάνουμε sort βάσει distance κάθε λεξικό-στοιχείο του simList και κρατάμε μόνο τα numNeighbors πρώτα στοιχεία κάθε φορά.

Τέλος, υπολογίζουμε τον μέσο όρο ομοιοτήτων ώστε να έχουμε ένα μέτρο σύγκρισης με τις επόμενες μεθόδους. Φτιάχνουμε έναν πίνακα AvgSim όπου τοποθετούμε σε κάθε i θέση του τον μέσο όρο των ομοιοτήτων του εγγράφου i+1 με τους numNeighbors κοντινότερους γειτόνους του.

Υπολογίζουμε και τυπώνουμε το Avg που είναι ο μέσος όρος όλων των μέσων όρων που υπάρχουν στον πίνακα AvgSim.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΗ δεύτερη μέθοδος ήταν με χρήση signature similarity για τις ομοιότητες:

Ακολουθώντας την ίδια δομή με την προηγούμενη μέθοδο, προσθέτουμε τώρα στα λεξικά το Signature similarity αντί για το jaccard. Κάνουμε sort ως προς το distance και κρατάμε τους numNeighbors πρώτους γείτονες.

Με τον ίδιο τρόπο, υπολογίζουμε την λίστα των average και τελικά το avg που είναι ο ολικός μέσος όρος. Τα αποτελέσματα αυτών των μεθόδων θα χρησιμοποιηθούν ως μέτρο σύγκρισης με τις επόμενες μεθόδους.

Είναι φανερό ότι η δεύτερη υλοποίηση με τις υπογραφές θα είναι πιο γρήγορη στην εκτέλεση, αλλά όχι αρκετά ώστε να μπορούμε να την χρησιμοποιούμε με άνεση σε μεγάλες εισόδους. Στη συνέχεια θα μελετήσουμε την λειτουργία της LSH, τις αδυναμίες της και αργότερα τα προτερήματά της σε σχέση με την brute force.

5) Υπολογισμός γειτόνων ανά έγγραφο με την χρήση Locality Sensitive Hashing

Το Locality sensitive hashing (LSH) χρησιμοποιεί μια σειρά από hashing σε διακριτά μέρη των υπογραφών και τοποθετεί τα αποτελέσματα σε αντίστοιχους κάδους. Η λογική είναι ότι αν δυο έγγραφα έχουν ομοιότητες, τότε και οι υπογραφές τους θα έχουν παρόμοια τμήματα. Επομένως αν κάνουμε hash τμήματα των υπογραφών και αρκετά από τα αποτελέσματα καταλήξουν στον ίδιο κάδο (βάσει ενός threshold) τότε μπορούμε να λάβουμε τα αντίστοιχα έγγραφα υποψήφια για έλεγχο ομοιότητας. Έτσι, έγγραφα που δεν μοιάζουν αρκετά ώστε να έχει νόημα η σύγκριση μεταξύ τους μπορούμε να τα αγνοήσουμε.

Η τελική μέθοδος έχει αρκετά μέρη. Αυτό οφείλεται στο ότι πρέπει τα αποτελέσματα της LSH να μετατραπούν στην μορφή μέσων όρων ομοιότητας, έτσι ώστε να μπορούμε να συγκρίνουμε τα αποτελέσματα με τις brute force μεθόδους.

Ας ξεκινήσουμε μελετώντας τα ορίσματα:

LSH(docList, sig, rowsPerBands, numNeighbors, simMethod)

Το docList είναι η λίστα των εγγράφων όπως τα παρήγαγε η μέθοδος ανάγνωσης που δημιουργήσαμε. Η λίστα αυτή είναι απαραίτητη στο δεύτερο μέρος του αλγορίθμου, αν αποφασίσουμε να συγκρίνουμε τα αποτελέσματα της LSH με jaccard similarity. Το sig είναι χρήσιμο τόσο στην ίδια την LSH όσο και στον υπολογισμό signature similarity αργότερα.

