Τμημα Μηχανικων Ηλεκτρονικων Υπολογιστων και Πληροφορικης

Πολυτεχνικη Σχολη

Βελτιστοποίηση Γραφημάτων του Graphical Set-based Μοντέλου*

Διπλωματική εργασία

του

Νικόλαου Σκαμνέλου

skamnelos@ceid.upatras.gr

1041878

Επιβλέπων: Μακρής Χρήστος
Αναπληρωτής Καθηγητής

Συνεπιβλέπων: Ηλίας Αριστείδης
ΕΔΙΠ

Συνεπιβλέπων: Τσώλης Δημήτριος
Επίκουρος Καθηγητής

Πάτρα 31/12/2021

*Graphical Set-based μοντέλο ανάκτησης πληροφορίας

Περίληψη

Στόχος την παρούσας διπλωματικής εργασίας αποτελεί η βελτίωση των γραφημάτων του επεκτεταμένου με γραφήματα Set - Based μοντέλου, δίνοντας έμφαση στην εξάρτηση μεταξύ των όρων. Τα γραφήματα που παράγονται από το επεκτεταμένο με γραφήματα Set - Based μοντέλο είναι πλήρη, πράγμα που μεταφράζεται σε αλληλεξάρτηση κάθε όρου με όλους τους υπόλοιπους. Συνεπώς, ορίζεται ένα τμήμα ή και τμήματα κειμένου - παράθυρο - αλληλεξάρτησης στο οποίο εφαρμόζονται αλγόριθμοι παραγωγής γραφημάτων, δημιουργώντας έτσι γραφήματα με ουσιαστικότερη δομική ισχύ. Προτείνονται αρκετές μέθοδοι που προσεγγίζουν μέγεθος παραθύρου με βάση το μέγεθος κειμένου, ενώ δοκιμάζεται και μία μέθοδος που συνδυάζει πολλαπλά παράθυρα προσεγγίζοντας τη σχέση πρότασης - παραγράφου. Σημαντική παρατήρηση στη συγκεκριμένη προσέγγιση είναι η ύπαρξη ακμών που δρουν ως γέφυρες μεταξύ των υπογραφημάτων κάθε παραθύρου. Για τον λόγο αυτό γίνεται μελέτη των συγκεκριμένων κόμβων και ακμών ως προς την νοηματική σημασία τους. Αυτό, λοιπόν, μεταφράζεται σε εντοπισμό ασήμαντων κόμβων - stop-words αρχικά μεταξύ των γεφυρών με μεθόδους αποσύνθεσης σε επίπεδα πυρήνα ενώ τελικά προτείνεται μία μέθοδο δειγματοληψίας και εντοπισμού τέτοιων λέξεων σε ολόκληρη τη συλλογή.

Abstract

The aim of this thesis is to improve upon the graphs generated by the Graphical Set-based model, while focusing on the dependence among document terms. The Graphical Set-based model generates complete graphs, which means that every term of a given document is interdependent with the rest. Therefore, we define an interdependence part or parts of a document, called windows, in which we apply graph generation algorithms, thus creating more cohesive graphs. The proposed methods approximate the size of the window based on document length in words. There is also an attempt at creating an algorithm that combines multiple windows, thus approaching a sentence - paragraph relationship. For that reason, we elaborate on such nodes and edges on their semantic importance. Finally, the problem is consider a stop-word detection issue on bridge nodes, implementing algorithms using core decomposition resulting at last into a collection sampling method as a solution on the aforementioned problem.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Ευχαριστίες

Πρώτα θα ήθελα να ευχαριστήσω θερμά τον κ. Χρήστου Μακρή, αναπληρωτή καθηγητή του τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Πατρών, τόσο για την άψογη συνεργασία που αναπτύξαμε, όσο και για τις πάντα στοχευμένες παρατηρήσεις και συμβουλές που μου έδινε κατά την διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας. Πάνω από όλα όμως θα ήθελα να τον ευχαριστήσω για την ευκαιρία που μου έδωσε και που ήτανε πρόθυμος, από την πρώτη στιγμή κιόλας, να με αναλάβει, ως επιβλέποντάς καθηγητής. Επίσης, θα ήθελα να ευχαριστήσω τον κ. Αριστείδη Ηλία και τον κ. Δημήτρη Τσώλη για τη συμμετοχή τους στην επιτροπή παρουσίασης της διπλωματικής μου εργασίας.

Στην συνέχεια θα ήθελα να ευχαριστήσω τα αδέλφια μου: Μαργαρίτα και Κωστή, που ήταν πάντα στο πλευρό μου στα δύσκολα και που πάντα με υποστήριξαν και με ανέχτηκαν, ακόμα και ίσως όταν δεν θα έπρεπε. Ακόμα, θα ήθελα να ευχαριστήσω τους γονείς μου που έδωσαν άπειρη αγάπη και υποστήριξη καθ όλη την διάρκεια της ακαδημαϊκής μου σταδιοδρομίας.

Θα ήθελα επίσης να ευχαριστήσω μερικούς μου φίλους που με βοήθησαν και μου έκαναν τα χρόνια σαν φοιτητής τόσο πιο πολύ καλύτερα και ξεχωριστά. Οπότε θα ήθελα να ευχαριστήσω τους: Γιάννη Ακαρέπη, Νικόλαο Βασιλακόπουλο, Αλέξανδρο Ρόσιο, Νικόλαο Τσαφά, Γεώργιο Ράπτη και Αποστόλη Σπυρόπουλο.

Τελευταίο αλλά καθόλου ξεχασμένο θα ήθελα να ευχαριστήσω τον μεταπτυχιακό μου, αλλά πάνω από όλα αγαπητό μου φίλο Νικήτα Ρήγα Καλογερόπουλο, που με καθοδήγησε σε αυτά τα άγνωστα μονοπάτια καθ'όλη τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας. Είναι αλήθεια ότι χωρίς την συνεισφορά του, τον παρόν έγγραφο δεν θα ήταν ότι είναι σήμερα. Ελπίζω η διαδρομή για την ολοκλήρωση του να του ήταν όσο ευχάριστη όσο ήταν και για εμένα. Τον ευχαριστώ για όλες τις γνώσεις και την αγάπη που μου χάρισε, και ελπίζω, όχι Ξέρω ότι κάποια μέρα θα μου δοθεί η ευκαιρία να του το ανταποδώσω.

Στους αγαπημένους μου

Εισαγωγή

Για χιλιάδες χρόνια, το ανθρώπινο είδος είχε συνειδητοποίηση την σημασία της αρχειοθέτησης και της εύρεσης πληροφορίας. Με την άνοδο των υπολογιστών, έγινε εφικτή, η αποθήκευση μεγάλου όγκου πληροφορίας και σαν αποτέλεσμα, έγινε αναγκαία η εύρεση χρήσιμης πληροφορίας από αυτές της συλλογές. Το πεδίο της Ανάκτησης της Πληροφορίας (Information Retrieval - IR) εδραιώθηκε την δεκαετία του 1950 εξαιτίας αυτής της ανάγκης.

Οι πρώτοι που αναγνώρισαν την αξία της αποθήκευσης της πληροφορίας ήταν οι αρχαίοι Σουμέριοι, που από το 3000 π.Χ., είχαν καθιερωμένους χώρους στους οποίους αποθήκευαν πήλινες πλάκες, στις οποίες είχαν γράψει διάφορες επιγραφές. Ακόμα και οι Σουμέριοι είχαν συνειδητοποιήσει ότι η σωστή οργάνωση και η πρόσβαση σε αυτά τα αρχεία ήταν κρίσιμη για την αποδοτική χρήση της πληροφορίας. Μέσα στους αιώνες η αποθήκευση και η ανάκτηση της γραπτής πληροφορίας έγινε εξαιρετικά σημαντική, ειδικά με την εφεύρεση του χαρτιού και της τυπογραφίας. Λίγο αργότερα εφευρέθηκαν οι πρώτοι υπολογιστές και οι άνθρωποι συνειδητοποίησαν ότι μπορούν να χρησιμοποιηθούν για την αποθήκευση τεράστιου όγκου πληροφορίας. Το 1945 ο Vannevar Bush μίλησε πρώτος για συστήματα αυτόματης πρόσβασης και ανάκτησης πληροφορίας στη δημοσίευση του με όνομα "As We May Think" (Bush, 1996). Το 1957 ο H.P. Luhn πρότεινε την χρήση λέξεων ως όροι δεικτοδότησης και την μέτρηση της επικάλυψης τους ως κριτήριο για την ανάκτηση πληροφορίας (Luhn, 1957). Την δεκαετία του 1960 εφευρέθηκε το SMART system από τον Gerard Salton (G. Salton, 1971), το οποίο επέτρεψε στους ερευνητές να πειραματιστούν με διάφορες τεχνικές για την βελτίωση της ικανότητας της ανάκτησης. Τις δεκαετίες του 1970 και του 1980 εφευρέθηκαν πάρα πολλά μοντέλα για την ανάκτηση της πληροφορίας, όπως για παράδειγμα το μοντέλο διανυσματικού χώρου (Vector Space Model) (G. Salton et al., 1975), το οποίο μάλιστα θα αναλυθεί στο κεφάλαιο 3.1. Ωστόσο, μέχρι τότε δεν υπήρχαν συλλογές με πολλά κείμενα, οπότε τα μοντέλα ως τώρα είχαν πειραματιστεί μόνο σε μικρές συλλογές, με αποτέλεσμα να μην είναι γνωστό αν αυτά μπορούν να χρησιμοποιηθούν και σε συλλογές μεγάλου όγκου πληροφορίας. Αυτό άλλαξε το 1992 όπου ιδρύθηκε το Text Retrieval Conference, η απλά TREC (Harman, 1993). Το TREC ήταν μία σειρά από συνέδρια σε συνεργασία με την αμερικάνικη υπηρεσία NIST, με σκοπό την ενθάρρυνσή

της έρευνας σε συλλογές με πάρα πολλά κείμενα.

Τα παραπάνω είναι μερικά από τα κύρια σημεία της ιστορίας της ανάκτησης της πληροφορίας. Δυστυχώς, η λεπτομερή περιγραφή της ιστορίας της ανάκτησης της πληροφορίας είναι πέρα από την εμβέλεια αυτής της διπλωματικής εργασίας. Ας περάσουμε τώρα σύντομα στον ορισμό της ανάκτησης της πληροφορίας.

Η ανάκτηση πληροφορίας ασχολείται με την επεξεργασία, την αναπαράσταση, την προσπέλαση και την αποθήκευση τμημάτων πληροφορίας, ώστε να ικανοποιηθούν η πληροφοριακές ανάγκες ενός χρήστη. Πιο συγκεκριμένα, στην παρούσα διπλωματική εργασία ασχολούμαστε με την βελτίωση των γραφημάτων, και ως αποτέλεσμα και της ανάκτησης, του Graphical Set-Based Model (GSB) (Kalogeropoulos et al., 2020), το οποίο με την σειρά του βασίζεται στο Set-Based Model (Possas et al., 2002). Επίσης, με αφορμή την χρήση γραφημάτων, γίνεται προσπάθεια ανάπτυξης τεχνικών για τον εντοπισμό ασήμαντων λέξεων (stopwords) με γραφήματα.

Η δομή της παρούσας διπλωματικής εργασίας είναι η εξής. Στο πρώτο κεφάλαιο περιγράφονται μερικές βασικές έννοιες για την ανάκτηση πληροφορίας.

Στο δεύτερο κεφάλαιο περιγράφονται μερικές τεχνικές αναπαράστασης κειμένων με την βοήθεια γραφημάτων, όπως και μερικές τεχνικές εντοπισμού σημαντικών κόμβων σε γραφήματα.

Στο τρίτο κεφάλαιο περιγράφονται μερικά μοντέλα ανάκτησης της πληροφορίας, δίνοντας έμφαση κυρίως στο Graphical Set-Based μοντέλο.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα προτεινόμενα μοντέλα και το ολισθούμενο παράθυρο των (Rousseau and Vazirgiannis, 2013b) για την δημιουργία γραφημάτων κειμένων με σκοπό την βελτίωση των αποτελεσμάτων ανάκτησης πληροφορίας.

Στο πέμπτο κεφάλαιο παρουσιάζονται διάφοροι τρόποι εντοπισμού ασήμαντων λέξεων.

Στο έκτο κεφάλαιο γίνεται παρουσίαση και μελέτη των αποτελεσμάτων μέσω μετρικών απόδοσης, ενώ παράλληλα προτείνονται και νέοι δίοδοι έρευνας και βελτιστοποίησης.

Τέλος, στο έβδομο κεφάλαιο παρατίθεται το παράρτημα, που περιλαμβάνει πίνακες αποτελεσμάτων και συναρτήσεις που χρησιμοποιήθηκαν συμπληρωματικά στην υλοποίηση.

Chapter 1

Εισαγωγή στα Συστήματα Ανάκτησης Πληροφορίας

1.1 Διαδικασία της Ανάκτησης Πληροφορίας

Όπως έχει ήδη αναφερθεί, στόχος της ανάκτησης πληροφορίας είναι η ικανοποίηση της πληροφοριακής ανάγκης του χρήστη. (Baeza-Yates and Ribeiro-Neto, 2008). Η διαδικασία της ανάκτησης της πληροφορίας είναι η παρακάτω. Αρχικά, κάποιος χρήστης θέτει ένα ερώτημα στο σύστημα που αναλαμβάνει την ανάκτηση από μία "βιβλιοθήκη" πληροφορίας. Τα ερωτήματα είναι καλά ορισμένες δηλώσεις για πληροφοριακές ανάγκες, όπως για παράδειγμα ένα αλφαριθμητικό για μία μηχανή αναζήτησης του παγκόσμιου ιστού. Στην επιστήμη της ανάκτησης της πληροφορίας, ένα ερώτημα δεν αντιστοιχίζεται σε ένα μοναδικό αντικείμενο στην συλλογή. Αντίθετα, αντιστοιχίζεται σε διάφορα αντικείμενα, όπου πολλές φορές είναι και με διαφορετικούς βαθμούς συνάφειας (relevance). Συνάφεια ονομάζεται ο βαθμός σχετικότητας ενός αντικειμένου σε σχέση με το ερώτημα που έχει τεθεί στο σύστημα.

Τώρα, ένα αντικείμενο είναι μία οντότητα που αναπαρίσταται από πληροφορία που περιέχεται σε μία συλλογή πληροφορίας. Τα ερωτήματα των χρηστών αντιστοιχίζονται με την πληροφορία στη συλλογή πληροφορίας. Ωστόσο, τα δεδομένα δεν είναι δομημένα, όπως για παράδειγμα σε μία κλασσική βάση δεδομένων. Επίσης, τα ερωτήματα που θέτει ο χρήστης πολλές φορές είναι ασαφή ή περιέχουν λεκτικές και συντακτικές ατέλειες. Τα δύο αυτά προβλήματα έχουν ως συνέπεια τα αποτελέσματα που επιστρέφονται από το σύστημα ανάκτησης πολλές φορές να μην ταιριάζουν ολοκληρωτικά με το ερώτημα ή ίσως και καθόλου. Άρα τα αποτελέσματα πρέπει να καταταχθούν σύμφωνα με την συνάφεια τους με το ερώτημα. Αυτή είναι και η κύρια διαφορά της ανάκτησης της πληροφορίας σε σχέση με της ανάκτηση στις κλασσικές βάσεις δεδομένων που χρησιμοποιούν γλώσσες όπως για παράδειγμα SQL.

Ανάλογα τον τομέα που χρησιμοποιείται η ανάκτηση της πληροφορίας, οι οντότητες

πληροφορίας μπορεί να έχουν την μορφή κειμένου, εικόνων, ήχου ή ακόμα και βίντεο κ.α. Συνήθως τα δεδομένα δεν αποθηκεύονται απευθείας στο σύστημα, αλλά αναπαριστώνται από αναπληρωτές των αρχείων ή μεταδεδομένα. Όπως αναφέρθηκε ήδη πολλά συστήματα ανάκτησης πληροφορίας αξιολογούν τα αποτελέσματα αριθμητικά ανάλογα τον βαθμό συνάφειας με το ερώτημα και κατατάζουν τα αποτελέσματα σύμφωνα με αυτή την μετρική. Έπειτα, τα κορυφαία αποτελέσματα παρουσιάζονται στον χρήστη. Η διαδικασία είναι δυνατόν να επαναληφθεί για την βελτίωση των αποτελεσμάτων αν ο χρήστης εκλεπτύνει το ερώτημα. Αυτό γίνεται είτε με την αναδιατύπωση του ερωτήματος, ή με τεχνικές ανάδρασης όπως για παράδειγμα την σήμανση των συναφή αποτελεσμάτων για την βελτίωση της ανάκτησης (Singhal and Google, 2001).

1.2 Τεχνικές Βελτίωσης της Ανάκτησης

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, ένα κείμενο μπορεί να είναι συναφή με κάποιο ερώτημα, είτε ολοκληρωτικά, είτε ακόμα και τμηματικά. Στα πρώτα μοντέλα, όπως για παράδειγμα το Boolean μοντέλο, που παίρνει το όνομα του από την Boolean Algebra, τα κείμενα μπορούσαν να ήταν είτε συναφή, είτε όχι. Τα ερωτήματα διατυπωνόταν σαν boolean εκφράσεις και αν ένα κείμενο ήταν συναφές, λάμβανε "βάρος" 1, ενώ αν δεν ήταν συναφές λάμβανε "βάρος" 0. Ένα τέτοιο μοντέλο, ωστόσο, παρουσίασε πολλούς και διάφορους περιορισμούς, τόσο στην ανάκτηση πληροφορίας, όσο και στην παρουσίαση των αποτελεσμάτων. Για παράδειγμα, με το boolean μοντέλο δεν υπήρχε η δυνατότητα για τμηματικό ταίριασμα κειμένων με κάποιο ερώτημα και δεν πρόσφερε κάποια σειρά κατάταξης των κειμένων. Για να λυθούν αυτά τα προβλήματα, από την δεκαετία του 1960 κιόλας, εισηγήθηκε η έννοια της ανάθεσης βάρους (term weighting) (Baeza-Yates and Ribeiro-Neto, 2008).

Ανάθεση Βάρους

Οι όροι ενός κειμένου δεν είναι όλοι εξίσου σημαντική για την περιγραφή του περιεχομένου του κειμένου. Για την ακρίβεια, μερικοί όροι είναι απλά περισσότερο ασαφής από άλλους. Υπάρχουν διάφορα χαρακτηριστικά των όρων που είναι χρήσιμα για την αξιολόγηση της σημαντικότητας ενός όρου μέσα σε μία συλλογή κειμένων. Για παράδειγμα, ένας όρος που εμφανίζεται σε όλα τα κείμενα τις συλλογής δεν βοηθάει στην διαφοροποίηση των κειμένων κατά την διαδικασία της ανάκτησης.

Άρα για να χαρακτηρίσουμε έναν όρο, όσο αφορά την σημαντικότητα του μέσα σε ένα κείμενο j, ορίζουμε μία μετρική που θα ονομάζεται βάρος ενός όρου i και θα συμβολίζεται με $W_{i,j}$. Τα βάρη των όρων ενός κειμένου δίνουν την δυνατότητα κατάταξης των κειμένων για ένα δοσμένο ερώτημα. Δεδομένου των βαρών όλων των

όρων ενός κειμένου, είναι δυνατόν ένα κείμενο να αναπαρασταθεί ως ένα διάνυσμα $d_j = (W_{1,j}, W_{2,j}, ..., W_{i,j})$, όπου j είναι το κείμενο και 1 έως i είναι οι όροι.

Ως τώρα έχει γίνει αναφορά στην θεωρητική πλευρά των βαρών ενός κειμένου. Στην συνέχεια ακολουθεί η διαδικασία υπολογισμού των βαρών. Τα βάρη των όρων ενός κειμένου μπορούν να υπολογιστούν με διάφορους τρόπους. Στην πιο απλή τους μορφή, μπορούν να υπολογισθούν ως την συχνότητα εμφάνισης του όρου στο κείμενο. Επεκτείνοντας, τον ορισμό της συχνότητας εμφάνισης σε ένα κείμενο, είναι δυνατόν να ορισθεί η συνολική συχνότητα εμφάνισης ενός όρου σε όλη την συλλογή. Έστω $f_{i,j}$ η συχνότητα εμφάνισης του όρου i στο κείμενο j. Τότε, η συνολική συχνότητα του όρου i, F_i, θα υπολογίζεται από τον παρακάτω τύπο:

$$F_i = \sum_{j=1}^{N} f_{i,j} \tag{1.1}$$

Όπου το N συμβολίζει τον αριθμό των κειμένων της συλλογής (Baeza-Yates and Ribeiro-Neto, 2008).