Η μέθοδος σύγκρισης των ευρημάτων της LSH είναι είτε jaccard similarity (simMethod=1) ή signature similarity (simMethod=2). Η εύρεση του avgSim γίνεται με παρόμοιο τρόπο με τις brute force μεθόδους, απλώς αντί να δημιουργήσουμε δυο διαφορετικές LSH έχουμε την επιλογή της επιθυμητής μεθόδου στα ορίσματα.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΤο rowsPerBands είναι η μόνη παράμετρος που καθορίζει τα χαρακτηριστικά της LSH, δηλαδή το πλήθος των γραμμών ανά μπάντα, το πλήθος των μπαντών και το threshold που προκύπτει. Δημιουργήσαμε, επίσης, την μέθοδο LSHcount, που υπολογίζει το κοντινότερο rowsPerBands που πρέπει να εφαρμόσουμε για να έρθουμε στο επιθυμητό threshold.

Εδώ βλέπουμε ότι είναι προγραμματισμένη για να βρίσκει threshold κοντά στο 0.5, αλλά όπως θα δούμε αργότερα, το threshold δεν έχει βέλτιστη τιμή. Για ανόμοια έγγραφα ή για μεγάλο αριθμό γειτόνων, οφείλουμε να το μειώσουμε ώστε να αφήσουμε να περάσουν περισσότερα ζευγάρια από την LSH.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΤώρα, θα μελετήσουμε σταδιακά την δομή του LSH.

Αρχικά, δημιουργούμε το hash function που θα χρησιμοποιήσουμε, πάλι με σταθερό seed. Έπειτα, υπολογίζουμε τον αριθμό των μπαντών και αρχικοποιούμε την λίστα από λεξικά που θα είναι οι μπάντες (LSHdicts). Τώρα ξεκινάει ο κατακερματισμός των υπογραφών: για κάθε μπάντα στο LSHdicts διατρέχουμε όλες τις υπογραφές στο μητρώο υπογραφών sig **ανά τμήματα**. Αυτά τα τμήματα καθορίζονται από την τωρινή μπάντα και έχουν αρχή την αρχή της τωρινής μπάντας και τέλος την αρχή της επόμενης μπάντας, εκτός και αν είμαστε στην τελευταία μπάντα όπου το τέλος είναι το τέλος της υπογραφής (γραμμή 207). Το αποτέλεσμα το μετατρέπουμε σε πλειάδα ώστε να το κατακερματίσουμε με την random hash function που δημιουργήσαμε, αφού πρώτα το περάσουμε από την hash της python για να είναι σε επεξεργάσιμη μορφή.

Διατρέχοντας τις for, τα αποτελέσματα προστίθενται κάθε φορά στο LSHdicts οπότε στο τέλος αυτού του σταδίου έχουμε μια λίστα από μπάντες και κάθε μπάντα περιέχει μαζεμένα (sorted κατά κάδους πλέον, γραμμή 210) τα αποτελέσματα της LSH. Τα υποψήφια ζεύγη είναι στους ίδιους κάδους οπότε έπεται να διατρέξουμε τις μπάντες και να δημιουργήσουμε ζευγάρια για κάθε έγγραφα που κατέληξαν στον ίδιο κάδο και ξεπερνούν το threshold.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΑυτό γίνεται αμέσως μετά με την χρήση του παρακάτω loop:

Διατρέχοντας κάθε μπάντα ξεχωριστά, αποθηκεύουμε όσα στοιχεία βρίσκονται στους ίδιους κάδους σε μία λίστα cand. Όταν φτάσουμε σε νέο κάδο, δηλαδή κατά την ανάγνωση ο επόμενος κάδος είναι διαφορετικός από τον προηγούμενο, σχηματίζουμε ζεύγη από όλα τα στοιχεία που έχουν προστεθεί στην λίστα cand (else statement, γραμμές 226-237). Τέλος μηδενίζουμε τον πίνακα cand και ξεκινάμε να τον ξαναγεμίζουμε με στοιχεία των ίδιων κάδων. Τα ζευγάρια που παράγονται αποθηκεύονται με το μικρότερο στοιχείο πρώτο σε ένα σετ ώστε να αποφευχθούν τα διπλότυπα. Ομολογουμένως, η παρούσα διαδικασία είναι χρονοβόρα, ειδικά για μικρά threshold. Σε κάθε, όμως, περίπτωση, έχουμε βελτίωση σε σύγκριση με τις μεθόδους ωμής βίας.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΤο σετ με τα ζεύγη που παρήξαμε περιέχει μόνο ζεύγη με αξιοσημείωτα ποσοστά ομοιότητας. Επομένως μένει να διατρέξουμε αυτά τα ζεύγη και να δημιουργήσουμε μια λίστα με τις ομοιότητες των κοντινότερων γειτόνων. Αυτό ακριβώς κάνουμε στο επόμενο μέρος της μεθόδου:

Αρχικοποιούμε το simList και ξεκινάμε να το γεμίζουμε. Ανάλογα με την μέθοδο ομοιότητας που έχουμε επιλέξει από τα ορίσματα, υπολογίζουμε είτε το jaccard similarity είτε το signature similarity. Όταν γεμίσει η λίστα, έχουμε πλέον φτάσει στο ίδιο σημείο με την brute force μέθοδο (σε πολύ λιγότερο χρόνο) οπότε κάνουμε sort βάσει της απόστασης (dist=1-similarity οπότε απλώς κάνουμε sort κατά φθίνουσα σειρά) και κρατάμε μόνο τους πρώτους numNeighbors γείτονες.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΑν το threshold ήταν κατάλληλο, πρέπει πλέον να έχουμε το ακριβώς ίδιο similarity list με τις brute force μεθόδους. Το τελευταίο κομμάτι είναι ο υπολογισμός της μέσης ομοιότητας μεταξύ όλων των εγγράφων. Αυτό είναι και το τελευταίο μέρος της μεθόδου:

Επιστρέφουμε το simList σε περίπτωση που θέλει να το τυπώσει ο χρήστης.

Ανάλυση main() μεθόδου και δοκιμές εκτέλεσης

Το κομμάτι αυτό αφορά την Πειραματική αξιολόγηση της άσκησης. Δημιουργήσαμε την main() μέθοδο που αναλαμβάνει την επικοινωνία με τον χρήστη, τυπώνοντας τα κατάλληλα μηνύματα και ζητώντας τις κατάλληλες τιμές. Θα παρουσιάσουμε την διαδικασία εκτέλεσης του προγράμματος από την main(), παράλληλα μέσω του κώδικα και από τη μεριά του χρήστη με παραδείγματα εκτέλεσης.

Αρχικά, ζητείται από τον χρήστη να διαλέξει αρχείο εισόδου από τις δύο επιλογές. Τα αρχεία πρέπει να βρίσκονται στον ίδιο κατάλογο με το εκτελέσιμο, ενώ σε περίπτωση λανθασμένης επιλογής το πρόγραμμα διαλέγει αυτόματα η δεύτερη επιλογή ενώ ενημερώνεται ο χρήστης ότι έκανε λάθος. Έπειτα εκτελούμε την MyReadDataRoutine για το αρχείο και τον αριθμό των εγγράφων που επιλέχτηκαν.

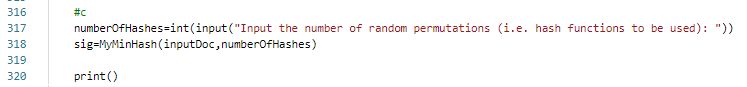
Εικόνα που περιέχει κείμενο

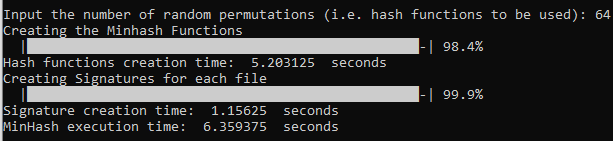
Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΈπειτα, επιλέγεται ο αριθμός των γειτόνων για τους οποίους θα υπολογιστούν οι ομοιότητες. Ο χρήστης περιορίζεται σε έναν μικρό αριθμό, μεταξύ 2 και 5 (default τιμή =2):

Ο χρήστης, μετά, πρέπει να επιλέξει τον αριθμό των τυχαίων μεταθέσεων που θα πραγματοποιηθούν (δηλαδή τον αριθμό των hash function και με αυτό το μήκος των υπογραφών).

Αφού έχουμε στην διάθεσή μας και το αρχείο εισόδου, υπολογίζουμε απευθείας τις υπογραφές με την MyMinHash. Για την καλύτερη εμπειρία του χρήστη, έχουν προστεθεί σε αρκετές από τις χρονοβόρες μεθόδους progress bars ώστε να παρατηρεί ζωντανά την πρόοδο. Ο κώδικας για αυτές βρίσκεται στην μέθοδο printProgressBar και δεν είναι δικός μας ([πηγή](https://stackoverflow.com/questions/3173320/text-progress-bar-in-terminal-with-block-characters)).