Ωστόσο, η πιο σημαντική μετρική για τον υπολογισμό του βάρους ενός όρου είναι η $TF \times IDF$, η οποία θα συζητηθεί στο κεφάλαιο 3.1.1, μαζί με το μοντέλο διανυσματικού χώρου (vector space model).

Για τα κλασσικά μοντέλα ανάκτησης πληροφορίας, τα βάρη των όρων θεωρούνται ανεξάρτητα μεταξύ τους, δηλαδή το βάρος $W_{i,j}$ δεν παρέχει καμία πληροφορία για το βάρος $W_{i+1,j}$. Αυτή είναι προφανώς μία απλοποίηση και δεν ισχύει στην πράξη. Για παράδειγμα, οι λέξεις "ηλεκτρονικός" και "υπολογιστής" εμφανίζονται συνήθως μαζί και όχι ξεχωριστά, για ένα κείμενο για τους υπολογιστές. Είναι φανερό ότι οι δύο λέξεις έχουν σχέση μεταξύ του και τα βάρη τους πρέπει να το δείχνουν αυτό. Στις προτεινόμενες μεθόδους, η χρήση των γραφημάτων (Graph-of-word)(Rousseau and Vazirgiannis, 2013b) δίνει την δυνατότητα εκμετάλλευσής αυτών των σχέσεων ανάμεσα στις λέξεις, οι οποίες μάλιστα οδηγούν και σε καλύτερα αποτελέσματα ανάκτησης.

1.3 Αξιολόγηση ενός Συστήματος Ανάκτησης

Ένα σημαντικό χαρακτηριστικό της ανάκτησης της πληροφορίας είναι η αντικειμενική αξιολόγηση του συστήματος. Από τα πρώτα χρόνια κιόλας, ήταν προφανές ότι η αποδοτική και αντικειμενική αξιολόγηση των συστημάτων ανάκτησης πληροφορίας θα έπαιζε ένα πολύ σημαντικό ρόλο στην εξέλιξη της επιστήμης της ανάκτησης της πληροφορίας, ειδικά λαμβάνοντας υπόψιν την πειραματική της φύση. Οι δοκιμές Cranfield την δεκαετία του 1960s, εδραίωσαν τα επιθυμητά χαρακτηριστικά ενός συστήματος

ανάκτησης. Τα δύο κυριότερα χαρακτηριστικά είναι η υψηλή ακρίβεια (precision) και η υψηλή ανάκληση (recall) (Singhal and Google, 2001).

Ακρίβεια και Ανάκληση

Για την κατανόηση των εννοιών της ακρίβειας και της ανάκλησης είναι απαραίτητο πρώτα να γίνει αναφορά σε κάποιες γενικές κατηγορίες που εντάσσονται τα αποτελέσματα, στην συγκεκριμένη περίπτωση τα κείμενα ή γενικότερα οι μονάδες πληροφορίας της συλλογής. Οι κατηγορίες είναι τέσσερις. Αρχικά, αν ο αλγόριθμος ανάκτησης κρίνει ένα κείμενο ως συναφή και αυτό το κείμενο είναι πράγματι συναφή με το ερώτημα, τότε το κείμενο θεωρείται αληθώς θετικό (true positive). Αντίθετα, αν ο αλγόριθμος ανάκτησης κρίνει ένα κείμενο ως συναφή αλλά δεν είναι στην πραγματικότητα συναφή με το ερώτημα τότε το κείμενο θεωρείται ψευδώς θετικό (false positive). Επίσης, αν ένα κείμενο κριθεί ως μη συναφή και δεν ήταν συναφή με το ερώτημα, τότε το κείμενο θεωρείται ως αληθές αρνητικό (true negative). Τέλος, αν ένα κείμενο κριθεί ως μη συναφές με το ερώτημα, αλλά στην πραγματικότητα ήταν συναφές, τότε το κείμενο θεωρείται ψευδοαρνητικό (false negrative).

	True	False	
Positive	tp	fp	pp
Negative	tn	fn	pn

Table 1.1: Positive - Negative notations

Όπου tp είναι τα true positives, fp έιναι τα false positives, tn είναι τα true negative, fn είναι τα false negative, pp είναι τα συνολικά predicted positives και pn είναι τα συνολικά predicted negatives.

Έχοντας τα παραπάνω υπόψιν, θα γίνει τώρα αναφορά στους ορισμούς της Ανάκλησης και της Ακρίβειας (Powers, 2008).

Ανάκληση

Με απλά λόγια η ανάκληση είναι το ποσοστό των αποτελεσμάτων που κρίθηκαν σωστά ως αληθές θετικά προς όλα τα θετικά αποτελέσματα. Για την ανάκτηση πληροφορίας, μπορεί να διατυπωθεί και ως τα συναφή κείμενα που ανακτήθηκαν προς όλα τα συναφή κείμενα της συλλογής, δοθέντος ενός ερωτήματος. Συνήθως, η μετρική της ανάκλησης δεν έχει ιδιαίτερη αξία στην ανάκτηση πληροφορίας, υπό την λογική ότι υπάρχουν πάρα πολλά συναφή κείμενα και δεν παίζει σημαντικό ρόλο ποια υποομάδα συναφή κειμένων ανακτήθηκε, ώστε να μην ξέρουμε τίποτα για την συνάφεια των κειμένων που δεν ανακτήθηκαν. Αυτό ωστόσο δεν ισχύει σε εφαρμογές, όπως

για παράδειγμα ιατρικές, που κύριο ενδιαφέρον είναι να βρεθούν όσο πιο πολλά αληθές θετικά γίνεται (Powers, 2008). Η ανάκληση ορίζεται ως εξής:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{1.2}$$

Ακρίβεια

Από την άλλη πλευρά, η ακρίβεια είναι μία μετρική η οποία εκφράζει το ποσοστό των προβλεπόμενων θετικών που είναι πράγματι αληθές θετικά. Είναι η κύρια μετρική που εστιάζεται στους κλάδους της μηχανικής μάθησης, της εξόρυξης της πληροφορίας και της ανάκτησης πληροφορίας (Powers, 2008). Η ακρίβεια ορίζεται ως εξής:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{1.3}$$

Οι ορισμοί της ακρίβειας και της ανάκλησης μπορούν να διατυπωθούν και με την βοήθεια συνόλων. Έστω ένα ερώτημα q με σύνολο R τα κείμενα που είναι συναφή για το ερώτημα, και P τα κείμενα που ανακτήθηκαν από τον αλγόριθμος ανάκτησης.

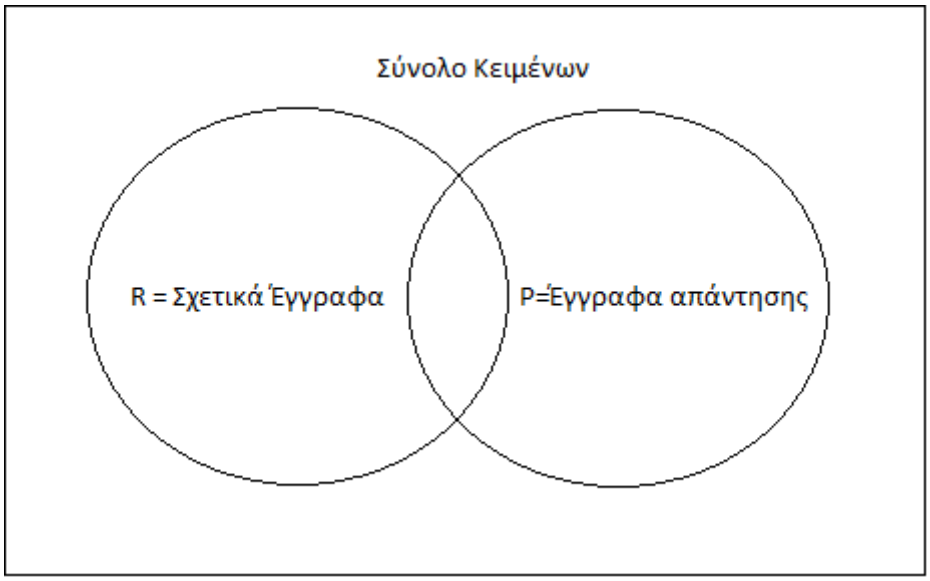

Figure 1.1: Precision - Recall

Τότε:

$$Recall = \frac{|R \cap P|}{|R|} \tag{1.4}$$

$$Precision = \frac{|R \cap P|}{|P|} \tag{1.5}$$

Ιδιαίτερο ενδιαφέρον παρουσιάζει το γεγονός ότι εξετάζοντας με τη σειρά ένα προς ένα τα επιστρεφόμενα έγγραφα οι τιμές ανάκλησης και ακρίβειας μεταβάλλονται ανάλογα με τη σχετικότητα ή μη του συγκεκριμένου εγγράφου. Εξαιτίας αυτού μπορεί να παραχθεί ένα διάγραμμα μεταξύ ανάκλησης-ακρίβειας, το οποίο αποτελεί μια από τις κλασσικές επιλογές στην εκτίμηση απόδοσης ανάκλησης. Ακόμα, δίνεται η δυνατότητα είτε εξετάζοντας σημείο προς σημείο τα διαγράμματα, είτε με τον υπολογισμό του μέσου όρου των σημείων να συγκρίνουμε δύο ή περισσότερες μηχανές αναζήτησης μεταξύ τους ώστε να παραχθούν ασφαλή συμπεράσματα τα για το ποια υπερισχύει (Baeza-Yates and Ribeiro-Neto, 2008). Για αυτό το λόγο χρησιμοποιείται και ως μετρική στην παρούσα εργασία κατά την αξιολόγηση των αποτελεσμάτων.

Chapter 2

Γραφήματα και κείμενα

2.1 Εισαγωγή

Στα υποκεφάλαια που ακολουθούν θα αναλυθούν μερικοί τρόποι με τους οποίους είναι δυνατόν τα κείμενα μίας συλλογής να αναπαρασταθούν με την βοήθεια γραφημάτων. Επίσης, γίνεται περιγραφική ανάλυση μερικών μεθόδων εντοπισμού σημαντικών κόμβων σε ένα γράφημα. Σημαντικοί κόμβοι ενός γραφήματος, θεωρούνται οι κόμβοι οι οποίοι περιέχουν την περισσότερη πληροφορία σε ένα γράφημα.

2.2 Κείμενο ως γράφημα

Έχουν υπάρξει πολλές προσπάθειες να αναπαρασταθούν κείμενα ή λέξεις ως γραφήματα. Σε αυτό το κεφάλαιο αναλύονται συνοπτικά μερικές από αυτές τις προσπάθειες. Ένα λοιπόν τέτοιο γράφημα κειμένου είναι το γράφημα που προτάθηκε από τον (Medeiros Soares et al., 2005), στο οποίο τα κείμενα χωρίζονται σε συλλαβές λέξεων και δημιουργείται ένα γράφημα $G = (V, E)$, του οποίου οι κόμβοι είναι συλλαβές και υπάρχει μία ακμή ανάμεσα σε δύο κόμβους, όταν υπάρχει μία λέξη στο κείμενο που περιέχει τις συλλαβές που αντιστοιχούν σε αυτούς τους κόμβους. Για τα γραφήματα του ο M.Soares χρησιμοποίησε την πορτογαλική γλώσσα.

Ένα άλλο παράδειγμα είναι ένα γράφημα "παράθεσης" που προτάθηκε από τον (Bordag et al., 2003). Ο Bordag παρατήρησε ότι μερικές λέξεις εμφανίζονται μαζί με κάποιες άλλες λέξεις με πολύ μεγαλύτερη πιθανότητα από άλλες και ότι αυτό είναι σημασιολογικά σημαντικό. Όρισε λοιπόν, ότι μία τέτοια εμφάνιση δύο ή περισσότερων λέξεων μέσα σε ένα καλά ορισμένο κομμάτι πληροφορίας, όπως για παράδειγμα μία πρόταση ή ένα κείμενο, ως μία "παράθεση" ή "σύμφαση". Έστω a, b ο αριθμός των προτάσεων που περιέχουν τα A και B αντίστοιχα, k ο αριθμός των προτάσεων που περιέχουν και το A και το B και n ο συνολικός αριθμός των προτάσεων. Ορίζεται μία μετρική σημαντικότητας την πιθανότητα εμφάνισης των λέξεων μαζί. Για $x = \frac{ab}{n}$:

$$\mathrm{sig}(A,B) = \frac{-\log\left(1 - e^{-x}\sum_{i=0}^{k-1}\frac{1}{i!}\cdot x^i\right)}{\log n} \tag{2.1}$$

Για $2x < k$, γίνεται η παρακάτω απλοποίηση, η οποία είναι ευκολότερη στον υπολογισμό:

$$\mathrm{sig}(A,B) = \frac{(x - k\log x + \log k!)}{\log n} \tag{2.2}$$

Γενικά, αυτή η μετρική δίνει σημασιολογικά αποδεκτές "παραθέσεις" για τιμές πάνω από ένα εμπειρικό κατώφλι. Έτσι χρησιμοποιείται αυτή η μετρική για τον εντοπισμό λέξεων σε μία πρόταση, οι οποίες αποτελούν "παράθεση".

Το γράφημα "παράθεσης" του Bordag είναι ένα μη βεβαρημένο μη κατευθυνόμενο γράφημα, του οποίου οι κόμβοι αντιστοιχούν σε λέξεις, οι οποίες ανήκουν σε τουλάχιστον μία "παράθεση" και ανάμεσα σε δύο κόμβους υπάρχει ακμή μόνο αν οι λέξεις, οι οποίες αντιστοιχούν σε αυτούς τους δύο κόμβους, ανήκουν στην ίδια παράθεση. Όσο πιο υψηλή είναι η μετρική σημαντικότητας δύο κόμβων, τόσο πιο πυκνό είναι το πάχος της ακμής τους.

Ένα γράφημα κειμένου μπορεί επίσης, σύμφωνα με τους (Ferrer i Cancho, Solé, et al., 2004), (Ferrer i Cancho, Capocci, et al., 2007), να βασίζεται στην σύνταξη των λέξεων μέσα στο κείμενο. Το γράφημα που περιέγραψαν είναι ένα μη βεβαρημένο κατευθυνόμενο γράφημα, στο οποίο οι κόμβοι είναι οι λέξεις του κειμένου. Κάθε λέξη, ενός λεξιλογίου, μπορεί να ανήκει σε μία από δύο κατηγορίες. Η πρώτη κατηγορία λέγεται επικεφαλίδες λέξεις (headers) και η δεύτερη τροποποιητές (modifiers). Οι τροποποιητές αλλάζουν την σημασία μίας επικεφαλίδας λέξης μέσα σε μία πρόταση. Αν μία επικεφαλίδα λέξη εμφανίζεται στην ίδια πρόταση με μία τροποποητή λέξη, τότε εισάγεται μεταξύ τους μία ακμή στο γράφημα. Οι ακμές μεταξύ δύο κόμβων είναι δυαδικές, δηλαδή είτε υπάρχουν είτε δεν υπάρχουν, και δηλώνουν την εξάρτηση μεταξύ των δύο λέξεων. Η κατεύθυνση των ακμών του γραφήματος μπορεί να είναι είτε από επικεφαλίδες λέξεις σε τροποποιητές λέξεις, είτε από τροποποιητές λέξεις σε επικεφαλίδες λέξεις. Παρακάτω ακολουθεί ένα παράδειγμα:

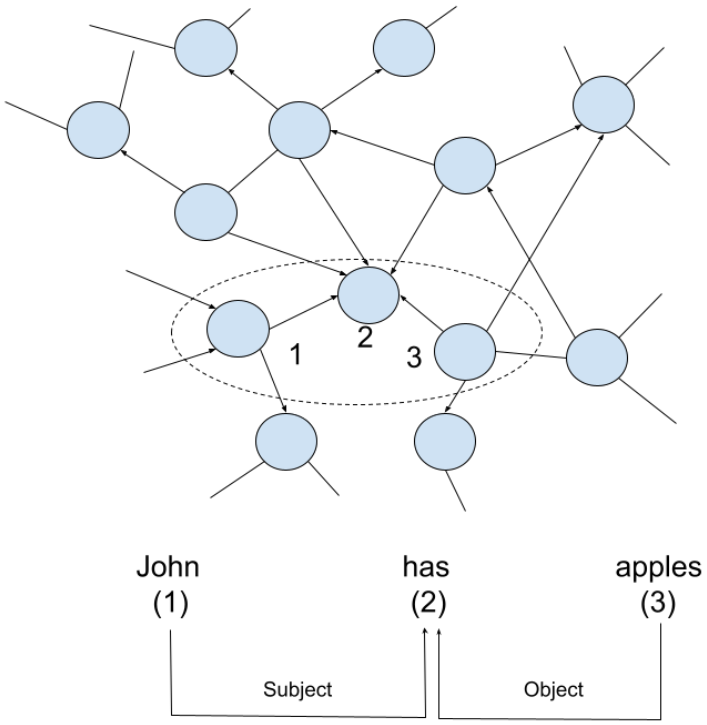

Figure 2.1: Παράδειγμα της σχέσης που περιγράφηκε.

Τα σημεία στίξης και η έσω-ακμές(ακμές με αρχή και τέλος τον ίδιο κόμβο) απορρίφτηκαν στο συντακτικό γράφημα.

Η αναπαράσταση κειμένου σαν γράφημα έχει βοηθήσει ακόμα και στην ανάπτυξη συστημάτων αυτόματου ορθογραφικού ελέγχου. Το SpellNet προτάθηκε πρώτα από τους (Choudhury et al., 2007), και περιγράφει ένα βεβαρημένο μη κατευθυνόμενο πλήρες γράφημα, του οποίου οι κόμβοι είναι οι λέξεις ενός λεξικού και οι ακμές συμβολίζουν την ορθογραφική ομοιότητα μεταξύ των δύο λέξεων που ενώνουν. Το βάρος κάθε ακμής αποτελεί την απόσταση Levenshtein((Levenshtein, 1966)), υποθέτοντας ότι οι μετακινήσεις, προσθέσεις και αφαιρέσεις κοστίζουν μία μονάδα. Για κάθε κόμβο ορίζεται επίσης ένα βάρος, το οποίο ισούται με την συχνότητα εμφάνισης της συγκεκριμένης λέξεις στην γλώσσα.

Υπάρχει μία παραλλαγή του παραπάνω γραφήματος ως ένα μη βεβαρημένο μη κατευθυνόμενο γράφημα με την βοήθεια κατωφλιών θ. Για ένα κατώφλι θ, ορίζεται το μη βεβαρημένο γράφημα $G_\theta = (V, E_\theta)$, το οποίο είναι υπογράφημα του $G = (V, E)$, και στο οποίο μία ακμή $e \in E$ αποτελεί και ακμή του G_θ, δηλαδή έχει βάρος ίσο με την μονάδα, αν και μόνο αν το βάρος της ακμής στο G είναι μικρότερο του κατωφλιού θ. Ειδάλλως το βάρος της ακμής ισούται με μηδέν. Με άλλα λόγια, το σύνολο ακμών E_θ

αποτελείται μόνο από τις ακμές που έχουν βάρος στο E λιγότερο ή ίσο με το κατώφλι θ.

2.3 Graph-of-Words

Σε αυτό το υποκεφάλαιο γίνεται αναφορά στο μοντέλο αναπαράστασης κειμένων Graph-of-word, το οποίο προτάθηκε από τους Rousseau και Vazirgiannis (Rousseau and Vazirgiannis, 2013b). Σύμφωνα με το μοντέλο, είναι δυνατόν ένα κείμενο με λέξεις, ή πιο συνηθισμένα μία ιστοσελίδα, να αναπαρασταθεί ως ένα μη βεβαρημένο κατευθυνόμενο γράφημα, του οποίου οι κόμβοι να είναι οι λέξεις του κειμένου και οι ακμές να είναι οι εμφανίσεις μεταξύ των όρων σε ένα ολισθούμενο παράθυρο. Οι κατευθύνσεις των ακμών ορίζονται σύμφωνα με την σειρά των λέξεων μέσα στο κείμενο.

Υπάρχουν διάφορες παραλλαγές για το μοντέλο Graph-of-word. Μία από αυτές είναι οι ακμές του γραφήματος να είναι μη κατευθυνόμενες. Έστω ότι υπάρχουν δύο όροι i,j, οι οποίοι εμφανίζονται στο κείμενο με δύο τρόπους. Μερικές φορές, είναι πρώτος στο κείμενο, ο i όρος σε σχέση με τον j, και άλλες φορές είναι πρώτος ο j όρος σε σχέση με τον i. Σε αυτή την περίπτωση οι ακμές $E_{i,j}$ και $E_{j,i}$ αναφέρονται στην ίδια ακμή. Μία ακόμη παραλλαγή είναι οι ακμές του γραφήματος να είναι βεβαρημένες. Τα βάρη των ακμών μπορεί να υπολογισθούν με διάφορες κλασσικές μετρικές, όπως για παράδειγμα TF-IDF ή BM-25 (Robertson and Zaragoza, 2009) ή με κάποια άλλη μετρική, όπως στην περίπτωση του Graphical Set-based μοντέλου, το οποίο θα αναλυθεί στο κεφάλαιο 3.1.3.