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΤέλος, πρέπει να επιλέξει τις μεθόδους που θα χρησιμοποιηθούν. Έχει δύο επιλογές για τις μεθόδους σύγκρισης και άλλες δύο για τις μεθόδους υπολογισμού γειτόνων, με ένα σύνολο 4ων συνδυασμών. Οι τιμές που επιλέγονται σε περίπτωση λάθους είναι 2 και 2.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Έχοντας τα κατάλληλα δεδομένα, προχωράμε στον υπολογισμό των ομοιοτήτων. Στην περίπτωση του brute force, τα πράγματα είναι απλά: ανάλογα με την μέθοδο ομοιότητας, τρέχουμε την αντίστοιχη μέθοδο με τα κατάλληλα ορίσματα:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

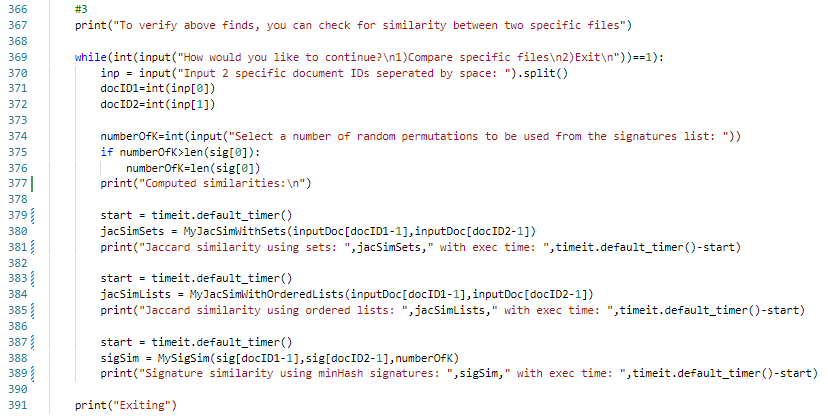
Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΣτην περίπτωση του LSH, πρέπει ο χρήστης να επιλέξει κατάλληλο rowsPerBands. Τρέχουμε την LSHcount, προτείνοντας στον χρήστη ένα threshold γύρω στο 0.5. Αυτό το threshold είναι κατάλληλο για το Enron αρχείο εισόδου, αλλά για το Nips ή για μεγαλύτερο αριθμό γειτόνων, ο χρήστης μπορεί να επιλέξει μικρότερο αριθμό rowsPerBands από τον προτεινόμενο για μικρότερο threshold.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Έχοντας, πλέον, υπολογίσει τα avg similarities, ο χρήστης μπορεί να διαλέξει να συγκρίνει την ομοιότητα μεταξύ οποιονδήποτε 2 εγγράφων για να επιβεβαιώσει τα ευρήματα. Η σύγκριση γίνεται με τις δυο jaccard μεθόδους και με την signature similarity ώστε να γίνει εμφανής η διαφορά στον χρόνο εκτέλεσης. Τα IDs των αρχείων που συγκρίνονται μειώνονται κατά 1 ώστε η σύγκριση να γίνεται μεταξύ IDs των πραγματικών αρχείων.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι η jaccard similarity με ordered lists είναι αρκετά πιο γρήγορη από την jaccard similarity με σετ. Επίσης παρατηρούμε ότι η signature similarity απέχει από την τιμή των jaccard similarity, αλλά αυτή η διαφορά μειώνεται για μεγαλύτερο αριθμό από hash function.

Μια απροσδόκητη μέτρηση είναι ο μεγαλύτερος χρόνος εκτέλεσης της signature similarity από την jaccard similarity with ordered lists. Αυτή η διαφορά εξηγείται στον μεγάλο αριθμό των hash function. 128 επαναλήψεις είναι περισσότερες από αυτές που θα κάνει η jaccard αφού τα αρχεία 5 και 9 είναι μικρά με λιγότερες από 128 λέξεις. Σε μεγαλύτερα αρχεία, η signature similarity είναι η γρηγορότερη επιλογή.