2.4 K-core Decomposition - MainCore

Έστω ένα γράφημα κειμένου $G = (V, E)$ και ένας ακέραιος αριθμός k. Σύμφωνα με τους (Rousseau and Vazirgiannis, 2015), το υπογράφημα $H_k = (V', E')$, όπου $V' \subseteq V$ και $E' \subseteq E$, ονομάζεται k-πυρήνας ($k - core$) ή πυρήνας τάξης k του G, αν και μόνο αν ισχύει ό,τι για κάθε κόμβο $u \in V'$ ο βαθμός του, deg_u, είναι μεγαλύτερος ή ίσος του ακέραιου k και το υπογράφημα H_k είναι το μέγιστο δυνατό με αυτή την ιδιότητα. Όταν το γράφημα είναι μη βεβαρημένο, ο βαθμός ενός κόμβου u ισούται με το άθροισμα των γειτόνων του. Αντίστοιχα, όταν το γράφημα είναι βεβαρημένο, ο βαθμός ενός κόμβου u ισούται με το άθροισμα των βαρών των προσκείμενων ακμών σε αυτόν.

Ο αριθμός πυρήνα ενός κόμβου u ($core(u)$), αποτελεί τον πυρήνα με μεγαλύτερο βαθμό που περιέχει αυτό το κόμβο. Ο πυρήνας με μέγιστο βαθμό ονομάζεται κύριος πυρήνας ($MainCore$), και το σύνολο όλων των πυρήνων, από το 0-πυρήνα μέχρι

και τον κύριο πυρήνα ονομάζεται ανάλυση (αποσύνθεση) σε επίπεδα πυρήνων ($k-core\ decomposition$) του γραφήματος G. Στο εξής, οι κ-πυρήνες θα αναφέρονται ως $k-core$, ο κύριος πυρήνας ως $MainCore$ και η ανάλυση σε επίπεδο πυρήνων ως $k-core\ decomposition$.

Ο σκοπός του $k-core\ decomposition$ είναι η εξαγωγή λέξεων κλειδιών ενός κειμένου αναλύοντας το γράφημα αυτού. Οι A.Tixier, F.Malliaros και M.Vazirgiannis (Tixier et al., 2016) απέδειξαν ωστόσο, ότι, το $k-core$ παράγει καλά αποτελέσματα όσο αφορά την συνεκτικότητά, αλλά όχι τα καλύτερα. Επίσης, απέδειξαν ότι η διατήρηση μόνο του $MainCore$ δεν είναι βέλτιστη, αφού πάρα πολλές λέξεις κλειδιά υπάρχουν στα κατώτερα επίπεδα του $k-core$. Τέλος, το $k-core\ decomposition$ λειτουργεί επιλέγοντας ή απορρίπτοντας μεγάλες ομάδες λέξεων, το οποίο έχει σαν αποτέλεσμα την μείωση της προσαρμοστικότητας και της απόδοσης της μεθόδου.

2.5 Density - CoreRank

Οι μέθοδοι Density - CoreRank προτάθηκαν από τους A.Tixier, F.Malliaros και M.Vazirgiannis (Tixier et al., 2016) με σκοπό την αντιμετώπιση των περιορισμών του $k-core\ decomposition$ που αναφέρθηκαν παραπάνω.

2.5.1 Density

Με βάση την υπόθεση ότι οι λέξεις κλειδιά δεν περιέχονται όλες στο $MainCore$ ενός γραφήματος κειμένου, ένα απλό κριτήριο για την ανάλυση των πυρήνων του γραφήματος είναι το *density* που βασίζεται στην πυκνότητα, δηλαδή ξεκινώντας την ανάλυση από το $MainCore$ κατεβαίνουμε την ιεραρχία του $k-core$ του γραφήματος, όσο διατηρούνται οι συνεκτικές ιδιότητες του $MainCore$ και σταματάμε όταν αυτές χάνονται. Η πυκνότητα ενός γραφήματος επιπέδου k δίνεται από τον παρακάτω τύπο:

$$density(G) = \frac{|E|}{|V|(|V|-1)} \tag{2.3}$$

Όπου το $G = (V, E)$ είναι το γράφημα του επιπέδου. Για κάθε επίπεδο του $k-core\ decomposition$ υπολογίζεται η πυκνότητα και αποθηκεύεται σε σε έναν πίνακα. Μόλις ολοκληρωθεί η διαδικασία ο πίνακας αυτός δίνεται σαν είσοδο στην elbow μέθοδο η οποία υπολογίζει το επίπεδο με τη βέλτιστη πυκνότητα.

Ο τρόπος με τον οποίο λειτουργεί η *elbow* είναι εξαιρετικά απλός. Θεωρεί μια ευθεία που σχηματίζεται από το πρώτο και το τελευταίο σημείο του πίνακα και υπολογίζει πιο από τα υπόλοιπα σημεία έχει την μεγαλύτερη απόσταση. Η απόσταση ενός σημείου $P(x_0, y_0)$ από μία ευθεία γνωρίζοντας μόνο δύο σημεία $P_1(x_1, y_1)$,$P_2(x_2, y_2)$ της, χωρίς

να υπολογιστεί η εξίσωση της ευθείας δίνεται από τον εξής τύπο:

$$distance(P_1, P_2, P) = \frac{|(y_2 - y_1) \cdot x_0 - (x_2 - x_1) \cdot y_0 + x_2 \cdot y_1 - x_1 \cdot y_2|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \quad (2.4)$$

2.5.2 CoreRank

Η *density* μέθοδος αντιμετωπίζει τα πρώτα δύο προβλήματα που παρουσιάστηκαν στην ενότητα 2.4. Ωστόσο, όπως και η κανονική μορφή του $k-core\ decomposition$, το *density* επίσης επιλέγει μεγάλες ομάδες κόμβων με αποτέλεσμα να επιλέγονται και πολύ κόμβοι που δεν περιέχουν πολύ πληροφορία. Η μέθοδος *corerank* δημιουργήθηκε με σκοπό την αντιμετώπιση αυτού του προβλήματος και η λειτουργία της είναι εξαιρετικά απλή. Η *corerank* αξιολογεί τους κόμβους του γραφήματος, τους ταξινομεί σε φθίνουσα σειρά και επιλέγει τους κορυφαίους $p\%$ όρους, όπου p το επιθυμητό ποσοστό, ή τους κόμβους πριν τη γραμμή του *elbow*. Τέλος, για να μειωθεί ο βαθμός της λεπτομέρειας, ενώ ταυτόχρονα διατηρώντας όσο το δυνατόν περισσότερο επιθυμητή πληροφορία, σε κάθε κόμβο ανατίθεται το άθροισμα των *core numbers* των γειτονικών σε αυτών κόμβων.

Chapter 3

Μοντέλα Ανάκτησης Πληροφορίας

3.1 Μοντέλα Ανάκτησης

3.1.1 Vector Space Model

Το Μοντέλο διανυσματικού χώρου (Vector Space Model) είναι ένα πολύ σημαντικό μοντέλο ανάκτησης πληροφορίας και προτάθηκε από τους G. Salton, A. Wong και C.S. Yang το 1975 (G. Salton et al., 1975). Στο μοντέλο διανυσματικού χώρου τα κείμενα και το ερώτημα αναπαριστώνται ως διανύσματα t διαστάσεων, με t να είναι ο αριθμός των μοναδικών όρων της συλλογής. Το διάνυσμα που αντιστοιχεί σε κάποιο κείμενο j έχει την παρακάτω μορφή:

$$\vec{D_j} = \{W_{1j}, W_{2j}, ..., W_{ij}, ..., W_{tj}\} \tag{3.1}$$

Όπου D_j είναι το κείμενο j και W_{ij} είναι το βάρος του όρου i στο κείμενο. Τα βάρη μπορούν να ορισθούν με διάφορους τρόπους, όπως για παράδειγμα δυαδικά:

$$W_{ij} = \begin{cases} 1 & \text{Αν υπάρχει ο όρος i στο κείμενο j} \\ 0 & \text{Αν δεν υπάρχει ο όρος i στο κείμενο j} \end{cases} \tag{3.2}$$

Η χρήση, ωστόσο, δυαδικών βαρών εμφανίζει κάποια προβλήματα, αφού οι πληροφορίες που είναι δυνατόν να εξαχθούν για ένα κείμενο είναι πολύ περιορισμένες. Για αυτό το λόγο αναπτύχθηκαν διάφορες άλλες μετρικές βαρών. Η πιο γνωστή και πιο διαδεδομένη είναι η χρήση του $TF \times IDF$. Το $TF \times IDF$ είναι ένα είδος βάρους το οποίο αποτελείται από δύο σκέλη. Πρώτα, το TF (term frequency) σκέλος, το οποίο αντιστοιχεί στην συχνότητα εμφάνισης ενός όρου μέσα σε ένα κείμενο. Συνήθως, χρησιμοποιείται απλή απαρίθμηση των εμφανίσεων του όρου μέσα στο κείμενο. Έχουν ωστόσο προταθεί και άλλες παραλλαγές υπολογισμού και κανονικοποίησης (Baeza-Yates and Ribeiro-Neto, 2008), (Gerard Salton and Buckley, 1988). Μερικές από αυτές φαίνονται στο πίνακα 3.1.

Δυαδικά	{0,1}
Απλή Απαρίθμηση	F_{ij}
Απλή Κανονικοποίηση	$1 + \log F_{ij}$
Διπλή 0,5 Κανονικοποίηση	$0.5 + 0.5 \cdot \frac{F_{ij}}{MAX(F_{ij})}$
Διπλή 1 Κανονικοποίηση	$K + (K-1) \cdot \frac{F_{ij}}{MAX(F_{ij})}$

Table 3.1: Υπολογισμοί TF_i

Μοναδιαία	1
Απλή Ανάστροφη Απαρίθμηση	$\log\left(\frac{N}{n_i}\right)$
Απλή λογαριθμική κανονικοποίηση	$\log\left(1 + \frac{N}{n_i}\right)$
Ανάστροφη κανονικοποίηση περισσοτερων εμφανίσεων	$\log\left(1 + \frac{\mathrm{MAX}(n_i)}{n_i}\right)$

Table 3.2: Υπολογισμοί IDF_i

Όπου $F_{i,j}$ είναι η συχνότητα του όρου i στο κείμενο j και $MAX(F_{ij})$ αναφέρεται στην μέγιστη συχνότητα εμφάνισης που συναντάται σε όρους του κειμένου.

Το δεύτερο σκέλος είναι η συχνότητα εμφάνισης στα κείμενα (document frequency) και αντιστοιχεί στον αριθμός των κειμένων, στα οποία εμφανίζεται ένας όρος τουλάχιστον μία φορά. Έχουν προταθεί επίσης πολλοί τρόποι υπολογισμού της συχνότητας εμφάνισης στα κείμενα. Μερικοί φαίνονται στο πίνακα 3.2.

Όπου N είναι ο αριθμός των κειμένων της συλλογής, n_i είναι ο αριθμός των κειμένων, στα οποία εμφανίζεται ο όρος i και το $MAX(n_i)$ υπολογίζει πόσες φορές εμφανίζεται ο όρος με τη μεγαλύτερη συχνότητα εμφάνισης στη συλλογή, δηλαδή, αφορά τον όρου που εμπεριέχεται στα περισσότερα κείμενα.

Συνδυάζοντάς τα δύο αυτά σκέλη δημιουργείται το $TF \times IDF$, το οποίο δείχνει πόσο σημαντικός είναι ένας όρος για κάποιο κείμενο. Αυξάνεται όσο αυξάνεται η συχνότητα ενός όρου μέσα στο κείμενο, και μειώνεται όταν αυτός ο όρος εμφανίζεται σε πολλά κείμενα.

Αντίστοιχα με τα διανύσματα κειμένων αναπαρίσταται και το διάνυσμα για το ερώτημα:

$$\vec{Q} = \{W_{1q}, W_{2q}, ..., W_{iq}, ..., W_{tq}\} \tag{3.3}$$

Τα βάρη για το ερώτημα μπορούν να υπολογισθούν όπως και τα βάρη για τα κείμενα.

Τέλος, χρειάζεται ένα τρόπος να συγκριθούν τα κείμενα με το ερώτημα. Για να επιτευχθεί αυτό χρησιμοποιείται η ομοιότητα του συνημίτονου:

$$\text{sim}\,(q, d_j) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Όσο πιο μικρή είναι η γωνία που σχηματίζει ένα κείμενο με το ερώτημα τόσο πιο συναφή είναι για το συγκεκριμένο ερώτημα.

3.1.2 Set-based Μοντέλο

Εισαγωγή

Σε αυτό το υποκεφάλαιο περιγράφεται η λειτουργία και οι δυνατότητες του Set-Based Model, όπως προτάθηκε από τους B.Possas, N.Ziviani, W.Meira και B.Ribeiro-Neto (Possas et al., 2002). Επίσης, γίνεται αναφορά σε διάφορους ορισμούς της θεωρίας των συνόλων όρων (termsets) και στην εξαγωγή συχνών συνόλων όρων με την χρήση του αλγόριθμου apriori.

Σύνολα όρων

Έστω μία συλλογή κειμένων D, με λεξιλόγιο T = {k_1,k_2,...,k_t}, δηλαδή το σύνολο των t μοναδικών όρων που μπορούν να εμφανιστούν σε ένα κείμενο της συλλογής. Υπάρχει μία συνολική κατάταξη στους όρους του λεξιλογίου, η οποία είναι βασισμένη στην αλφαριθμητική σειρά των όρων, τέτοια ώστε $k_i < k_j$ για $1 \leq i \leq j \leq t$. Τότε, ένα σύνολο n όρων, έστω s, ορίζεται ως ένα καταταγμένο σύνολο n μοναδικών όρων, το οποίο αποτελεί υποσύνολο του λεξιλογίου T ($s \subseteq T$) και οι όροι του ακολουθούν την κατάταξη που αναφέρθηκε προηγούμενος.

Έστω το σύνολο S = {s_1,s_2,...,s_{2^t}}. Αυτό αποτελεί όλα τα δυνατά σύνολα s που μπορούν να εμφανιστούν σε ένα κείμενο της συλλογής D. Κάθε κείμενο της συλλογή D, μπορεί να χαρακτηριστεί με ένα διάνυσμα στον χώρο που ορίζεται από τα σύνολα όρων. Για κάθε σύνολο s_i, με $1 \leq i \leq 2^t$, συσχετίζεται μία ανεστραμμένη λίστα, η οποία περιλαμβάνει αναγνωριστικά για τα κείμενα τα οποία περιλαμβάνουν το σύνολο. Επίσης, ορίζεται η συχνότητα εμφάνισης ds_i του συνόλου s_i, ως τον αριθμό τον εμφανίσεων του συνόλου στην συλλογή D.

Ορισμός 1

Ένα σύνολο s_i είναι συχνό όταν η συχνότητα εμφάνισης του ds_i είναι μεγαλύτερη από ένα δοσμένο κατώφλι. Αυτό το κατώφλι ονομάζεται υποστήριξη (support) ή ελάχιστη συχνότητα συνόλου.

Ορισμός 2

Ένα κλειστό σύνολο cs_i είναι ένα συχνό σύνολο, το οποίο είναι το μεγαλύτερο ανάμεσα στα υποσύνολα που το απαρτίζουν και που εμφανίζονται στο ίδιο σύνολο κειμένων.

Ορισμός 3

Το μέγιστο σύνολο είναι ένα συχνό σύνολο, το οποίο δεν αποτελεί υποσύνολο σε κανένα άλλο συχνό σύνολο.

Τα μέγιστα και τα κλειστά σύνολα βοηθάνε στην μείωση των απαραίτητων συνόλων για τον χαρακτηρισμό ενός κειμένου. Τα απαραίτητα σύνολα παράγονται με την χρήση κάποιου αλγόριθμου εξόρυξης κανόνων συσχέτισης, όπως για παράδειγμα ο apriori.

Set-Based Model

Στο Set-Based Model ένα κείμενο χαρακτηρίζεται από τα κλειστά σύνολα όρων, τα οποία κιόλας εξάγονται από το ίδιο το κείμενο. Για κάθε κλειστό σύνολο όρων συσχετίζονται δύο βάρη. Πρώτα, ένα το οποίο περιγράφει την σημαντικότητα του κλειστού συνόλου όρων στο κείμενο, και δεύτερον ένα το οποίο περιγράφει την σημαντικότητα του ίδιου συνόλου για όλη την συλλογή. Παρόμοια, ένα ερώτημα περιγράφεται από κλειστά σύνολα όρων, στα οποία αναθέτετε επίσης ένα βάρος, το οποίο περιγράφει την σημαντικότητα κάθε συνόλου στο ερώτημα. Η αλγεβρική αναπαράσταση από τα κλειστά σύνολα όρων, ενός κειμένου, αλλά και ενός ερωτήματος είναι ένα διάνυσμα 2^t διαστάσεων στο ευκλείδειο χώρο, όπου t οι μοναδικοί όροι που εμφανίζονται στην συλλογή.

Το βάρος ενός κλειστού συνόλου όρων, έστω i, σε ένα κείμενο j υπολογίζεται από το παρακάτω τύπο:

$$w_{i,j} = sf_{i,j} \times ids_i = sf_{i,j} \times \log \frac{N}{ds_i} \tag{3.4}$$

Όπου το $sf_{i,j}$ είναι ο αριθμός των εμφανίσεων του συνόλου i στο κείμενο j, ids_i είναι η ανάστροφη συχνότητα εμφάνισης του συνόλου i στη συλλογή και το N είναι ο αριθμός των κειμένων στην συλλογή.

Όπως αναφέρθηκε παραπάνω τα κείμενα και τα ερωτήματα αναπαριστώνται ως διανύσματα. Σύμφωνα με αυτό, για κάθε κείμενο που περιέχει κάποιο κλειστό σύνολο όρων του ερωτήματος, υπολογίζεται μία μετρική ομοιότητας, η οποία υπολογίζεται από τον ακόλουθο τύπο:

$$\text{sim}\,(q, d_j) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{s \in C_q} w_{s,j} \times w_{s,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Όπου το $w_{s,j}$ είναι το βάρος που σχετίζεται με το κλειστό σύνολο όρων s στο κείμενο j, $w_{s,q}$ είναι το βάρος που σχετίζεται με το κλειστό σύνολο όρων s στο ερώτημα q, C_q είναι το σύνολο όλων των κλειστών συνόλων όρων για τα οποία ισχύει $s \subseteq q$, $w_{i,j}$ είναι το βάρος του όρου i στο κείμενο j και $w_{i,q}$ το βάρος του όρου i στο ερώτημα q.

3.1.3 Graphical Set-based Μοντέλο

Το Graphical Set-based (GSB) μοντέλο, των Καλογερόπουλο, Δούκα, Καναβό και Μακρή (Kalogeropoulos et al., 2020), αποτελεί μία επέκταση του Set-based μοντέλου. Στο GSB, η διαδικασία της δεικτοδότησης (των όρων) γίνεται χρησιμοποιώντας γραφήματα, αλλά η αναπαράσταση των κειμένων και των ερωτημάτων παραμένει η ίδια με το Set-based μοντέλο. Ακόμα, το GSB ακολουθεί μία εντελώς διαφορετική προσέγγιση, όσο αφορά την διαδικασία υπολογισμού των βαρών. Αναλύοντας τα γραφήματα κειμένων σε επίπεδα πυρήνων (K-core Decomposition), εισάγει την έννοια της σημαντικότητας των κόμβων, τους οποίους θεωρεί κιόλας ως λέξεις κλειδιά (keywords).