**2ο Μέρος: Πειραματική αξιολόγηση προγράμματος**

2.1 Ανάλυση δοκιμαστικής μεθόδου

Για την διεξαγωγή των πειραμάτων, ήταν ασύμφορο να χρειάζεται η περιήγηση από την main() μέθοδο κάθε φορά. Γι’ αυτόν τον λόγο, δημιουργήσαμε την test() μέθοδο που δέχεται 6 ορίσματα και εκτελεί ανάλογα με αυτά τα τεστ. Ακολουθεί μια εξήγηση του κάθε ορίσματος:

**simMethod** = 1: Η μέθοδος ομοιότητας που θα χρησιμοποιηθεί, 1=Jaccard, 2=MinHash

**numOfPermutations** = 64: Το πλήθος των τυχαίων hash function που θα χρησιμοποιηθούν

**neighborsMethod** = 1: Η μέθοδος εύρεσης γειτόνων, 1=BruteForce, 2=LSH

**numOfNeighbors** = 2: Ο αριθμός γειτόνων που θα αναζητηθούν

**file** = 1: Το αρχείο εισόδου, 1=Enron, 2=Nips

**docNum** = 15000: Το πλήθος των εγγράφων που θα διαβαστούν

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΔεν προσθέσαμε επιλογή για την αλλαγή του threshold αφού σπάνια αλλάζαμε την τιμή του (παρόλα αυτά δοκιμάστηκαν αρκετές τιμές του στα τελικά τεστ). Για να αλλάξει αυτό, πρέπει κανείς να επεξεργαστεί τον κώδικα του LSHcount.

Ένα παράδειγμα εκτέλεσης είναι το test(2,64,2,2,1,15000), που εκτελεί LSH αναζήτηση 2 γειτόνων με χρήση signature similarity 64ων permutations σε 15000 έγγραφα του Enron αρχείου εισόδου.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

2.2 Ανάλυση δεδομένων που προκύπτουν

Η ενότητα αυτή περιέχει τα συμπεράσματα που προέκυψαν από τα πειράματα . Τα πειράματα έγιναν με τρόπο που αναφέρεται στην ενότητα (2.1). Ο σκοπός των πειραμάτων ήταν να εντοπίσουμε χρονικές διάφορες και κρίσιμα σημεία της υλοποίησης.

Οι βασικοί συνδυασμοί εκτελέσεων είναι τέσσερις:

* **Baseline** λύση: Brute force υπολογισμοί ομοιοτήτων (σύνολο D\*(D-1)/2) με χρήση Jaccard similarity (με ordered lists)
* **Minhash** λύση: Brute force υπολογισμοί ομοιοτήτων (σύνολο D\*(D-1)/2) με χρήση Signature similarity υπογραφών της Minhash
* **LSH+Jaccard** λύση: Φιλτράρισμα σημαντικότερων ζευγαριών με την LSH και υπολογισμός ομοιοτήτων με Jaccard similarity
* **LSH+Minhash** λύση: Φιλτράρισμα σημαντικότερων ζευγαριών με την LSH και υπολογισμός ομοιοτήτων με Signature similarity υπογραφών της Minhash

Τα αρχικά τεστ έγιναν για σύγκριση των παραπάνω συνδυασμών ως προς τον χρόνο εκτέλεσης και το average similarity που υπολογίζουν. Εξετάσαμε, επίσης, διαφορετικές τιμές του threshold και του αριθμού των permutations ώστε να βρούμε αν υπάρχει βέλτιστη τιμή ή αν προκύπτουν περιορισμοί (π.χ. από το αρχείο εισόδου).

2.2.1) Σύγκριση αποτελεσμάτων βάσει numOfPermutations

Πριν εξετάσουμε τον χρόνο εκτέλεσης, οφείλουμε να επιβεβαιώσουμε την ορθή εκτέλεση των μεθόδων. Η baseline λύση αποτελεί το μέτρο σύγκρισης και πρέπει ιδανικά κάθε άλλη λύση να τείνει σε αυτή. Εκτελούμε τις μετρήσεις αρχικά στο Enron αρχείο εισόδου για **15000 έγγραφα** εισόδου, **LSH threshold=0.5**, αναζήτηση **2 γειτόνων** και μεταβάλλοντας το πλήθος των hash function κάθε φορά.