Κάθε κείμενο αναπαριστάται ως ένα μη κατευθυνόμενο σταθμισμένο γράφημα. Κατά την κατασκευή των γραφημάτων κειμένων, το GSB υποθέτει ότι κάθε όρος σχετίζετε με τους υπόλοιπους όρους του κειμένου. Κάθε όρος ενός δοθέντος κειμένου, αναπαρίσταται ως έναν μοναδικό κόμβο στο γράφημα, ενώ παράλληλα η σχέση μεταξύ δύο όρων στο κείμενο, αναπαριστάται ως μία ακμή με αρχή και τέλος τους αντίστοιχους κόμβους των όρων. Σε κάθε τέτοια ακμή ανατίθεται ένα βάρος, το οποίο συμβολίζεται ως W_{out}. Επίσης, κάθε κόμβος του γραφήματος, έχει μία εσωτερική ακμή (self loop), η οποία έχει τον ίδιο κόμβο ως αρχή και τέλος. Σε αυτόν, ανατίθεται ένα βάρος, το οποίο θα συμβολίζεται ως W_{in}. Τα παραπάνω βάρη υπολογίζονται σύμφωνα με τα θεωρήματα που ακολουθούν:

Θεώρημα 1

Δοθέντος ενός κόμβου N_i, ο οποίος ανήκει σε ένα κείμενο D_j, με συχνότητα όρου tf_i, το βάρος της εσωτερικής ακμή (N_i, N_i) υπολογίζετε ως εξής:

$$W_{in} = \frac{tf_i \times (tf_i + 1)}{2} \tag{3.5}$$

Θεώρημα 2

Δοθέντος δύο κόμβων N_i, N_j, οι οποίοι ανήκουν σε ένα κείμενο D_j, με συχνότητα όρων tf_i, tf_j αντίστοιχα, το βάρος της εξωτερικής ακμής (N_i, N_j) υπολογίζετε ως εξής:

$$W_{out} = tf_i \times tf_j, \quad \text{όταν} \quad N_i \neq N_j \tag{3.6}$$

Σύμφωνα με αυτά τα δύο θεωρήματα, η πολυπλοκότητά χρόνου είναι $O(n^2)$ + $O(m^2)$, όπου n είναι ο αριθμός των όρων του κειμένου και m είναι ο αριθμός των ξεχωριστών όρων που υπάρχουν στο κείμενο.

Αρχικά, ο αλγόριθμος διατρέχει το κείμενο και παράγει πλειάδες που περιέχουν κάθε μοναδικό όρο, καθώς και τη συχνότητα τους στο κείμενο. Στη συνέχεια, για κάθε έναν από αυτούς τους μοναδικούς όρους δημιουργεί έναν κόμβο στο γράφημα και αρχικοποιεί την εσωτερική ακμή, όπως και τις εξωτερικές του ακμές προς τους άλλους κόμβους. Τέλος, υπολογίζει τα βάρη σύμφωνα με τα θεωρήματα (3.5) και (3.6).

Απαλοιφή Ακμών

Τα γραφήματα που παράγονται είναι πλήρης, λόγου της υπόθεσης ότι κάθε όρος σχετίζετε με όλους τους υπόλοιπους όρους του κειμένου, με αποτέλεσμα όλοι οι κόμβοι να έχουν βαθμό ίσο με N-1, όπου N ο αριθμός των κόμβων στο γράφημα. Οι εσωτερικές ακμές δεν λαμβάνονται υπόψιν. Αυτό αποτελεί πρόβλημα αφού ο GSB, για να λειτουργήσει σωστά, βασίζετε στο βαθμό κάθε κόμβου του γραφήματος. Για να λυθεί το παραπάνω πρόβλημα, είναι αναγκαίο να διαγραφούν κάποιες ακμές. Προτιμούνται αυτές, που με την διαγραφή τους, ελαχιστοποιείται η απώλεια πληροφορίας.

Αρχικά, υπολογίζεται ο μέσος όρος των βαρών όλων των ακμών και στη συνέχεια αφαιρούνται όλες οι ακμές που έχουν βάρος μικρότερο από αυτό, πολλαπλασιασμένο με ένα ποσοστό P. Το P είναι μία πειραματική μεταβλητή και είναι διαφορετική για κάθε συλλογή. Αφού ολοκληρωθεί η παραπάνω διαδικασία, εντοπίζονται οι σημαντικοί όροι σύμφωνα με αυτά που συζητήθηκαν στο κεφάλαιο 2.4 και 2.5.

Συνολικό Γράφημα Συλλογής

Αφού παραχθούν τα γραφήματα κειμένων και εντοπιστούν οι σημαντικοί όροι, δημιουργείται ένα συνολικό γράφημα για όλη την συλλογή. Αυτό, περιλαμβάνει όλους τους όρους της συλλογής σαν κόμβους και όλες τις εναπομείναντες ακμές των γραφημάτων των κειμένων. Τα βάρη των ακμών, του συνολικού γραφήματος, υπολογίζονται ως το άθροισμα όλων των επιμέρους βαρών των αντίστοιχων ακμών σε κάθε γράφημα,

στο οποίο οι ακμές υπάρχουν.

Αρχικά, το συνολικό γράφημα της συλλογής U είναι ίσο με το κενό γράφημα. Σε κάθε επανάληψη του αλγόριθμου, το συνολικό γράφημα U θα ανανεώνετε με ένα γράφημα κειμένου G. Η ανανέωση γίνετε σύμφωνα με τη διαδικασία που ακολουθεί. Πρώτα, ελέγχεται κάθε όρος t του γραφήματος G και προσδιορίζετε αν αυτός είναι σημαντικός κόμβος. Αν είναι, τότε καθορίζετε η τιμή σημαντικότητας h. Για τους όρους που δεν αποτελούν σημαντικοί κόμβοι, η τιμή σημαντικότητας h είναι ίση με 1. Στη συνέχεια, γίνεται έλεγχος αν υπάρχει ο όρος t στο συνολικό γράφημα U. Αν υπάρχει, τότε το εσωτερικό βάρος $\mathrm{W_{in}}$ του όρου t του συνολικού γραφήματος U αθροίζεται με το αντίστοιχο εσωτερικό βάρος $\mathrm{W_{in}}$ του όρου t του γραφήματος G, πολλαπλασιασμένο με τη τιμή σημαντικότητας h. Έπειτα, γίνετε ανανέωση των τιμών όλων των εξωτερικών ακμών, στις οποίες ανήκει ο όρος t. Στην περίπτωση που δεν υπάρχει κάποιος κόμβος ή ακμή στο συνολικό γράφημα U, δημιουργείτε ένας κόμβος ή ακμή και η διαδικασία της ανανέωσης συνεχίζει.

Όταν ολοκληρωθεί η κατασκευή του συνολικού γραφήματος της συλλογής, το βάρος κάθε κόμβου k υπολογίζεται ως εξής:

$$NW_k = log(1 + a \times \frac{W_{out_k}}{(W_{in_k} + 1) \times (ng_k + 1)}) \times log(1 + b \times \frac{1}{ng_k + 1}) \tag{3.7}$$

Όπου το $\mathrm{W_{out_k}}$ και το $\mathrm{W_{in_k}}$ είναι το άθροισμα των εξωτερικών και εσωτερικών ακμών ενός κόμβου k αντίστοιχα. Ο αριθμός των γειτονικών κόμβων συμβολίζεται με ng_k. Τέλος, τα a και b είναι παράμετροί βαρύτητας σε σχέση με το Set-based model.

Ανάκτηση Πληροφορίας με το GSB

Η διαδικασία ανάκτησης είναι ίδια με το Set-based Model. Αρχικά, παράγονται σύνολα όρων με την χρήση του αλγόριθμου Apriori. Έπειτα, συνδυάζονται, ανά δύο, σύνολα τα οποία διαφέρουν μόνο κατά ένα όρο, δημιουργώντας νέα σύνολα εκτεταμένα κατά έναν όρο.

Στην συνέχεια, για κάθε σύνολο S_i υπολογίζεται μία νέα μετρική TN_{S_i}, η οποία είναι το γινόμενο των βαρών του γραφήματος, κάθε όρου του συνόλου.

$$TN_{S_i} = \prod_{k \in S_i} NW_k \tag{3.8}$$

Έπειτα, συμπεριλαμβάνεται αυτή τη νέα μετρική στο τύπο υπολογισμού βάρους ενός συνόλου όρων S_i μέσα σε ένα κείμενο D_j, όπου το Sf_{ij} αναπαριστά την συχνότητά όρου (TF) και το $\frac{N}{dS_i}$ την αντίστροφη συχνότητα κειμένου (IDF) του συγκεκριμένου

συνόλου.

$$W_{S_{ij}} = (1 + logSf_{ij}) \times log(1 + \frac{N}{dS_i}) \times tnw_{S_i} \tag{3.9}$$

Ο τύπος για το βάρος του ερωτήματος είναι ο παρακάτω:

$$W_{S_{iq}} = log(1 + \frac{N}{dS_i}) \tag{3.10}$$

Τέλος, τα κείμενα και τα ερωτήματα αναπαριστώνται ως διανύσματα:

$$\begin{aligned} \vec{d_j} &= (W_{S_{1j}},\ W_{S_{2j}},\ \ldots\ , W_{S_{2^n j}}) \\ \vec{Q} &= (W_{S_{1q}},\ W_{S_{2q}},\ \ldots\ , W_{S_{2^n q}}) \end{aligned} \tag{3.11}$$

Επειδή, το σύνολο στοιχείων βασίζεται στους όρους του ερωτήματος, το μέγεθος των διανυσμάτων δεν μπορεί να ξεπερνάει το 2^{n}, όπου n ο αριθμός των λέξεων του ερωτήματος.

Για την ομοιότητα μεταξύ των κειμένων και του ερωτήματος, χρησιμοποιείται η ομοιότητα του συνημίτονου (cosine similarity).

$$sim(\vec{Q}, \vec{d_j}) = \frac{\vec{d_j} \times \vec{Q}}{\|\vec{d_j}\| \times \|\vec{Q}\|} \tag{3.12}$$

Το αποτέλεσμα αυτής της διαδικασίας χρησιμοποιείται σαν συνάρτηση βαθμολόγησης του μοντέλου.

Chapter 4

Προτεινόμενο μοντέλο

4.1 Εισαγωγή

Το GSB, όπως αναφέρθηκε και στο κεφάλαιο 3.1.3, θεωρεί ότι όλες οι λέξεις σε ένα κείμενο σχετίζονται ισάξια μεταξύ τους, το οποίο οδηγεί στην δημιουργία πλήρες γραφημάτων κειμένων. Αυτά, με την σειρά τους, αποτελούν πρόβλημα στον εντοπισμό σημαντικών κόμβων, αφού λόγου της πληρότητας τους, ο κύριος πυρήνας αποτελείται από ένα μόνο επίπεδο και όλο το γράφημα θεωρείτε σημαντικό. Μία λύση είναι, η εκτέλεση ενός επιπλέον σταδίου, αφότου δημιουργηθούν τα γραφήματα, κατά το οποίο γίνεται απαλοιφή ακμών. Με αυτόν τον τρόπο μετατρέπονται τα γραφήματα σε μη πλήρης και γίνεται εφικτός ο εντοπισμός των σημαντικών κόμβων. Περισσότερες πληροφορίες, για την συγκεκριμένη προσέγγιση, υπάρχουν στο κεφάλαιο 3.1.3.

Στην παρούσα διπλωματική εργασία θα εξεταστεί μία εντελώς διαφορετική προσέγγισή στην λύση του παραπάνω προβλήματος. Αντί να γίνεται επεξεργασία των γραφημάτων, αφότου αυτά έχουν δημιουργηθεί, προτείνεται ένας εντελώς διαφορετικός τρόπος δημιουργίας των γραφημάτων κειμένων, που βασίζεται στην ανάλυση των κειμένων σε επίπεδο πρότασης ή και παραγράφου. Καταρρίπτεται η υπόθεση ότι όλες οι λέξεις σχετίζονται ισάξια μεταξύ τους και εισάγεται η έννοια συσχέτισης όρων σε επίπεδο πρότασης και σε επίπεδο παραγράφου. Η προσέγγιση αυτή βασίζεται στην πρόταση επέκτασης του GSB μοντέλου των Καλογερόπουλο, Δούκα, Καναβό και Μακρή (Kalogeropoulos et al., 2020)

Τα κείμενα χωρίζονται, με την χρήση πλαισίων ή παραθύρων, σε κομμάτια τα οποία θα ονομάζονται παράθυρα ενός κειμένου. Ανάλογα με το μέγεθος του παραθύρου θα αναφερόμαστε σε αυτά σαν παράθυρο πρότασης, είτε σαν παράθυρο παραγράφου, όπου αυτό χρησιμοποιείται. Με την χρήση των παραθύρων, παράγονται μη πλήρη γραφήματα κειμένων, τα οποία μάλιστα είναι σημασιολογικά πιο ισχυρά από αυτά του GSB. Στη συνέχεια, προαιρετικά, μπορεί να εφαρμοστεί κάποια τεχνική για $k-core\ decomposition$ γραφήματος, όπως αυτές περιγράφονται στο κεφάλαιο 2.4 και

2.5.

Έπειτα, δημιουργούνται τα αναστραμμένα ευρετήρια και εφαρμόζεται κανονικά το Set-based μοντέλο για την απάντηση των ερωτημάτων της συλλογής. Οι συλλογές που χρησιμοποιήθηκαν ήταν κυρίως η CF Collection (Shaw et al., 1991), που αποτελείται από 1240 ξεχωριστά κείμενα και 100 ερωτήματα, αλλά και η NPL Collection, η Time Collection και η Cranfield Collection (*Test collections* n.d.). Περισσότερες πληροφορίες για τις υπόλοιπες συλλογές στο κεφάλαιο 6.

4.2 Περιγραφή των Μοντέλων

Παρακάτω γίνεται παρουσίαση των προτεινομένων μοντέλων. Σκοπός των μοντέλων μας είναι, με την χρήση παραθύρων, να εξεταστούν τρόποι με τους οποίους τα γραφήματα κειμένων που κατασκευάζονται να είναι όσο το δυνατόν πιο σημασιολογικά ισχυρά. Εξετάζονται πέντε διαφορετικές μέθοδοι χωρισμού των κειμένων με παράθυρα, όπως επίσης και η μέθοδος ολισθούμενου παραθύρου, όπως περιγράφεται από τους (Rousseau and Vazirgiannis, 2013b). Τέλος, θα γίνει αναφορά σε μερικές εναλλακτικές προσεγγίσεις που απέτυχαν.

4.2.1 Παράθυρο σταθερού μεγέθους

Αποτελεί την πιο απλή προσέγγιση στο πρόβλημα. Η κεντρική ιδέα είναι, ότι κάθε κείμενο χωρίζεται με βάση ένα παράθυρο σταθερού μεγέθους (σε λέξεις) W_{size}. Το μέγεθος του παραθύρου W_{size} είναι μία πειραματική μεταβλητή και διαφέρει για κάθε συλλογή. Τα γραφήματα κειμένου που παράγονται με αυτή την διαδικασία είναι μη κατευθυνόμενα και σταθμισμένα.

Αρχικά, κάθε κείμενο χωρίζεται σε κομμάτια που θα ονομάζονται παράθυρα W. Αυτό γίνεται ξεκινώντας από την αρχή του κειμένου και σύμφωνα με το W_{size}, το κείμενο χωρίζεται σε παράθυρα, μέχρι να φτάσει στο τέλος αυτού. Όλα τα παράθυρα W ενός κειμένου έχουν μέγεθος ίσο με W_{size}, εκτός ίσως από το τελευταίο παράθυρο, στην περίπτωση που το μέγεθος του κειμένου δεν είναι ακέραιο πολλαπλάσιο του W_{size}. Στη συνέχεια, τα παραγόμενα παράθυρα εξετάζονται ως προς το περιεχόμενο τους. Για κάθε όρο του παραθύρου δημιουργείται ένας κόμβος στο γράφημα κειμένου, εφόσον δεν υπάρχει ήδη. Έπειτα, ξεκινώντας από τον πρώτο όρο, γίνεται σύγκριση με τους υπολειπόμενους δεξιούς όρους. Για κάθε δεξιό όρο, που είναι διαφορετικός από τον αρχικό, αρχικοποιείται στο γράφημα, εφόσον δεν υπάρχει ήδη, μία ακμή που ονομάζεται εξωτερική ακμή και έχει αρχή τον κόμβο που αντιστοιχεί στον αρχικό όρο και τέλος τον κόμβο που αντιστοιχεί στο δεξιό όρο. Το βάρος αυτή της ακμής θα αυξάνεται

κατά μία μονάδα και ονομάζεται $Weight_{out}$. Στην περίπτωση που ο δεξιός όρος είναι ίδιος με τον αρχικό, αρχικοποιείται στο γράφημα, εφόσον δεν υπάρχει ήδη, μία ακμή που ονομάζεται εσωτερική ακμή και έχει για αρχή και τέλος τον αρχικό όρο. Το βάρος αυτή της ακμής αυξάνεται επίσης κατά μία μονάδα και ονομάζεται $Weight_{in}$. Η ίδια διαδικασία επαναλαμβάνεται για κάθε παράθυρο W του κειμένου. Ο αλγόριθμος που περιγράφηκε παραπάνω φαίνεται στο 4.1.

Τα παραπάνω θα γίνουν πιο κατανοητά με το παράδειγμα που ακολουθεί.

Παράδειγμα 1

Έστω το κείμενο, T1 T2 T3 T2 T4 T5 T1 T3 T5 T4 T1 T2 T4 T6. Επίσης, έστω ότι το μέγεθος του παραθύρου W_{size} ισούται με 3. Αν χωριστεί το κείμενο με το συγκεκριμένο μέγεθος παραθύρου θα προκύψουν τα παρακάτω παράθυρα W:

Λίστα Όρων	T1	T2	T3	T2	T4	T5	T1	T3	T5	T4	T1	T2	T4	T6
Δείκτης W		1			2			3			4			5

Table 4.1: Λίστα Παραθύρων

Σύμφωνα με τον αλγόριθμο που περιγράφτηκε παραπάνω, οι εξωτερικές ακμές του γραφήματος του δοθέντος κειμένου θα είναι οι εξής:

ΑΑ	ΑΚΜΗ	ΒΑΡΟΣ
1	T1,T2	2
2	T1,T3	2
3	T1,T5	1
4	T2,T3	1
5	T2,T4	2
6	T2,T5	1
7	T3,T5	1
8	T4,T1	1
9	T4,T5	1
10	T4,T6	1

Table 4.2: Πίνακας βάρους εξωτερικών ακμών νέου γραφήματος

Επειδή τα γραφήματα είναι μη κατευθυνόμενα, ακμές όπως για παράδειγμα, η T2,T4 και η T4,T2, είναι οι ίδιες και έχουν σημειωθεί στο πίνακα ως μία.

```
1 input: Document Dj, window size w
2 output: Graph Gj
3 Gj = {}
4 split Dj into parts P according to w
5 for all P in Dj:
```

```
6   for term in P:
7    for t' in P:
8         if edge(term,t') exists:
9       increment edge(term,t') by 1
10        else
11         init edge(term,t') with weight 1
12   decrease edge(term,term) by 1
```

Αλγόριθμος 4.1: Creating the graph using constant window

4.2.2 Παράθυρο σταθερού μεγέθους για κάθε κείμενο

Η μέθοδος που συζητήθηκε στη προηγούμενη ενότητα, χρησιμοποιεί σταθερό μέγεθος παραθύρου για όλη τη συλλογή. Οι περισσότερες συλλογές, ωστόσο, παρουσιάζουν σημαντική ανομοιογένειά ως προς τα μεγέθη των κειμένων, με αποτέλεσμα η χρήση ενός σταθερού παραθύρου για όλα τα κείμενα να μην είναι βέλτιστη. Συνεπώς, υπάρχει ανάγκη για την δημιουργία μίας νέας μεθόδου, όπου το μέγεθος παραθύρου W_{size} να είναι μεταβλητό για κάθε κείμενο. Ο τρόπος με τον οποίο επιτυγχάνεται αυτό είναι πολύ απλός. Το μέγεθος του παραθύρου θεωρείται ως ένα ποσοστό του μεγέθους του κειμένου (σε λέξεις) και ορίζεται σύμφωνα με το παρακάτω θεώρημα:

Θεώρημα 1

Δοθέντος ενός κειμένου D_j με μέγεθος, σε λέξεις, N_j, το μέγεθος του παραθύρου W_{size_j} για αυτό το κείμενο θα ορίζεται σύμφωνα με το παρακάτω τύπο:

$$W_{size_j} = \begin{cases} N_j \times P_{size} & \text{όπου} \quad 0 < P_{size} < 1 \text{ και } W_{size_j} > T_{size} \\ T_{size} & \text{όταν} \quad W_{size} \leq T_{size} \end{cases} \tag{4.1}$$

Το P_{size} δηλώνει το επιθυμητό ποσοστό του μεγέθους του κειμένου, το οποίο θα αποτελέσει το μέγεθος του παραθύρου και είναι μία πειραματική μεταβλητή που ορίζεται ανάλογα τα κείμενα της συλλογής. Ακόμα, υπάρχει ένα κατώτατο όριο T_{size}, με σκοπό πολύ μικρά κείμενα της συλλογής να μην παράγουν πολύ μικρά παράθυρα. Στην περίπτωση, που παράγεται ένα τέτοιο παράθυρο, το μέγεθος παραθύρου του κειμένου ισούται με το όριο T_{size}. Το όριο T_{size} αποτελεί επίσης μία πειραματική μεταβλητή και διαφέρει σε κάθε συλλογή. Όμοια με την μέθοδο σταθερού μεγέθους παραθύρου, τα παράθυρα W_j που παράγονται από ένα κείμενο D_j έχουν μέγεθος W_{size_j}, εκτός ίσως από το τελευταίο παράθυρο του κειμένου. Αυτό γίνεται όταν το W_{size_j} δεν είναι ακέραιο πολλαπλάσιο του μεγέθους του κειμένου D_j. Ο αλγόριθμος φαίνεται στο 4.2.

```
1  input: Document Dj, file percentage per
2  output: Graph Gj
3  Gj = {}
4  calculate window size w from Dj using per
5  if w <5
6     then w = 5
7  split Dj into parts P according to w
8  for all P in Dj:
9   for term in P:
10   for t' in P:
11         if edge(term,t') exists:
12       increment edge(term,t') by 1
13         else
14          init edge(term,t') with weight 1
15    decrease edge(term,term) by 1
```

Αλγόριθμος 4.2: Creating the graph using constant window based on the document size

Η διαδικασία κατασκευής των γραφημάτων κείμενου που ακολουθείται είναι η ίδια με αυτή του σταθερού μεγέθους παραθύρου για όλη τη συλλογή.