Όπως φαίνεται και στο παραπάνω γράφημα, οι τιμές της minhash τείνουν στην baseline λύση όσο αυξάνουμε τον αριθμό των Hash functions. Το γεγονός αυτό ωφειλεται στην μείωση του σφάλματος που υπάρχει στην διαδικασία παραγωγής μητρώου υπογραφών. Πιο συγκεκριμένα, διαδικασία του ΜinHash δημιουργεί ένα σφάλμα. Το αναμενόμενο σφάλμα για k αριθμό HashFunctions είναι το πολύ 1/sqrt(k). Αυξάνοντας λοιπόν των αριθμό των HashFunctions το αναμενόμενο σφάλμα μειώνεται.

Για τα επόμενα τεστ θα χρησιμοποιήσουμε 64-128 hash function ώστε να πετύχουμε χαμηλό χρόνο εκτέλεσης και καλά αποτελέσματα.

2.2.2) Σύγκριση αποτελεσμάτων βάσει LSH threshold

Παρατηρήστε ότι στις παραπάνω δοκιμές χρησιμοποιήσαμε διαφορετικό threshold στο Nips αρχείο εισόδου απ’ ότι στο Enron. Αυτό οφείλεται στην έμφυτη μέση ομοιότητα που έχουν μεταξύ τους τα έγγραφα του κάθε αρχείου. Τα έγγραφα στο Nips παρουσιάζουν πολύ μικρή ομοιότητα μεταξύ τους, οπότε οφείλουμε να χρησιμοποιήσουμε μικρό threshold για την εύρεση όλων των κοντινότερων γειτόνων που ζητάμε. Τα έγγραφα στο Enron έχουν μεγαλύτερη ομοιότητα οπότε χρησιμοποιούμε μεγαλύτερο threshold. Ας δούμε την επήρεια του threshold στα αποτελέσματα του LSH στο Enron(15000) και στο Nips(1500) με 128 permutations:

Είναι φανερό ότι το threshold αποτελεί σημαντική παράμετρο στην εύρεση σωστών αποτελεσμάτων, ειδικά σε αρχεία όπως το Nips που η ομοιότητα μεταξύ των στοιχείων του είναι μικρή. Βέβαια ένα μικρό threshold πολλαπλασιάζει τον χρόνο εκτέλεσης του αλγορίθμου LSH, ενώ σε περιπτώσεις μικρών threshold μπορεί ακόμη και να φτάσει στον χρόνο εκτέλεσης το brute force. Αυτό ίσως είναι και το αδύναμο σημείο του LSH: σε αρχεία όπου αναζητούμε πολύ μικρούς βαθμούς ομοιότητας η εύρεση κοντινών ζευγαριών ίσως είναι ασύμφορη.

Στα παραπάνω γραφήματα παρατηρούμε ότι το καταλληλότερο threshold είναι **0.52** κατά την ανάλυση του Enron και **0.125** κατά την ανάλυση του Nips ώστε να λαμβάνουμε σωστά αποτελέσματα. Φυσικά, για μεγαλύτερο αριθμό γειτόνων πρέπει να μειώσουμε ακόμα περισσότερο τα παραπάνω threshold, αλλά για την αξιολόγηση της άσκησης διατηρούμε τον αριθμό αυτό σταθερό στους 2 γείτονες.

2.2.4) Σύγκριση αποτελεσμάτων βάσει χρόνου εκτέλεσης

Αφού καταλήξαμε στα καταλληλότερα threshold και numOfPermutations για κάθε αρχείο, μένει να εξετάσουμε τους χρόνους εκτέλεσης που είχε η κάθε εκδοχή. Συγκεκριμένα, εξετάζουμε τόσο το Enron όσο και το Nips, για **15000 και 1500 έγγραφα** αντίστοιχα, χρησιμοποιώντας **0.5** και **0.125 threshold** για το κάθε έγγραφο και έχοντας **64 numOfPermutations** ψάχνοντας να βρούμε **2 γείτονες**.

Η διαφορά στους χρόνους εκτέλεσης είναι εξαιρετική. Είναι εμφανές ότι η LSH είναι πολύ πιο γρήγορη από την BruteForce.