4.2.3 Παράθυρο μεγέθους πρότασης

Όπως αναφέρθηκε και στην εισαγωγή, οι λέξεις ενός κειμένου δεν σχετίζονται ισάξια μεταξύ τους. Δηλαδή, μία λέξη στην αρχή του κειμένου και μία λέξη στο τέλος του κειμένου δεν μπορούν να σχετίζονται όσο δύο λέξεις στην ίδια πρόταση ή στην ίδια παράγραφο. Μέχρι τώρα, τα παράθυρα ενός κειμένου ορίζονταν σύμφωνα με την πειραματική μεταβλητή W_{size} και δεν αντιστοιχούσαν πραγματικά στις προτάσεις του κειμένου.

Η ιδέα του μοντέλου που περιγράφεται σε αυτή την ενότητα είναι ο ορισμός των προτάσεων του κειμένου ως τα παράθυρα αυτού. Το μοντέλο θεωρεί ότι μία πρόταση ξεκινάει με έναν όρο αμέσως μετά από μία τελεία και τελειώνει με έναν όρο αμέσως πριν από μία τελεία. Το μέγεθος παραθύρου W_{size} είναι πλέον εντελώς μεταβλητό και παύει να είναι πειραματική μεταβλητή της συλλογής. Η διαδικασία της κατασκευής των γραφημάτων κειμένου είναι ίδια με της προηγούμενες μεθόδους.

4.2.4 Παράθυρο μεγέθους πρότασης και παραγράφου

Τα μοντέλα που χρησιμοποιήθηκαν ως τώρα επεξεργάζονται το κείμενο σε επίπεδο πρότασης και έχουν μέγεθος παραθύρου συνήθως μικρότερο από 30 λέξεις. Μία επέκταση είναι να γίνεται επεξεργασία του κειμένου σε επίπεδο πρότασης και ταυτόχρονα σε επίπεδο παραγράφου. Θεωρείται ότι ένα παράθυρο μεγέθους μικρότερο από 30 λέξεις αντιστοιχεί σε παράθυρο πρότασης και ένα παράθυρο μεγέθους μεγαλύτερο από 80 λέξεις αντιστοιχεί σε παράθυρο παραγράφου.

Σκοπός είναι να χωρίσουμε το κείμενο σύμφωνα με δύο παράθυρα, ένα για πρόταση

και ένα για παράγραφο, και έπειτα να συνδυάσουμε τα επιμέρους γραφήματα κειμένων που προκύπτουν. Η κατασκευή του γραφήματος κειμένου επιπέδου πρότασης πραγματοποιείτε με οποιαδήποτε μέθοδο έχει ήδη αναφερθεί. Αντίστοιχα, η κατασκευή του γραφήματος κειμένου επιπέδου παραγράφου πραγματοποιείτε με οποιαδήποτε μέθοδος είναι δυνατόν να οριστεί μέγεθος παραθύρου μεγαλύτερο από 80. Τα δύο γραφήματα θα έχουν τους ίδιους κόμβους αφού προέρχονται από το ίδιο κείμενο. Το συνολικό γράφημα θα αποτελείται από τους κόμβους και τις ακμές των δύο επιμέρους γραφημάτων. Τα συνολικά βάρη των ακμών θα υπολογίζονται σύμφωνα με το παρακάτω θεώρημα.

Θεώρημα 2

Δοθέντος μίας ακμής e, το βάρος W_e αυτής στο συνολικό γράφημα πρότασης/παραγράφου υπολογίζεται ως εξής:

$$W_e = \alpha \times W_{sentence_e} + \beta \times W_{paragraph_e} \tag{4.2}$$

Όπου $W_{sentence_e}$ είναι το βάρος της ακμής e στο γράφημα κειμένου επιπέδου πρότασης και $W_{paragraph_e}$ είναι το βάρος της ακμής e στο γράφημα κειμένου επιπέδου παραγράφου. Σε περίπτωση που δεν υπάρχει μία από τις ακμές σε ένα από τα δύο γραφήματα, το αντίστοιχο βάρος θα ισούται με 0. Τα α και β είναι μεταβλητές σημαντικότητας των επιμέρους βαρών.

4.2.5 Ολισθούμενο παράθυρο Graph of Words

Σε αυτή την ενότητα, δανείζεται ο τρόπος κατασκευής του Graph-of-word από τους (Rousseau and Vazirgiannis, 2013b). Δηλαδή, χρησιμοποιείται ένα ολισθούμενο παράθυρο μεγέθους W_{size}, το οποίο διατρέχει το κείμενο, ξεκινώντας από την αρχή μέχρι και το τέλος του. Το παράθυρο μετατοπίζεται κατά μία λέξη σε κάθε βήμα. Μέχρι τώρα, τα κείμενα χωρίζονταν σε παράθυρα τα οποία ήταν ανεξάρτητα μεταξύ τους, ενώ πλέον εισάγεται η έννοια των επικαλυπτόμενων παραθύρων. Αυτό φαίνεται καλύτερα στο παρακάτω παράδειγμα.

Παράδειγμα 2

Έστω ότι μας δίνεται το κείμενο του παραδείγματος 1, T1 T2 T3 T2 T4 T5 T1 T3 T5 T4 T1 T2 T4 T6. Επίσης, έστω ότι το W_{size} είναι ίσο με 5. Τότε το κείμενο

θα χωριστεί, όπως φαίνεται παρακάτω:

Βήμα 1	T1	T2	T3	T2	T4	T5	T1	T3	T5	T4	T1	T2	T4	T6
Βήμα 2	T1	T2	T3	T2	T4	T5	T1	T3	T5	T4	T1	T2	T4	T6
Βήμα 3	T1	T2	T3	T2	T4	T5	T1	T3	T5	T4	T1	T2	T4	T6
						. . .								
Βήμα n	T1	T2	T3	T2	T4	T5	T1	T3	T5	T4	T1	T2	T4	T6

Table 4.3: Λίστα Ολισθούμενων Παραθύρων

Αφότου χωριστεί το κείμενο σε επικαλυπτόμενα παράθυρα, η διαδικασία κατασκευής των γραφημάτων που ακολουθεί είναι ίδια με τις παραπάνω υλοποιήσεις. Μία σημαντική διαφορά της παρών υλοποίησης με το κλασσικό ολισθούμενο παράθυρο είναι ότι τα γραφήματα κειμένων που παράγονται είναι μη κατευθυνόμενα σταθμισμένα και όχι κατευθυνόμενα μη σταθμισμένα. Τέλος, το μέγεθος παραθύρου W_{size} αποτελεί πειραματική μεταβλητή και είναι διαφορετική για κάθε συλλογή.

4.2.6 Ποινή στο συνολικό γράφημα συλλογής

Ως τώρα έχει γίνει αναφορά κυρίως σε μοντέλα που αφορούν την κατασκευή των γραφημάτων κειμένων της συλλογής. Επειδή, τα μοντέλα μας ακολουθούν τον αλγόριθμο GSB, κάποια στιγμή πρέπει δημιουργηθεί το συνολικό γράφημα συλλογής. Σε αυτή την ενότητα, προτείνεται ένα νέο είδος ποινής στα βάρη των κόμβων κατά την κατασκευή του συνολικού γραφήματος. Η διαδικασία που ακολουθείται είναι πολύ απλή.

Κατά την κατασκευή του συνολικού γραφήματος δίνεται ποινή, με την μορφή ποσοστιαίας μείωσης στα βάρη όλων των ακμών που εισάγονται στο γράφημα. Αυτό έχει σαν αποτέλεσμα την μείωση όλων των βαρών W_{out} των κόμβων. Εξαιτίας του λογαρίθμου που περιλαμβάνεται στη εξίσωση 3.7, τα βάρη των πιο σπάνιων κόμβων θα μειωθούν, αναλογικά, περισσότερο σε σχέση με τα βάρη των πιο συχνών κόμβων. Το παραπάνω είναι μία μορφή κανονικοποίησης των βαρών των κόμβων του συνολικού γραφήματος. Στο κεφάλαιο των αποτελεσμάτων θα φανεί ότι η συγκεκριμένη μορφή κανονικοποίησης βελτιώνει, έστω και λίγο, τα αποτελέσματα. Ο αρχικός σκοπός της κανονικοποίησης αυτής ήταν η μελέτη της επίδρασης των πιο συχνών όρων, που επί το πλείστων είναι ασήμαντες λέξεις (5), στην διαδικασία της ανάκτησης. Η βελτίωση των αποτελεσμάτων υποθέτει ότι οι ασήμαντες λέξεις βοηθάνε, μάλλον, την διαδικασία της ανάκτησης. Αυτό μπορεί να οφείλεται στο γεγονός ότι η δομή των κειμένων

αποτυπώνεται καλύτερα στα γραφήματα κειμένων όταν συμπεριλαμβάνονται και οι ασήμαντες λέξεις μαζί με τους υπόλοιπους όρους.

4.2.7 K-core decomposition

Όπως και στο GSB, έτσι και εδώ, είναι δυνατή η εφαρμογή τεχνικών ανάλυσης γραφήματος σε επίπεδα. Κατά την δημιουργία του συνολικού γραφήματος της συλλογής, εντοπίζεται τα διάφορα επίπεδα πυρήνων. Μία σημαντική παρατήρηση, είναι ότι, στο GSB, επειδή τα γραφήματα κειμένων είναι πλήρες, το $k-core\ decomposition$ αποτελείται μόνο από ένα επίπεδο, το οποίο είναι και το $maincore$. Συνεπώς, χρειάζεται μία διαδικασία απαλοιφής ακμών που δεν περιέχουν πολύ πληροφορία. Στα γραφήματα κειμένων, που παράγονται από τις προτεινόμενες μεθόδους ωστόσο αυτό το πρόβλημα δεν υπάρχει αφού όπως έχει αναφερθεί ήδη, όχι μόνο δεν είναι πλήρες, αλλά είναι και πιο συνεκτικά και νοηματικά ισχυρά. Το ίδιο ισχύει και για παραλλαγές στην μέθοδο του maincore, όπως η density και η corerank. Αφού γίνει η ανάλυση σε επίπεδα, τα βάρη των ακμών των σημαντικών κόμβων που εξάγονται, προσαυξάνονται κατά μία θετική τιμή. Έπειτα, η διαδικασία συνεχίζεται κανονικά με την δημιουργία των ευρετηρίων. Για πειραματικούς λόγους, έγιναν και πειράματα χρησιμοποιώντας την διαδικασία απαλοιφής ακμών του GSB. Τα αποτελέσματα ήταν και στις δύο περιπτώσεις, σημαντικά καλύτερά από αυτά του GSB και θα συζητηθούν περαιτέρω στο αντίστοιχο κεφάλαιο.

4.3 Ανάκτηση πληροφορίας στα μοντέλα

Η διαδικασία ανάκτησης πληροφορίας στα μοντέλα είναι ίδια με την διαδικασία που περιγράφηκε στο κεφάλαιο για το GSB (3.1.3). Αυτό είναι δυνατόν γιατί τα μοντέλα απλά αλλάζουν τον τρόπο δημιουργίας του συνολικού γραφήματος και όχι τον τρόπο που δημιουργούνται και χρησιμοποιούνται τα βάρη των όρων από το συνολικό γράφημα. θα περιγραφεί παρακάτω συνοπτικά η διαδικασία ξανά.

Αρχικά, παράγονται τα σύνολα όρων με την βοήθεια του αλγόριθμου apriori και συνδυάζονται ανά δύο τα σύνολα που διαφέρουν μόνο κατά ένα όρο. Για κάθε σύνολο υπολογίζεται η μετρική TN_{S_i}, η οποία ισούται με το γινόμενο των βαρών του γραφήματος για κάθε όρο του συνόλου. Στην συνέχεια, συμπεριλαμβάνεται αυτή η μετρική στο βάρος που περιγράφεται από το Set-Based μοντέλο ($TF \times IDF$). Έπειτα, υπολογίζεται το βάρος του ερωτήματος.

Τέλος, τα κείμενα και τα ερωτήματα αναπαριστώνται ως διανύσματα, και υπολογίζεται η ομοιότητα του συνημίτονου. Τα κείμενα ανακτώνται σε φθίνουσα σειρά σύμφωνα με την ομοιότητα τους με το ερώτημα.

4.4 Εναλλακτικές επιλογές που απέτυχαν

Στο υποκεφάλαιο 5.3.1, κάνουμε την υπόθεση ότι ο κύριος πυρήνας(maincore) των γραφημάτων των κειμένων, αποτελείται κυρίως από stopwords. Ακολουθώντας αυτή την λογική, έγινε εφαρμογή ποινής, αντί μπόνους στους κόμβους του maincore, ελπίζοντας ότι τα αποτελέσματα κατά την ανάκτηση θα βελτιωθούν, αφού πλέον η επίδραση των stopwords θα είναι μικρότερη. Τα αποτελέσματα ωστόσο χειροτέρευσαν, το οποίο μας οδηγεί στον ισχυρισμό ότι η αρχική μας υπόθεση ήταν λανθασμένη και ότι ο κύριος πυρήνας αποτελείτε και από keywords, καθώς και λέξεις που δεν ανήκουν σε κάποια κατηγορία. Στο κεφάλαιο που προαναφέρθηκε πριν, έγινε ανάλυση του maincore και αποδείχθηκε ο ισχυρισμός ότι ο κύριος πυρήνας περιέχει stopwords, keywords και άλλες τυχαίες λέξεις.

Μία άλλη εναλλακτική προσέγγιση ήταν ότι, κατά την προ-επεξεργασία της συλλογής, αφαιρέθηκαν τα stopwords και εφαρμόστηκαν οι προτεινόμενες μέθοδοι. Τα αποτελέσματα που προέκυψαν κατά την ανάκτηση ήταν σημαντικά χειρότερα, σε σχέση με τα αποτελέσματα όταν διατηρούνται τα stopwords. Συμπεραίνεται ότι η παρουσία των stopwords τελικά βοηθάει την διαδικασία της ανάκτησης. Μία εξήγηση είναι, ότι όταν τα stopwords αφαιρούνται τα κείμενα της συλλογής χάνουν ένα σημαντικό κομμάτι της νοηματικής τους αξίας, το οποίο μεταφράζεται σαν πιο συνεκτικά αδύναμα γραφήματα κειμένων. Επιπρόσθετα, σύμφωνα με τη διαδικασία που αναλύθηκε στο κεφάλαιο 4.2.6, ενώ η παρουσία των stopwords οδηγεί σε πιο συνεκτικά ισχυρά και συμπαγή γραφήματα, είναι προς όφελος μας τα βάρη των κόμβων που αντιστοιχούν σε stopwords να είναι χαμηλά. Οι ακμές των κόμβων που αντιστοιχούν σε stopwords φαίνεται να λειτουργούν ως γέφυρες μεταξύ διάφορων υπογραφημάτων που αποτελούν τα γραφήματα κειμένων. Περισσότερα για τα αποτελέσματα θα συζητηθούν στο κεφάλαιο 6.

4.5 Περιγραφή υλοποίησης

4.5.1 Εισαγωγή

Σε αυτό το υποκεφάλαιο θα συζητηθεί πιο λεπτομερώς η υλοποίηση των παραπάνω προτεινομένων μεθόδων. Υλοποιήθηκαν διάφορα συστήματα προ-επεξεργασίας των τεσσάρων συλλογών ((Shaw et al., 1991), (*Test collections* n.d.)) που χρησιμοποιήθηκαν, ένα σύστημα κατασκευής γραφημάτων και παραγωγής ανεστραμμένων ευρετηρίων από αυτές για τις προτεινόμενες μεθόδους, καθώς και ένα σύστημα υλοποίησης του Set-Based μοντέλου του οποίου τα αποτελέσματα χρησιμοποιήθηκαν συγκριτικά για την αξιολόγηση των προτεινόμενων μεθόδων.

Αρχικά θα αναφερθούν μερικά πιο τεχνικά κομμάτια της υλοποίησης. Το λειτουργικό σύστημα που χρησιμοποιήθηκε ήταν το Windows 10 της Microsoft. Σαν γλώσσα προγραμματισμού επιλέχθηκε η Python, λόγου της ευελιξίας της, καθώς και του μεγάλου όγκου βιβλιοθηκών που είναι διαθέσιμες. Είναι μία υψηλού επιπέδου διερμηνευόμενη γλώσσα, οι οποία δίνει έμφαση στην εύκολη αναγνωσιμότητα. Η python έχει την δυνατότητα ανάπτυξης διαδικαστικών και αντικειμενοστραφών εφαρμογών. Το περιβάλλον ανάπτυξης του κώδικα που χρησιμοποιήθηκε ήταν το Notepad++ που αναπτύχθηκε από την εταιρία Don Ho. Παρακάτω ακολουθούν οι βιβλιοθήκες που χρησιμοποιήθηκαν:

```
import sys
import time
import os

import csv
import pandas as pd
import numpy

import string
import math
from math import log

import nltk
from nltk.corpus import stopwords
import networkx as nx
from networkx import core_number, k_core

import matplotlib.pyplot as plt
from openpyxl import load_workbook
```

Αλγόριθμος 4.3: Python Libraries Used.

Η βιβλιοθήκη sys χρησιμοποιήθηκε για την πρόσβαση που παρέχει σε διάφορες μεταβλητές του διερμηνευτή της Python. Η time χρησιμοποιήθηκε για την ακριβή μέτρηση του χρόνο που χρειάζεται η ολοκλήρωση μίας εκτέλεσης του προγράμματος. Η os χρησιμοποιήθηκε για τις δυνατότητες που παρέχει στην επεξεργασία διαδρομών και προσπέλασης διάφορων αρχείων του συστήματος αρχείων.

Η βιβλιοθήκη Natural Language Toolkit (nltk) χρησιμοποιήθηκε για το διαχωρισμό των κειμένων σε προτάσεις, όπου αυτό ήταν απαραίτητο. Επίσης, από την nltk χρησιμοποιήθηκε η λίστα των αγγλικών ασήμαντων λέξεων για τις συλλογές που δεν παρείχαν δικιά τους λίστα ασήμαντων λέξεων.

Η βιβλιοθήκη networkx χρησιμοποιήθηκε για την δημιουργία και επεξεργασία των διαφόρων γραφημάτων, καθώς και για την ανάλυση αυτών σε επίπεδα πυρήνων. Η matplotlib χρησιμοποιήθηκε σε συνδυασμό με την networkx για την απεικόνιση των γραφημάτων.

Η math χρησιμοποιήθηκε για διάφορους αριθμητικούς υπολογισμούς, η string για επεξεργασία αλφαριθμητικών όπου αυτό ήταν αναγκαίο, και η numpy για την δημιουργία και την επεξεργασία μητρώων.

Τέλος, με την βοήθεια των pandas και openyxl, τα αποτελέσματα αποθηκεύτηκαν στο δίσκο σε ένα βιβλίο του microsoft Excel για περαιτέρω επεξεργασία. Επίσης, η matplotlib βοήθησε στο σχεδιασμό διαγραμμάτων ανάκλησης - ακρίβειας για κάθε ερώτημα.

Η υλοποίηση έχει δύο βασικές καταστάσεις λειτουργίας. Στην πρώτη κατάσταση παράγονται τα αναστραμμένα ευρετήρια σύμφωνα με τις προτεινόμενες μεθόδους, ενώ στην δεύτερή, αυτά, εισάγονται σαν είσοδο και υπολογίζονται τα αντίστοιχα βάρη. Έπειτα, υλοποιείται το Set-Based μοντέλο και δημιουργούνται τα γραφήματα για κάθε ερώτημα της συλλογής. Τέλος, τα αποτελέσματα των μεθόδων καταγράφονται σε ένα αρχείο Excel, όπως έχει ήδη αναφερθεί. Συνολικά παράγονται έξι ευρετήρια, τα οποία αντιστοιχούν σε πέντε προτεινόμενες μεθόδους (ανάλογα ποιες εξετάζονται στο παρών πείραμα) και στο GSB(Kalogeropoulos et al., 2020), για λόγους σύγκρισης. Τόσο η επιλογή της κατάστασης, όσο και η επιλογή των παραμέτρων για τις μεθόδους γίνεται με εισαγωγή κάποιων μεταβλητών από μία γραμμή εντολών, παράλληλα με την εκκίνησή του προγράμματος. Σημειώνεται επίσης ότι για να λειτουργήσει σωστά η δεύτερη κατάσταση λειτουργίας είναι απαραίτητο να υπάρχουν στον ίδιο κατάλογο αρχείων τα κατάλληλα ευρετήρια που παράγονται από την πρώτη κατάσταση.

4.5.2 Δημιουργία των γραφημάτων κειμένων.

Σε αυτό το υπόκεφάλαιο θα γίνει ανάλυση των τρόπων με τους οποίους χωρίζονται τα αρχεία μίας συλλογής με την χρήση παραθύρων. Σε όλες τις μεθόδους σαν είσοδο εισάγεται ένα αρχείο της συλλογής και σαν έξοδο και σαν έξοδο λαμβάνεται ένας πίνακας γειτνίασης που περιγράφει το γράφημα κειμένου της εισόδου.

Πριν την εκτέλεση των συναρτήσεων για κάθε μέθοδο, εκτελείται πρώτα μία συνάρτηση που ονομάζεται createInvertedIndexFromFile και δέχεται σαν είσοδο ένα αρχείο και μία λίστα που περιλαμβάνει τους όρους της συλλογής (posting list). Γίνεται ανάλυση του αρχείου και επιστρέφονται λίστες που περιλαμβάνουν τους όρους του κειμένου μαζί με τις συχνότητες εμφάνισης τους, το μέγεθος του κειμένου σε λέξεις και μία ανανεωμένη έκδοση της λίστας με τους όρους της συλλογής που αναφέρθηκε πρωτύτερα.

```
def createInvertedIndexFromFile(file, postingl):
    with open(file, 'r') as fd:
    # list containing every word in text document
        text = fd.read().split()
        uninque_terms = []
        termFreq = []
        for term in text:
            if term not in uninque_terms:
                uninque_terms.append(term)
                termFreq.append(text.count(term))
            if term not in postingl:
                postingl.append(term)
                postingl.append([file, text.count(term)])
            else:
                existingtermindex = postingl.index(term)
                if file not in postingl[existingtermindex + 1]:
                    postingl[existingtermindex + 1].extend([file, text.
                        count(term)])
    return (uninque_terms, termFreq, postingl, len(text))
```

Αλγόριθμος 4.4: File Analysis Algorithm

Παράθυρο σταθερού μεγέθους

Αρχικά, πριν δημιουργηθεί ο πίνακας γειτνίασης πρέπει να χωριστεί το αρχείο σε κομμάτια σύμφωνα με το μέγεθος παραθύρου W_{size}. Αυτό επιτυγχάνεται με την χρήση μίας συνάρτησης, που ονομάζεται splitFileConstantWindow, και δέχεται σαν είσοδο ένα κείμενο της συλλογής και το μέγεθος του παραθύρου με το οποίο θα χωριστεί το κείμενο. Στην συνέχεια χωρίζεται το αρχείο σε λέξεις και ενώνονται αυτές σε αλφαριθμητικά μεγέθους W_{size}. Στην διαδικασία αυτή διατηρείται η σειρά με την οποία εμφανίζονται οι λέξεις στο κείμενο. Τέλος, επιστρέφεται ένας πίνακας, ο οποίος περιέχει τα αλφαριθμητικά παραθύρου που δημιουργήθηκαν παραπάνω.

```
def splitFileConstantWindow(file,window):
    #Open the file and split it into words
    inputFile = open(file, 'r').read().split()
    num_of_words = len(inputFile)
    outputFile =[]
    #Join words according to window
    for i in range(0,num_of_words,window):
        outputFile.append(' '.join(inputFile[i:i+window]))
    #print(outputData)
    return outputFile
```

Αλγόριθμος 4.5: File Splitting Algorithm with Constant Window Size

Έπειτα, χρησιμοποιείται μία συνάρτηση που ονομάζεται CreateAdjMatrixFromInvIndexWithWindow και δέχεται σαν είσοδο τους όρους του αρχείου, το κείμενο του αρχείου και το μέγεθος του παραθύρου. Αυτή καλεί την splitFileConstantWindow και για κάθε αλφαριθμητικό παραθύρου, εξάγει τους όρους που το απαρτίζουν και μετράει τα ζεύγη εμφανίσεων και κατασκευάζει το πίνακα γειτνίασης. Τέλος, επιστρέφει τον πίνακα γειτνίασης.

```
#First we split the file into an array, in which each element
#represents a part of the file with length equal to window_size. We
#then take each part and create the adjacency matrix. Each element
#a(i,j), where i,j are two different terms of the file, is equal to the
#number of times the term i and the term j appear in the same window.
def CreateAdjMatrixFromInvIndexWithWindow(terms, file, window_size):
    adj_matrix = numpy.zeros(shape=(len(terms),len(terms)))
    split_file = splitFileConstantWindow(file, window_size)
    for subfile in split_file:
        window_terms = subfile.split()
        for term in window_terms:
            row_index = terms.index(term)
            for x in range(0,len(window_terms)):
                col_index = terms.index(window_terms[x])
                adj_matrix[row_index][col_index]+=1
            adj_matrix[row_index][row_index]-=1
    return (adj_matrix)
```

Αλγόριθμος 4.6: Adjacency Matrix Creation Algorithm with Constant Window Size

Παράθυρο σταθερού μεγέθους για κάθε κείμενο

Η διαδικασία για την κατασκευή του πίνακα γειτνίασης όταν χρησιμοποιούνται παράθυρα διαφορετικά για κάθε κείμενο είναι παρόμοια με αυτή του σταθερού παραθύρου. Η μόνη διαφορά είναι ότι κατά τον διαχωρισμό του κειμένου σε παράθυρα, το μέγεθος του παραθύρου υπολογίζεται με βάση το ποσοστό του κειμένου. Οπότε, χρησιμοποιείται μία επέκταση της συνάρτησης με όνομα splitFileConstantWindow στην οποία όταν το μέγεθος του παραθύρου W_{size} που εισάγεται είναι ίσο με μηδέν, χρησιμοποιεί μία καινούρια μεταβλητή για να υπολογισθεί το νέο μέγεθος του κειμένου. Αυτή η μεταβλητή ονομάζεται per_window και αντιπροσωπεύει το ποσοστό του κειμέ-

νου που θα χρησιμοποιηθεί σαν μέγεθος παραθύρου. Όπως έχει αναφερθεί ήδη και στο αντίστοιχο υπο-κεφάλαιο(4.2.2) υπάρχει ένα κατώφλι για το μέγεθος παραθύρου που στα πειράματά που εκτελέστηκαν ήταν ίσο με πέντε.

```
def splitFileConstantWindow(file,window,per_window):
    #Open the file and split it into words
    inputFile = open(file, 'r').read().split()
    num_of_words = len(inputFile)
    outputFile =[]
    #If window is equal to zero, calculate window size according to
        file length
    if window == 0:
        window = int(num_of_words * per_window) + 1
        if window <5:
            window =5
    #Join words according to window
    for i in range(0,num_of_words,window):
        outputFile.append(' '.join(inputFile[i:i+window]))
    return outputFile
```

Αλγόριθμος 4.7: Extension of the splitFileConstantWindow Algorithm

Όπως και στην μέθοδο για το σταθερό μέγεθος παραθύρου, η splitFileConstantWindow καλείται από την CreateAdjMatrixFromInvIndexWithWindow και αφού χωριστεί το κείμενο σε παράθυρα, κατασκευάζεται ο πίνακας γειτνίασης. Έχουν γίνει κατάλληλες αλλαγές για να συμπεριληφθεί και η μεταβλητή per_window σαν είσοδο στην συνάρτηση.

```
#First we split the file into an array, in which each element
#represents a part of the file with length equal to window_size. We
#then take each part and create the adjacency matrix. Each element
#a(i,j), where i,j are two different terms of the file, is equal to the
#number of times the term i and the term j appear in the same window.
def CreateAdjMatrixFromInvIndexWithWindow(terms, file, window_size,
    per_window):
    adj_matrix = numpy.zeros(shape=(len(terms),len(terms)))
    split_file = splitFileConstantWindow(file, window_size,per_window)
    for subfile in split_file:
        window_terms = subfile.split()
        for term in window_terms:
            row_index = terms.index(term)
            for x in range(0,len(window_terms)):
                col_index = terms.index(window_terms[x])
                adj_matrix[row_index][col_index]+=1
            adj_matrix[row_index][row_index]-=1
    return (adj_matrix)
```

Αλγόριθμος 4.8: Adjacency Matrix Creation with Constant/Percent Window Size

Παράθυρο μεγέθους πρότασης

Όταν χρησιμοποιείται παράθυρο μεγέθους πρότασης, για κάθε κείμενο εντοπίζονται οι προτάσεις που το απαρτίζουν και έπειτα ορίζονται αυτές ως τα παράθυρα του κειμένου. Μετά, η διαδικασία που ακολουθεί είναι η μέτρηση των εμφανίσεων κάθε

ζευγαριού όρων και η κατασκευή του πίνακα γειτνίασης, όπως δηλαδή και στις προηγούμενες μεθόδους που αναφέρθηκαν.

Για την υλοποίηση χρησιμοποιήθηκε, όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου, η βιβλιοθήκη nltk. Πιο συγκεκριμένα χρησιμοποιήθηκε από την υποβιβλιοθήκη tokenize, η συνάρτηση sent_tokenize, η οποία δέχεται σαν είσοδο ένα αλφαριθμητικό κειμένου και ένα αλφαριθμητικό που δηλώνει την γλώσσα στην οποία είναι γραμμένο το κείμενο. Επιστρέφει ένα πίνακα αλφαριθμητικών, ο οποίος περιέχει τις προτάσεις του κειμένου όπως αυτές ορίστηκαν από τον sentence tokenizer της nltk. Την παρούσα περίοδο που γράφτηκε αυτή η διπλωματική χρησιμοποιόταν ο PunktSentenceTokenizer σαν sentence tokenizer. Παρόμοια με την μέθοδο για παράθυρο σταθερού μεγέθους για κάθε κείμενο, εδώ επεκτείνεται η συνάρτηση CreateAdjMatrixFromInvIndexWithWindow, ώστε να μπορεί να υλοποιήσει και την παρούσα μέθοδο. Δημιουργήθηκε μία μεταβλητή "σημαία"(Flag) που ονομάζεται dot_split και ορίζεται ίση με αληθείς,αν επιλεχθεί η τρέχον μέθοδο κατά την επιλογή της μεθόδου που θα εφαρμοστεί στο παρόν πείραμα.

```
#First we split the file into an array, in which each element
#represents a part of the file with length equal to window_size. We
#then take each part and create the adjacency matrix. Each element
#a(i,j), where i,j are two different terms of the file, is equal to the
#number of times the term i and the term j appear in the same window.
def CreateAdjMatrixFromInvIndexWithWindow(terms, file, window_size,
    per_window, dot_split):
    adj_matrix = numpy.zeros(shape=(len(terms),len(terms)))
    if dot_split:
        inputFile = open(file, 'r').read()
        split_file = nltk.tokenize.sent_tokenize(inputFile, language='
            english')
    else:
        split_file = splitFileConstantWindow(file, window_size,
            per_window)
    for subfile in split_file:
        window_terms = subfile.split()
        for term in window_terms:
            row_index = terms.index(term)
            for x in range(0,len(window_terms)):
                col_index = terms.index(window_terms[x])
                adj_matrix[row_index][col_index]+=1
            adj_matrix[row_index][row_index]-=1
    return (adj_matrix)
```

Αλγόριθμος 4.9: Adjacency Matrix Creation w/ Constant/Percent/Sentence Window Size

Παράθυρο μεγέθους πρότασης και παραγράφου

Στο υποκεφάλαιο 4.2.4 εξηγήθηκε η διαδικασία κατασκευής ενός συνολικού γραφήματος κειμένου, που παράγεται από δύο επιμέρους γραφήματα, ένα που αντιστοιχεί στην ανάλυση του κειμένου σε επίπεδο πρότασης και ένα που αντιστοιχεί στην ανάλυση του κειμένου σε επίπεδο παραγράφου. Το γράφημα κειμένου επιπέδου πρότασης, είναι δυνατόν να κατασκευαστεί από οποιαδήποτε μέθοδο έχει ήδη αναφερθεί. Αντίστοιχα, το γράφημα κειμένου επιπέδου παραγράφου, είναι δυνατόν να κατασκευαστεί από οποιαδήποτε μέθοδο έχει αναφερθεί, εκτός από την μέθοδο με παράθυρο μεγέθους πρότασης.

Λαμβάνοντας τα παραπάνω υπόψιν, η υλοποίηση είναι εξαιρετικά απλή. Δημιουργήθηκε μία συνάρτηση, που ονομάζεται CreateAdjMatrixFromInvIndexWithSenParWindow, και δέχεται σαν είσοδο μία λίστα με τους όρους του κειμένου, το ίδιο το κείμενο, το μέγεθος παραθύρου πρότασης, το μέγεθος παραθύρου παραγράφου, καθώς και τη μεταβλητή σημαία dot_split. Η δουλεία της συνάρτησης είναι να καλέσει δύο φορές την CreateAdjMatrixFromInvIndexWithWindow που έχει αναφερθεί παραπάνω. Την πρώτη φορά της δίνει σαν είσοδο το παράθυρο που αντιστοιχεί σε πρόταση, ενώ την δεύτερη αυτό που αντιστοιχεί στην παράγραφο. Έπειτα, συνδυάζει τους δύο περιστρεφόμενους πίνακες γειτνίασης σε έναν συνολικό πίνακα χρησιμοποιώντας δύο μεταβλητές α,β ως μεταβλητές σημαντικότητας. Τέλος, επιστρέφει τον τελικό πίνακα.

```
def CreateAdjMatrixFromInvIndexWithSenParWindow(terms, file,
    sen_window_size, par_window_size, dot_split):
        matrix_size = len(terms)
        #Create the matrices
        sen_adj_matrix = numpy.zeros(shape=(matrix_size, matrix_size,))
        par_adj_matrix = numpy.zeros(shape=(matrix_size, matrix_size))
        #Get the adjacency matrix for each window
        sen_adj_matrix = CreateAdjMatrixFromInvIndexWithWindow(terms,
            file, sen_window_size,0, dot_split)
        par_adj_matrix = CreateAdjMatrixFromInvIndexWithWindow(terms,
            file, par_window_size,0, false)
        #Create the final Matrix
        final_adj_matrix = numpy.zeros(shape=(matrix_size, matrix_size))
        #Create coefficients a and b
        a = 1.0
        b = 0.05
        #Add the two matrices
        final_adj_matrix = [[a*sen_adj_matrix[r][c] + b*par_adj_matrix[
            r][c]  for c in range(len(sen_adj_matrix[0]))] for r in
            range(matrix_size)]
        return final_adj_matrix
```

Αλγόριθμος 4.10: Adjacency Matrix Creation w/ Sentence-Paragraph Window Size

Ολισθούμενο παράθυρο Vazirgianni/Rousseau (Rousseau and Vazirgiannis, 2013b)

Το ολισθούμενο παραθύρο των Vazirgianni/Rousseau υλοποιήθηκε με την χρήση μίας συνάρτησης που ονομάζεται CreateAdjMatrixFromInvIndexWitSlidinghWindow και δέχεται ως είσοδο μία λίστα με τους όρους ενός αρχείου, το ίδιο το αρχείο και το επιθυμητό μέγεθος παραθύρου W_{size}. Η διαδικασία που ακολουθήθηκε είναι η εξής. Αρχικά, χωρίζεται το κείμενο σε όρους. Έπειτα ξεκινώντας από τον πρώτο όρο, υπολογίζονται οι εμφανίσεις των ζευγαριών των όρων σε μέγεθος ίσο με το W_{size} και ανανεώνεται ο πίνακας γειτνίασης. Στην συνέχεια, μετατοπίζεται το παράθυρο κατά μία θέση δεξιά, δηλαδή προς το τέλος του κειμένου, και επαναλαμβάνεται η διαδικασία. Την θέση λέξης στο κείμενο την ορίζει μία μεταβλητή μέτρησης που ονομάζεται counter. Τέλος, επιστρέφεται ο τελικός πίνακας γειτνίασης.

```
def CreateAdjMatrixFromInvIndexWithWindow(terms, file, window_size):
    adj_matrix = numpy.zeros(shape=(len(terms),len(terms)))
    split_file = open(file, 'r').read().split()
    counter = 0
    for term in split_file:
        row_index = terms.index(term)
        for x in range(0,window_size):
            try:
                col_index = terms.index(split_file[counter + x])
                adj_matrix[row_index][col_index]+=1
            except IndexError:
                break
        counter+=1
        adj_matrix[row_index][row_index]-=1
    return (adj_matrix)
```

Αλγόριθμος 4.11: Adjacency Matrix Creation w/ Sliding Window

Graphical Set-Based Model (GSB)

Σε αυτό το σημείο θα γίνει αναφορά στο τρόπο που υλοποιήθηκε η δημιουργία γραφημάτων κειμένων σύμφωνα με το GSB. Χρησιμοποιήθηκε μία συνάρτηση που ονομάζεται CreateAdjMatrixFromInvIndex και δέχεται σαν είσοδο μία λίστα που περιλαμβάνει τους όρους του κειμένου (terms), καθώς και μία λίστα που περιλαμβάνει την συχνότητα εμφάνισης αυτών (tf). Όπως έχει ήδη αναφερθεί στο κεφάλαιο 3.1.3, το GSB παράγει πλήρη γραφήματα κειμένων γιατί υποθέτει ότι κάθε όρος σχετίζεται ισάξια με κάθε άλλο όρο του κειμένου. Σύμφωνα με αυτό, είναι δυνατόν να υπολογισθεί το βάρος όλων των έξω ακμών ως το εσωτερικό γινόμενο δύο διανυσμάτων $\vec{u}, \vec{p}$, παράγοντας ένα πίνακα A, μεγέθους $n \times n$, όπου σε κάθε θέση A_{ij} θα περιέχεται το γινόμενο $\vec{u_i} \cdot \vec{p_j}$. Έπειτα, για τον υπολογισμό των εσωτερικών ακμών, χρησιμοποιήθηκε μία δομή επανάληψης, η οποία διατρέχει τον παραπάνω πίνακα και υπολογίζει την διαγώνιο, η οποία αντιστοιχεί στις εσωτερικές ακμές κάθε όρου. Τέλος, επιστρέφεται ο

τελικός πίνακας A ως πίνακας γειτνίασης του γραφήματος του κειμένου.

```
def CreateAdjMatrixFromInvIndex(terms, tf):
    rows = numpy.array(tf)
    row = numpy.transpose(rows.reshape(1, len(rows)))
    col = numpy.transpose(rows.reshape(len(rows), 1))
    adj_matrix = numpy.array(numpy.dot(row, col))
    for i in range(len(adj_matrix)):
        for j in range(len(adj_matrix)):
            if i == j:
                adj_matrix[i][j] = rows[i] * (rows[i] + 1) * 0.5
    del row, rows, col
    return (adj_matrix)
```

Αλγόριθμος 4.12: Adjacency Matrix Creation using GSB

Δημιουργία γραφήματος κειμένου από πίνακα γειτνίασης.

Όλες οι παραπάνω συναρτήσεις δημιουργούν έναν πίνακα γειτνίασης, ο οποίος περιγράφει το γράφημα ενός κειμένου. Προαιρετικά, σε αυτό το σημείο, ο αλγόριθμος μπορεί να μεταφράσει τους πίνακες αυτούς σε γραφήματα. Αυτό το στάδιο είναι προαιρετικό και βοηθάει στην περίπτωση που για το τρέχον πείραμα εφαρμόζονται μέθοδοι που λαμβάνουν υπόψιν σημαντικούς κόμβους και δεν γίνεται απαλοιφή στις ακμές του γραφήματος κειμένου .