Θα εκτελέσουμε τον ίδιο έλεγχο και στο Nips αρχείο εισόδου:

Στο δεύτερο αρχείο εισόδου με μικρότερο αριθμό εγγράφων παρατηρούμε ότι η brute force με χρήση υπογραφών πλησιάζει αρκετά την LSH. Σε αυτό ευθύνεται το μικρό threshold της LSH αλλά και ο μικρός αριθμός αρχείων εισόδου.

Για να εξετάσουμε την επήρεια του πλήθους των αρχείων εισόδου, διεξάγουμε το αντίστοιχο τεστ παρακάτω.

2.2.5) Σύγκριση αποτελεσμάτων βάσει πλήθους εγγράφων εισόδου

Αξιοσημείωτη είναι η συμπεριφορά του προγράμματος κατά την εφαρμογή διαφορετικών αριθμών στο numOfDocuments. Συγκεκριμένα, δοκιμάσαμε την εκτέλεση όλων των μεθόδων με τη χρήση των καταλληλότερων **0.5** threshold και **64** αριθμό hash function για το έγγραφο Enron.

Ακολουθεί γράφημα που παρουσιάζει τις διαφορές:

Το αποτέλεσμα είναι ξεκάθαρο: σε κάθε περίπτωση η LSH είναι καλύτερη από τις υπόλοιπες μεθόδους όσον αφορά τον χρόνο. Παρατηρούμε ότι δεν υπάρχει τρομερή διαφορά μεταξύ της Jaccard similarity και signature similarity σε μικρές εισόδους και αυτό οφείλεται στον χρόνο που παίρνει για να παραχθεί το μητρώο υπογραφών. Όσο αυξάνεται ο αριθμός των εγγράφων, τόσο αυξάνεται και η επίδοση της sigSim.

Χρόνος εκτέλεσης LSH:

Η εφαρμογή του αλγορίθμου LSH γίνεται στο μητρώο υπογράφων που παρήξε η MinHash. Ο χρόνος που διαρκεί ο αλγόριθμος εξαρτάται κυρίως από τον χρόνο που απαιτείται για την παραγωγή του μητρώου αυτού. Η παρατήρηση αυτή επιβεβαιώνεται από τον παρακάτω πίνακα ο οποίος μας δείχνει τον συνολικό χρόνο που εκτελέστηκαν οι LSH+MINHASH σε δευτερόλεπτα.

Τελικά συμπεράσματα

Από την υλοποίηση των αλγορίθμων συμπεράναμε τα εξής:

Κατ’ αρχάς, δεν βρήκαμε κάποιον αριθμό permutations που παρήγαγε τα καλύτερα αποτελέσματα στον λιγότερο χρόνο. Όσο μεγαλύτερος τόσο καλύτερα αποτελέσματα βγάζουμε, με μία χρονική καθυστέρηση βέβαια για την παραγωγή και τον έλεγχο των υπογραφών.

Κατά δεύτερον, το threshold του LSH εξαρτάται κυρίως από το αρχείο εισόδου. Δεν υπάρχει threshold που να βρίσκει σωστά αποτελέσματα για κάθε αρχείο εισόδου. Όσο μεγαλώνει ο αριθμός των γειτόνων και όσο μικρότερο είναι το average similarity μεταξύ των εγγράφων, τόσο μικρότερο πρέπει να είναι το threshold. Και φυσικά, μικρότερο threshold συνεπάγεται μικρότερο rows per band.

Η brute force μέθοδος εγγυάται μια ορθή λύση στο πρόβλημα μας και αποτέλεσε κατάλληλο μέτρο σύγκρισης για όλες τις υπόλοιπες μεθόδους. Αποδείχτηκε, βέβαια, ασύμφορη η εκτέλεσή της για πλήθη αρχείων μεγαλύτερα από 1000.

Εν τέλει, συμπεραίνουμε ότι η LSH αποτελεί κατάλληλη μέθοδο για σύγκριση ομοιοτήτων μεταξύ αρχείων. Θεωρούμε χρησιμότερη, βέβαια, την χρήση της κατά την εύρεση κοντινών ζευγαριών και όχι τόσο κατά την εύρεση μέσων ομοιοτήτων, αλλά αυτό εξαρτάται πάντα από την εφαρμογή.