Αυτό επιτυγχάνεται με την χρήση μίας συνάρτησης, που ονομάζεται graphUsingAdjMatrix και δέχεται ως είσοδο τον πίνακα γειτνίασης(adjmatrix) για το τρέχον κείμενο, όπως αυτός παράχθηκε από τις μεθόδους που έχουν ήδη αναφερθεί, καθώς και μία λίστα με τους όρους του κειμένου(termlist). Δημιουργείται αρχικά ένα κενό γράφημα(gr) με την βοήθεια της βιβλιοθήκης networkx. Στην συνέχεια, διατρέχεται ο πίνακας γειτνίασης, με την χρήση μίας δομής επανάληψης και προσθέτονται στο γράφημα gr, οι όροι του κειμένου ως κόμβοι. Με την βοήθεια μίας ακόμα δομής επανάληψης προσθέτονται στο γράφημα gr και οι ακμές, μαζί με τα βάρη τους, όπως αυτά περιγράφονται από τον πίνακα γειτνίασης.Όπως, είναι προφανές το παραγόμενο γράφημα περιλαμβάνει μόνο τις έξω-ακμές σε αυτό το σημείο και επιλέχθηκε αυτός ο τρόπος ώστε να αποφευχθούν προβλήματα κατά την εφαρμογή διάφορων αλγορίθμων στο μέλλον της εκτέλεσης.

```
def graphUsingAdjMatrix(adjmatrix, termlist, *args, **kwargs):
    gr = nx.Graph()
    for i in range(0, len(adjmatrix)):
        gr.add_node(i, term=termlist[i])
        for j in range(len(adjmatrix)):
            if i > j:
                if adjmatrix[i][j]!=0:
                    gr.add_edge(i, j, weight=adjmatrix[i][j])
    # graphToPng(gr,filename = filename)
    return gr
```

Αλγόριθμος 4.13: Graph creation from adjacency matrix

Ενημέρωση του αρχείου docinfo

Σε αυτό το σημείο προσθέτεται στο αρχείο docinfo το όνομα του κειμένου και τον αριθμό των λέξεων που περιέχει. Το αρχείο docinfo είναι ένα βοηθητικό αρχείο που περιέχει πληροφορίες για τα κείμενα και χρησιμοποιείται για την μείωση της πολυπλοκότητας της ανάκτησης στο δεύτερο μέρος, αφού χάρις αυτό αποφεύγεται ο επανυπολογισμός του μεγέθους των κειμένων.

4.5.3 Εντοπισμός σημαντικών κόμβων γραφήματος κειμένου

Αφού ολοκληρωθούν τα παραπάνω, ανάλογα των μεθόδων που δοκιμάζονται στο παρόν πείραμα, ο αλγόριθμος περνάει προαιρετικά στο στάδιο εντοπισμού των σημαντικών κόμβων των γραφημάτων κειμένων. Ο εντοπισμός σημαντικών κόμβων είναι απαραίτητος για τις μεθόδους που χρησιμοποιούν MainCore, Density και CoreRank.

Απαλοιφή Ακμών στο γράφημα κειμένου

Πριν ο αλγόριθμος περάσει στο στάδιο εντοπισμού σημαντικών κόμβων, ωστόσο, προαιρετικά, εκτελείτε απαλοιφή ακμών στα γραφήματα. Αυτό επιτυγχάνθηκε με την χρήση μίας συνάρτησης, που ονομάζεται pruneGraphbyWeight και δέχεται ως είσοδο τον πίνακα γειτνίασης του γραφήματος κειμένου και μία λίστα με τους όρους του κειμένου. Η απαλοιφή γίνεται με βάση το μέσο βάρος των ακμών του γραφήματος. Η διαδικασία που ακολουθήθηκε είναι παρόμοια με την δημιουργία του γραφήματος κειμένου από τον πίνακα γειτνίασης, όπως περιγράφηκε στο 4.5.2. Η μόνη διαφορά είναι ότι γίνεται έλεγχος του βάρους κάθε ακμής και εισάγονται στο γράφημα, μόνο οι ακμές, των οποίων το βάρος είναι μεγαλύτερο απο το μέσο όρο.

```
def pruneGraphbyWeight(aMatrix,termlist):
    temp = Woutdegree(aMatrix) #[list of weight sums of each node]
    maxval = (sum(temp[0]) / (len(temp[0]) * len(temp[0]))) #avarage
        weight of node
    #maxval = (maxval * (+0.5)) + maxval #average weight of
    gr = nx.Graph()
    for i in range(len(aMatrix)):
        gr.add_node(i, term=termlist[i])
        for j in range(len(aMatrix)):
            if i > j:
                if aMatrix[i][j] >=  S * maxval: #S is the persentage
                    of the allowed weight based on maxval
                    gr.add_edge(i, j, weight=aMatrix[i][j])
                elif maxval < 1:
                    gr.add_edge(i, j, weight=1)#used because we had
                        implemented a precentange on average
    return (gr)
```

Αλγόριθμος 4.14: Graph pruning based on average edge weight

Η μέθοδο GSB παράγει πλήρες γραφήματα, η ανάλυση του γραφήματος σε πυρήνες, επιστρέφει πάντα μόνο ένα επίπεδο πυρήνα, το οποίο περιέχει όλο το γράφημα και θεωρούνται σημαντικοί κόμβοι, όλοι οι κόμβοι του γραφήματος. Άρα για να παραχθούν ουσιαστικά αποτελέσματα με τις μεθόδους που λαμβάνουν υπόψιν σημαντικούς κόμβους, η απαλοιφή ακμών είναι αναγκαία.

MainCore

Ο εντοπισμός των κόμβων που αποτελούν τον κύριο πυρήνα του γραφήματος υλοποιήθηκε με την χρήση της συνάρτησης k_core της βιβλιοθήκης networkx, οι οποία δέχεται ως είσοδο ένα γράφημα G και το επιθυμητό βαθμό πυρήνα k, και επιστρέφει ως γράφημα τον πυρήνα του γραφήματος G με βαθμό k. Στην περίπτωση, που δεν οριστεί ο βαθμός k, η συνάρτηση επιστρέφει το κύριο πυρήνα.

Στην συνέχεια με την βοήθεια της βιβλιοθήκης numpy μεταφράζουμε το γράφημα που επιστράφηκε ως ένα πίνακα γειτνίασης.

Density

Η μέθοδο Density υλοποιήθηκε με την διαδικασία που περιγράφεται παρακάτω. Αρχικά, εξάγεται από το γράφημα ένα λεξιλόγιο που περιέχει ως κλειδιά τους κόμβους και ως τιμές τον πυρήνα που ανήκει ο κόμβος. Αυτό επιτυγχάνεται με την βοήθειά της συνάρτησης core_number που περιέχεται στην βιβλιοθήκη networkx. Έπειτα, εντοπίζονται τα μοναδικά επίπεδα πυρήνων του γραφήματος.

```
cores = core_number(prunedgr)
nlevels = []
for key in cores.keys() :
    if cores[key] not in nlevels:
        nlevels.append(cores[key])
density_list = []
for n in nlevels:
    kcoregraph = k_core(prunedgr,n,cores)
    density_list.append(density(kcoregraph))
bestindex = elbow(density_list)
bestcore = nlevels[bestindex]
kcore = k_core(prunedgr,bestcore,cores)
prunedadjm = nx.to_numpy_array(prunedgr, weight='weight')
```

Αλγόριθμος 4.15: Density Method

Στην συνέχεια, χρησιμοποιείται μία συνάρτηση που ονομάζεται density για τον υπολογισμό της πυκνότητας κάθε επιπέδου.

```
def density(A_graph):
    graph_edges = A_graph.number_of_edges()
    graph_nodes = len(list(A_graph.nodes))
    dens = graph_edges/(graph_nodes*(graph_nodes-1))
    return dens
```

Αλγόριθμος 4.16: Density Calculation of each level

Τέλος, χρησιμοποιείται η συνάρτηση elbow η οποία εντοπίζει το κατάλληλο επίπεδο πυρήνα.

```
def elbow(listofpoints):
    #at first we need to create a line between first and last element
    if len(listofpoints)==1:
        bestindex = 0
    elif len(listofpoints)==2:
        if listofpoints[0]>listofpoints[1]:
            bestindex = 0
        else:
            bestindex = 1
    elif len(listofpoints)>2:
        #p1 the starting point of line and p2 the last point of line
        #using that we will calulate the distance of each point of our
            starting list
        #from the line using the known forumla
        p1 = numpy.array([listofpoints[0],0])
        p2 = numpy.array([listofpoints[-1],(len(listofpoints)-1)])
        distance = []
        pnt = []
        for point in listofpoints:
            pnt.append(point)
            pnt.append(listofpoints.index(point)+1)
            distance.append(distance_to_line(p1,p2,pnt))
            pnt =[]
        bestdistance = max(distance)
        bestindex = distance.index(bestdistance)
    return bestindex
```

Αλγόριθμος 4.17: Elbow function

CoreRank

Για την μέθοδο CoreRank, αρχικά εξάγεται από το γράφημα ένα λεξιλόγιο που περιέχει ως κλειδιά τους κόμβους και ως τιμές τον πυρήνα που ανήκει ο κόμβος. Όπως και στην μέθοδο density, χρησιμοποιήθηκε η συνάρτηση core_number της βιβλιοθήκης networkx. Στην συνέχεια, κάθε κόμβος αποθηκεύεται σε μία λίστα και υπολογίζεται το άθροισμα των αριθμών των επιπέδων κάθε γειτονικού του κόμβου, το οποίο αποθηκεύεται επίσης σε μία λίστα. Οι δύο λίστες συνδυάζονται(με την χρήση της συνάρτησης zip) και ταξινομούνται(με την χρήση της συνάρτησης sorted) φθίνουσα ως προς το άθροισμα των πυρήνων των γειτόνων. Τέλος, ορίζουμε ως σημαντικούς κόμβους ένα ποσοστό των εγγραφών αυτής της λίστας, ξεκινώντας απο την αρχή και συνεχίζοντας προς το τέλος της.

```
cores = core_number(prunedgr)
CoreRank = [] #the score of the sums
label = [] #the name of node corerank and label is connected via their
    index
for nd in prunedgr:
    sum = 0
```

```
    label.append(nd)
    for nbrs in prunedgr.neighbors(nd):
        sum += cores[nbrs]
    CoreRank.append(sum)
zipped_list = zip(label,CoreRank)
s_zipped_list = sorted(zipped_list,key=itemgetter(1), reverse=True)
no_of_words = len(s_zipped_list)
core_size = round(no_of_words*0.3)
kcore_nodes = []
for i in range(core_size):
    kcore_nodes.append(s_zipped_list[i][0])
prunedadjm = nx.to_numpy_array(prunedgr, weight='weight')
```

Αλγόριθμος 4.18: CoreRank method

4.5.4 Δημιουργία του συνολικού γραφήματος συλλογής.

Το επόμενο βήμα της υλοποίησής είναι η κατασκευή του συνολικού γραφήματος της συλλογής. Η διαδικασία της κατασκευής του συνολικού γραφήματος της συλλογής δεν πραγματοποιείται αφού έχουν παραχθεί όλα τα γραφήματα κειμένων. Αντίθετα, καθώς παράγονται τα γραφήματα κειμένων, εντάσσονται ένα-ένα με την σειρά τους στο συνολικό γράφημα. Σύμφωνα με αυτό, χρησιμοποιήθηκε μία συνάρτηση που ονομάζεται uniongraph, και δέχεται ένα γράφημα κειμένου (γράφημα, πίνακας γειτνίασης, λίστα όρων) και το συνολικό γράφημα της συλλογής με τα γραφήματα κειμένων ως τώρα (συνολικό γράφημα, λίστα όρων συνολικού γραφήματος). Επιπλέον, δέχεται μία λίστα με τους όρους της συλλογής (collection_terms) και τις συχνότητες εμφάνισις τους (collection_term_freq), το τρέχον αναγνωριστικό (id) για την λίστα με τους όρους της συλλογής, μία μεταβλητή σημαία (corebool), την λίστα με τους σημαντικούς κόμβους ενός γραφήματος κειμένου(kcore) και τέλος μία ακόμα μεταβλητή σημαία (UnionPen). Η μεταβλητή corebool είναι στο λογικό αληθές(True) μόνο όταν χρησιμοποιούμε μεθόδους που λαμβάνουν υπόψιν σημαντικούς κόμβους. Αντίστοιχα, η μεταβλητή UnionPen είναι στο λογικό αληθές (True) μόνο όταν θέλουμε να εισάγουμε ποινή στις ακμές του συνολικού γραφήματος. Η συνάρτηση επιστρέφει το ανανεωμένο συνολικό γράφημα της συλλογής, το οποίο πλέον περιλαμβάνει το γράφημα κειμένου που δόθηκε ως είσοδο, καθώς και τις λίστες που αφορούν όλη την συλλογή (collection_terms, collection_term_freq).

Παρακάτω θα γίνει ανάλυση των βημάτων ένταξης ενός γραφήματος κειμένου στο συνολικό γράφημα. Αρχικά, γίνεται έλεγχος αν θα πρέπει να εφαρμοσθεί ποινή στις ακμές του συνολικού γραφήματος. Στην περίπτωση που θα εφαρμοσθεί, η τιμή μίας μεταβλητής με όνομα union_penalty γίνεται ίση με το επιθυμητό ποσοστό ποινής, ειδάλλως η τιμή της μεταβλητής γίνεται ίση με την μονάδα.

Στην συνέχεια για κάθε όρο του κειμένου, πρώτα ελέγχεται αν ανήκει στους σημαντικούς όρους και αν ανήκει θέτεται μία μεταβλητή με όνομα h ίση με το μπόνους που

έχει ορισθεί από την γραμμή εντολών κατά την έναρξη του προγράμματος.

Έπειτα, εξετάζεται αν ο όρος υπάρχει στην λίστα με τους όρους της συλλογής και αν δεν υπάρχει τότε εισάγεται στην προαναφερθείσα λίστα και στην λίστα με τους όρους του συνολικού γραφήματος. Ακόμα, υπολογίζεται και αποθηκεύεται το βάρος της εσωτερικής του ακμής. Επίσης, δημιουργείται ένας κόμβος για τον όρο στο συνολικό γράφημα. Σε αυτό το σημείο, η τιμή του τρέχοντος αναγνωριστικού id αυξάνεται κατά μία μονάδα. Στην περίπτωση που υπάρχει ήδη στην λίστα με τους όρους της συλλογής, απλά ενημερώνεται προσθετικά το βάρος της εσωτερικής ακμής.

Στο επόμενο στάδιο, εξετάζεται ο πίνακας γειτνίασης και δημιουργούνται εξωτερικές ακμές για κάθε συσχέτιση του αρχικού όρου, έστω i, με κάποιον άλλον όρο, έστω j. Το βάρος της εξωτερικής ακμής E_{ij} είναι η τιμή του πίνακα γειτνίασης στην θέση που ορίζεται από τους δύο όρους, πολλαπλασιασμένο από την μεταβλητή union_penalty. Στην περίπτωση που υπάρχει ήδη η εξωτερική ακμή, όπως και στις εσωτερικές ακμές, ενημερώνεται προσθετικά το βάρος της ακμής. Σημειώνεται ότι πριν την δημιουργία της εξωτερικής ακμής ο όρος j, αντίστοιχα με τον όρο i, εισάγεται στην λίστα με τους όρους της συλλογής, αν δεν υπάρχει ήδη και τότε γίνεται υπολογισμός του βάρους της εσωτερικής του ακμής. Τέλος, εισάγεται στην λίστα με τους όρους του συνολικού γραφήματος και δημιουργείται ένας κόμβος για αυτόν. Το αναγνωριστικό id αυξάνεται κατά μία μονάδα.

```
def uniongraph(terms, term_freq, adjmatrix, collection_terms,
    union_graph_termlist_id, union_gr, id,
              collection_term_freq, kcore, kcorebool,UnionPen):
    if UnionPen:
        union_penalty = 0.0595
    else:
        union_penalty = 1

    for i in range(len(adjmatrix)):
        if i in kcore and kcorebool == True:
            h = hargs
        else:
            h = 1
        if terms[i] not in collection_terms:
            collection_terms[terms[i]] = id
            union_graph_termlist_id.append(id)
            collection_term_freq.append(term_freq[i] * (term_freq[i] +
                1) * 0.5* h * union_penalty)
            union_gr.add_node(terms[i], id=id)
            id += 1
        elif terms[i] in collection_terms:
            index = collection_terms[terms[i]]
            collection_term_freq[index] += term_freq[i] * (term_freq[i]
                + 1) * 0.5* h * union_penalty
        for j in range(len(adjmatrix)):
            if i > j:
                if adjmatrix[i][j]!=0:
                    if terms[j] not in collection_terms:
                        collection_terms[terms[j]] = id
                        union_graph_termlist_id.append(id)
                        collection_term_freq.append(term_freq[j] * (
                            term_freq[j] + 1) * 0.5 * h * union_penalty
                            )
```

```
                    union_gr.add_node(terms[i], id=id)
                    id += 1
                if not union_gr.has_edge(terms[i], terms[j]):
                    union_gr.add_edge(terms[i], terms[j], weight=
                        adjmatrix[i][j] * union_penalty)
                elif union_gr.has_edge(terms[i], terms[j]):
                    union_gr[terms[i]][terms[j]]['weight'] +=
                        adjmatrix[i][j] * union_penalty
    return terms, adjmatrix, collection_terms, union_graph_termlist_id,
        union_gr, id, collection_term_freq
```

Αλγόριθμος 4.19: Uniongraph Function

4.5.5 Δημιουργία ανεστραμμένου ευρετηρίου

Ολοκληρώνοντας την κατασκευή του συνολικού γραφήματος της συλλογής, ο αλγόριθμος είναι έτοιμος για την παραγωγή του ανεστραμμένου ευρετηρίου για την μέθοδο που εκτελείται στο παρόν πείραμα. Πριν φτάσουμε, ωστόσο, στο σημείο υπολογισμού των βαρών των όρων της συλλογής, αναλύεται πρώτα το συνολικό γράφημα και εξάγονται από αυτό, μερικές χρήσιμες πληροφορίες για τους κόμβους του, οι οποίες θα διευκολύνουν των υπολογισμό των βαρών. Πιο συγκεκριμένα, δημιουργείται ένα μητρώο, του οποίου οι εγγραφές είναι μία τριπλέτα που αποτελείται από το όνομα του κόμβου, το άθροισμα των βαρών των εξωτερικών ακμών του κόμβου και του αριθμού των γειτονικών, σε αυτόν, κόμβων.

```
def Woutusinggraph(inputgraph):
    nd = inputgraph.nodes()
    woutlist = []
    for n in nd:
        woutlist.append([n, inputgraph.degree(n, weight='weight'), len(
            list(inputgraph.neighbors(n)))])
    return woutlist
```

Αλγόριθμος 4.20: Analysis of the union graph.

Για δημιουργία του ανεστραμμένου ευρετηρίου χρησιμοποιήθηκε μία συνάρτηση, που ονομάζεται w_and_write_to_filev2, και σκοπός της είναι ο υπολογισμός των βαρών των όρων σύμφωνα με τον τύπο 3.7 και η δημιουργία του ανεστραμμένου ευρετηρίου. Δέχεται ως είσοδο, το μητρώο που προέκυψε από την ανάλυση του συνολικού γραφήματος της συλλογής (wout), το λεξιλόγιο με τους όρους της συλλογής (collection_terms), την λίστα με τα βάρη των έσω-ακμών των κόμβων του συνολικού γραφήματος (collection_term_freq), την λίστα με τους όρους της συλλογής (postinglist) και το όνομα του ευρετηρίου.

```
def w_and_write_to_filev2(wout, collection_terms,
    union_graph_termlist_id, collection_term_freq, postinglist, file):
    # wout |[[term][wout][neibours]]\
    for entry in collection_terms:
        term = entry
```

```
        id = collection_terms[entry]
        win = collection_term_freq[id]
        for sublist in wout:
            if term in sublist[0]:
                Wo = sublist[1]
                nbrs = sublist[2]
                break
        temp = log( 1 + ((Wo / (nbrs + 1)) / (win + 1))) * log(1 + 10 *
            (1 / (nbrs + 1)))
        indexofwordinlist = 2 * id + 1
        graphToIndex(id, term, temp, postinglist[indexofwordinlist],
            filename=file)
    return 1
```

Αλγόριθμος 4.21: Inverted index creation.

Αφού υπολογισθεί το βάρος ενός όρου, καλείται μία συνάρτηση με όνομα graphToIndex, η οποία δημιουργεί μία εγγραφή για τον όρο στο αρχείο του ανεστραμμένου ευρετηρίου στη μνήμη. Η εγγραφή περιλαμβάνει το αναγνωριστικό (id) του όρου, το οποίο εξάγεται από το λεξικό collection_terms, το όνομα του όρου (term), το βάρος του κόμβου (temp) και την εγγραφή που αντιστοιχεί στον όρο από την postinglist. Η τελευταία περιλαμβάνει τα ονόματα των κειμένων στα οποία εμφανίζεται ο όρος, καθώς και την συχνότητά εμφάνισις του σε αυτά.

```
def graphToIndex(id, terms, calc_term_w, plist, *args, **kwargs):
    filename = kwargs.get('filename', None)
    if not filename:
        filename = 'inverted index.dat'
    f = open(filename, "a+")
    data = ','.join(
        [str(i) for i in plist]) # join list to a string so we can
            write it in the inv index and load it with ease
    f.write('%d;%s;%f;%s;\n' % (id, terms, calc_term_w, data))
    f.close()
    return 1
```

Αλγόριθμος 4.22: The graphToIndex function.

4.5.6 Ανάκτηση πληροφορίας

Σε αυτό το υποκεφάλαιο θα γίνει αναφορά στην διαδικασία ανάκτησης πληροφορίας από τα ευρετήρια που δημιουργήθηκαν παραπάνω. Όπως έχει ήδη αναφερθεί ακολουθείται η διαδικασία ανάκτησης σύμφωνα με το GSB. Αρχικά, πρέπει να διαβαστούν τα ευρετήρια που παράχθηκαν προηγούμενος. Αυτό επιτυγχάνεται με την χρήση μίας συνάρτησης, η οποία αναλαμβάνει να διαβάσει κάθε ευρετήριο από την μνήμη και να εξάγει χρήσιμη πληροφορία από αυτό. Η συνάρτηση ονομάζεται load_inv_index και δέχεται ως είσοδο το όνομα ενός από τα παραγόμενα ευρετήρια του αλγορίθμου ανάκτησης. Το ευρετήριο είναι πρακτικά ένα αρχείο csv, όπου οι στήλες του χωρίζονται με τον χαρακτήρα ";". Για διευκόλυνσή, το ευρετήριο διαβάζεται με την βοήθεια μίας βιβλιοθήκης που ονομάζεται csv. Επιστρέφονται τέσσερις λίστες που περιέχουν τα

αναγνωριστικά (ID) και τα ονόματα των όρων, τα αντίστοιχα βάρη τους και τα ονόματα των αρχείων που εμφανίζονται μαζί με την συχνότητα εμφάνισης τους.

```
def load_inv_index(*args):
    arg = list(args)
    if not arg:
        invindex = 'inverted index.dat'
    else:
        invindex = arg[0]
    ids = []
    trms = []
    W = []
    plist = []
    with open(invindex, 'r') as csvf:
        reader = csv.reader(csvf, delimiter=";")
        for row in reader:
            if row[0] not in ids:
                ids.append(row[0])
                trms.append(row[1])
                W.append(row[2])
                plist.append(row[3].split(','))
    csvf.close()
    # print(len(ids))
    return ids, trms, W, plist
```

Αλγόριθμος 4.23: Reading the inverted index

Στην συνέχεια, γίνεται προ-επεξεργασία κάθε ερωτήματος που δίνεται και παράγονται τα σύνολα των όρων που περιέχουν μόνο ένα όρο (μονό-σύνολα).

```
# Queries
Q = Query
# Q = input('Input Query : ')
print('Query ======== ',Q)
Q = Q.upper()
Q = Q.split()
Q = list(set(Q))  # to ignore duplicate words as queries
```

Αλγόριθμος 4.24: Query Preprocess

Το δεύτερο επιτυγχάνεται με την χρήση μίας συνάρτησης που ονομάζεται one_termsets και δέχεται ως είσοδο το ερώτημα, την λίστα με τα κείμενα που εμφανίζεται κάθε όρος και την συχνότητα εμφάνισης τους, και την ελάχιστη συχνότητα εμφάνισης που πρέπει να έχει κάθε όρος. Η συνάρτηση ελέγχει αν υπάρχει στην λίστα με τα ονόματα των όρων, κάθε όρος του ερωτήματος, και αν υπάρχει και ικανοποιείται και η συνθήκη της ελάχιστης συχνότητας, τότε αποθηκεύει το όνομα του όρου μαζί με τον αριθμό εμφάνισης του σε κάθε κείμενο σε μία λίστα.

```
# input the Query Q as a list of words consisting the Query
def one_termsets(Q, trms, plist, minfreq):
    termsets = []
    One_termsets = []
    for word in Q:
        if word in trms:
            i = trms.index(word)
            doc = plist[(i + 1)]
            doc = doc[::2]
```

```
            word = [''.join(word)]
            if len(doc) > minfreq:
                One_termsets.append([word, doc])
        else:
            print('word %s has not required support or it already
                exists:' % word)
    return One_termsets
```

Αλγόριθμος 4.25: Generating the one-termsets

Έχοντας παράξει τα μονό-σύνολα, ο αλγόριθμος είναι σε θέση να εντοπίσει τα συχνά σύνολα όρων. Για τον εντοπισμό τους, υλοποιήθηκε ο αλγόριθμος apriori. Καλείται λοιπόν μία συνάρτηση με όνομα apriori, που δέχεται ως είσοδο τα μονό-σύνολα και την ελάχιστη επιθυμητή συχνότητα εμφάνισης που θα έχουν και επιστρέφει μία λίστα με τα συχνά σύνολα όρων και τα κείμενα στα οποία εμφανίζεται κάθε σύνολο. Σε κάθε θέση της λίστας περιέχεται μία διαφορετική λίστα που περιέχει τα σύνολα μεγέθους k με $k = 1, 2, ..., n$. Στην θέση αμέσως μετά από ένα σύνολο, περιέχεται μία ακόμα λίστα που περιλαμβάνει τα ονόματα των κειμένων στα οποία εμφανίζεται το συγκεκριμένο σύνολο όρων. Αυτό φαίνεται καλύτερα στο παρακάτω σχήμα:

Σύνολα Όρων	S_{11}	Docs S_{11}	S_{12}	Docs S_{12}	...	S_{n1}	Docs S_{n1}	S_{n2}	Docs S_{n2}
Μέγεθος k			k=1		...			k=n	

Table 4.4: Λίστα Συχνών Συνόλων Όρων

```
def apriori(l1, minfreq):
    final_list = []
    final_list.append(l1)
    l = l1
    while (l != []):
        c = apriori_gen(l)
        l = apriori_prune(c, minfreq)
        final_list.append(l)
    return final_list
```

Αλγόριθμος 4.26: Apriori algorithm

Αρχικά, προσθέτεται στην τελική λίστα τα μονό-σύνολα αφού για αυτά έχει ήδη γίνει έλεγχος όσο αφορά την ελάχιστη συχνότητα τους. Έπειτα, χρησιμοποιείται μία δομή επανάληψης, η οποία παράγει νέα σύνολα όρων, μεγαλύτερα κατά ένα στοιχείο, ξεκινώντας από τα μονό-σύνολα. Στη συνέχεια ελέγχονται τα νέα σύνολα όσο αφορά την ελάχιστη συχνότητα τους και τα σύνολα όρων που είναι συχνά προσθέτονται στην τελική λίστα μαζί με τα κείμενα στα οποία περιέχονται. Τέλος, χρησιμοποιώντας αυτά τα συχνά σύνολα δημιουργούνται νέα σύνολα όρων, μεγαλύτερα κατά ένα όρο και η διαδικασία επαναλαμβάνεται. Όταν δεν υπάρχουν πλέον νέα συχνά σύνολα, ο αλγόριθμος έχει ολοκληρωθεί και επιστρέφεται η ολοκληρωμένη λίστα με τα συχνά σύνολα.

Για την δημιουργία νέων συνόλων όρων, μεγαλύτερα κατά ένα στοιχείο χρησιμοποιήθηκε μία συνάρτηση με όνομα apriori_gen, η οποία δέχεται ως είσοδο μία λίστα από σύνολα όρων, καθώς και τα ονόματα των κειμένων στα οποία εμφανίζεται κάθε σύνολο. Αρχικά επιλέγεται ένα σύνολο στοιχείων. Στη συνέχεια το σύνολο συνδυάζεται ανά δύο με σύνολα με τα οποία διαφέρει κατά ένα στοιχείο. Η ένωση δύο συνόλων αποθηκεύεται σε μία λίστα. Τέλος, αφού αποθηκευτούν όλα τα σύνολα που παράχθηκαν από αυτή την διαδικασία και τα κείμενα που εμφανίζονται, επιλέγεται το επόμενο σύνολο και η διαδικασία επαναλαμβάνεται.

```
def apriori_gen(itemset):
    candidate = []
    ck = []
    texts = []
    length = len(itemset)
    for i in range(length):
        ele = itemset[i][0]
        for j in range(i + 1, length):
            ele1 = itemset[j][0]
            if ele[0:len(ele) - 1] == ele1[0:len(ele1) - 1]:
                texts.append(intersection(itemset[i][1],itemset[j][1]))
                candidate.append([union(ele, ele1), intersection(
                    itemset[i][1], itemset[j][1])])
    return candidate

def intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))

def union(lista, listb):
    c = []
    for i in lista + listb:
        if i not in c:
            c.append(i)
    return c
```

Αλγόριθμος 4.27: Generating the new termsets

Έχοντας δημιουργήσει τα νέα σύνολα το μόνο που πρέπει να γίνει είναι έλεγχος για να εντοπιστούν ποια από αυτά είναι συχνά. Η διαδικασία ελέγχου είναι εξαιρετικά απλή. Για κάθε υποψήφιο σύνολο, γίνεται καταμέτρηση των κειμένων που εμφανίζεται και αν ο αριθμός τους είναι μεγαλύτερος από την ελάχιστη συχνότητα εμφάνισης, το σύνολο θεωρείται συχνό.

```
def apriori_prune(termsets_list, min_support):
    prunedlist = []
    for j in termsets_list:
        if len(j[1]) > min_support:
            prunedlist.append([j[0], j[1]])
    return prunedlist
```

Αλγόριθμος 4.28: Determing frequent termsets

Τα επόμενα βήματα εκτελούνται για κάθε διαφορετική μέθοδο, χρησιμοποιώντας τις αντίστοιχες μεταβλητές κάθε μεθόδου.

Το επόμενο βήμα του αλγόριθμου είναι ο υπολογισμός των βαρών για κάθε σύνολο

όρων. Πριν φτάσει σε αυτό το σημείο όμως, είναι απαραίτητο να υπολογισθούν οι συχνότητες των όρων των συνόλων όρων(termset frquency) στα κείμενα που τα περιέχουν και οι αντίστροφες συχνότητες κειμένων των όρων των συνόλων όρων (inverse document frequency). Για τα termset frequency χρησιμοποιήθηκε μία συνάρτηση που ονομάζεται fij_calculation και δέχεται ως είσοδο το αρχείο docinfo, την λίστα με τα συχνά σύνολα, την λίστα με τα αρχεία στα οποία εμφανίζεται κάθε όρος και τέλος την λίστα των όρων. Επιστρέφει, δύο λίστες, μία που περιέχει τα κείμενα της συλλογής και μία που περιέχει λίστες με τις συχνότητες εμφάνισης κάθε συχνού συνόλου όρων για κάθε κείμενα.

```
def fij_calculation(docinfo, final_list, plist, trms):
 doc_vectors = []
 docs = []
 # counting the number of apperances of each termset in each doc in
     which the set exists
 for document in docinfo:
  docs.append([document[0]])
  # test is a temp list which contains the termfreq of TSets for the
      current doc, then we append that list to create
  # a matrix of [Docs X Termsets].
  test = []
  for itemsets in final_list:
   for i in range(len(itemsets)):
    # calculating termset frequency
    if document[0] in itemsets[i][1]:
     sum = 0
     for term in itemsets[i][0]:
      if term in trms:
       termindx = trms.index(term)
       if document[0] in plist[termindx]:
        docindex = plist[termindx].index(document[0])
        sum += int(plist[termindx][docindex + 1])
        # to ameswS epomeno stoixeio antistoixei ston ari8mo emfanisis
            tou orou TERM(i) sto Document(j)
     test.append(sum)
    else:
     test.append(0)
  doc_vectors.append(test)
 with open('debuglog.dat', 'a') as fd:
  fd.write('doc vectors \n')
  for doci in doc_vectors:
   fd.write('%s \n' % str(len(doci)))
 fd.close()
 return docs, doc_vectors
```

Αλγόριθμος 4.29: Termset Frequency Calculation

Η λειτουργία της συνάρτησης είναι η ακόλουθη. Για κάθε συχνό σύνολο όρων, με την βοήθεια δομών επανάληψις, εντοπίζονται τα κείμενα τα οποία περιέχουν το σύνολο όρων και έπειτα μετριούνται πόσες φορές εμφανίζεται σε αυτό το κείμενο κάθε όρος του συνόλου. Τέλος, γίνεται εγγραφή μερικών πληροφοριών, που βοηθάνε στην αποσφαλμάτωση, σε ένα αρχείο με όνομα "debug.dat".

Για το inverse document frequency(idf) χρησιμοποιήθηκε μία συνάρτηση με όνομα calculate_idf, η οποία χρησιμοποιεί την λίστα με τα συχνά σύνολα και υπολογίζει το σκέλος του βάρους που αντιστοιχεί στο idf. Επιστρέφει ένα διάνυσμα/λίστα με τις

αντίστροφες συχνότητες κειμένου για κάθε συχνό σύνολο όρων.

```
def calculate_idf(termsetsL, numofdocs):
    idf_vector = []
    for ts in termsetsL:  # iterate based on the number of terms in
        termset
        for item in ts:  # iterate all termsets with the same number of
            terms in set
            Nt = len(item[1])
            N = numofdocs
            if Nt != 0:
                idf = log(1 + (N / Nt))
                idf_vector.append(idf)
            else:
                idf_vector.append(0)
    return idf_vector
```

Αλγόριθμος 4.30: Termset IDF Calculation

Το μόνο που απομένει είναι ο υπολογισμός του σκέλους του τελικού βάρους που προέρχεται από το GSB. Για άλλη μία φορά χρησιμοποιήθηκε μία συνάρτηση με όνομα calculate_termset_W, η οποία επιστρέφει, χρησιμοποιώντας την λίστα με τα συχνά σύνολα όρων και τους όρους συλλογής μαζί με τα βάρη τους, ένα τελευταίο διάνυσμα που περιέχει τα βάρη κάθε συνόλου όρου σύμφωνα με το GSB.

```
def calculate_termset_W(termsetsL, W, terms):
    termset_W_vector = []
    for ts in termsetsL:  # iterate based on the number of terms in
        termset
        for item in ts:  # iterate all termsets with the same number of
            terms in set
            product = 1
            for term in item[0]:
                if term in terms:
                    tindx = terms.index(term)
                    weight = W[tindx]
                    product *= float(weight)
            termset_W_vector.append(product)

    return termset_W_vector
```

Αλγόριθμος 4.31: GSB weight calculation

Η λειτουργία της συνάρτησης είναι αρκετά απλή. Για κάθε όρο, ενός συχνού συνόλου όρων, εντοπίζει το βάρος του που προέρχεται από το ανεστραμμένο ευρετήριο και το συνδυάζει με τα αντίστοιχα βάρη των υπόλοιπων όρων. Το αποτέλεσμα αποθηκεύεται στο τελικό διάνυσμα που επιστρέφεται. Η διαδικασία επαναλαμβάνεται για κάθε συχνό σύνολο όρων.

Τα τρία διανύσματα συνδυάζονται σε ένα τελικό διάνυσμα σύμφωνα με τον τύπο 3.9.

```
def doc_rep(doc_vec, idf_vec, *args, **kwargs):
    args = list(args)
    if not args:
        nw = []
        for i in range(len(idf_vec)):
            nw.append(1)
```

```
    else:
        nw = args[0]
    test = numpy.zeros((len(doc_vec), len(idf_vec)))
    for i in range(len(doc_vec)):
        for j in range(len(idf_vec)):
            if doc_vec[i][j] > 0:
                test[i][j] = (1 + log(doc_vec[i][j])) * idf_vec[j] *
                    float(nw[j])
            else:
                test[i][j] = 0
    return test
```

Αλγόριθμος 4.32: Final weight calculation

Το τελευταίο βήμα είναι ο υπολογισμός της ομοιότητας του συνημιτόνου για κάθε κείμενο σε σχέση με το ερώτημα.

```
def q_D_similarities(q, docmatrix, docs):
    ret_list = []
    cnt = 0
    for doci in docmatrix:
        try:
            temp = cos_sim(q, doci)
            ret_list.append([docs[cnt], temp])
            cnt += 1
        except ValueError:
            print(doci)
            print(len(doci))
            print(cnt)
            exit(-1)
    return ret_list
```

Αλγόριθμος 4.33: Query - Document Similarity

Για κάθε κείμενο υπολογίζεται η ομοιότητα του συνημιτόνου και αποθηκεύεται, μαζί με το όνομα του αρχείου σε ένα πίνακα.

```
# computes the cosine similarity of 2 vectors
def cos_sim(a, b):
    a = numpy.asarray(a)
    b = numpy.asarray(b)
    if numpy.all(a == 0) or numpy.all(b == 0):
        ret = 0
    else:
        dot_p = numpy.dot(a, b)
        normA = numpy.linalg.norm(a)
        normB = numpy.linalg.norm(b)
        ret = dot_p / (normA * normB)
    return ret
```

Αλγόριθμος 4.34: Similarity Calculation

Τέλος, ταξινομούνται τα αποτελέσματα κατά φθίνουσα σειρά σύμφωνα με την ομοιότητα του ερωτήματος και του κειμένου. Έπειτα, υπολογίζονται οι μετρικές ακρίβεια - ανάκληση και αποθηκεύονται τα αποτελέσματα σε ένα αρχείο excel, με την βοήθεια της βιβλιοθήκης openpyxl, για περαιτέρω ανάλυση. Προαιρετικά παράγονται και διάφορα διαγράμματα με την βοήθεια της βιβλιοθήκης matplotlib.

```
def pre_rec_calculation(rev_list,relevant):
```

```
    cnt = 0
    retrieved = 1
    recall = []
    precision = []
    for doc in rev_list:
        if doc in relevant:
            cnt += 1
            p = cnt / retrieved
            r = cnt / len(relevant)
            precision.append(p)
            recall.append(r)
        retrieved += 1
    av_pre = average(precision)
    av_rec = average(recall)
    return av_pre, av_rec, precision, recall
```

Αλγόριθμος 4.35: Precision - Recall Calculation

4.5.7 Ανάλυση και προ-επεξεργασία των συλλογών

Οι συλλογές που χρησιμοποιήθηκαν δεν ήταν σε κατάσταση ώστε να μπορούν να χρησιμοποιηθούν απευθείας με τις μεθόδους και τις συναρτήσεις που υλοποιήθηκαν, με εξαίρεση της συλλογής CF, και χρειαζόταν περαιτέρω ανάλυση και προ-επεξεργασία. Οι τελικές μορφές που είχαν τα διάφορα στοιχεία της συλλογής ήταν διαφορετικές ανάλογα το στοιχείο. Κάθε κείμενο αντιστοιχούσε σε ένα διαφορετικό αρχείο με όνομα το αναγνωριστικό του κειμένου (αριθμός) και οι λέξεις μέσα στο κείμενο ήταν σε κεφαλαία γράμματα χωρίς στοιχεία στίξης, ίσως εκτός από την τελεία σε περίπτωση που χρησιμοποιόταν η αντίστοιχη μέθοδο. Τα ερωτήματα αναλύθηκαν σε μία λίστα, με ένα ερώτημα σε κάθε θέση της λίστας, και τα συναφή κείμενα για κάθε ερώτημα αναλύθηκαν σε μία λίστα από λίστες, όπου κάθε επιμέρους λίστα περιείχε τα αντίστοιχα συναφή κείμενα (αναγνωριστικά μόνο) για κάθε ερώτημα. Στην περίπτωση που η συλλογή περιλάμβανε λίστα με ασήμαντες λέξεις, αυτές αναλυόταν σε μία λίστα, όπου κάθε θέση της περιείχε μία ασήμαντη λέξη. Σε αυτό το σημείο θα γίνει μία συνοπτική αναφορά στην υλοποίηση των παραπάνω αναλύσεων. Για κάθε συλλογή χρησιμοποιήθηκε παρόμοια αλλά διαφορετική υλοποίηση.

Ανάλυση στην συλλογή NPL

Αρχικά, για την NPL έπρεπε να αναλυθούν τα κείμενα. Αυτό επιτεύχθηκε με τον αλγόριθμος που φαίνεται στο 4.36. Αν και φαίνεται πολύπλοκος, η λειτουργία του είναι αρκετά απλή. Στην συλλογή NPL κάθε κείμενο στο αρχείο κειμένων, χωρίζεται από το προηγούμενο με τον χαρακτήρα "/". Επίσης στο τέλος του αρχείου έχει προστεθεί η λέξη "END" για την διευκόλυνση μας. Λαμβάνοντας αυτά υπόψιν, ο αλγόριθμος διαβάζει κάθε γραμμή γραμμή του κειμένου και αποθηκεύει το κείμενο σε μία λίστα, μέχρι να συναντήσει τον χαρακτήρα "/". Όταν τον συναντήσει καλεί μία συνάρτηση όπου προ-επεξεργάζεται το κείμενο και το αποθηκεύει σε ένα αρχείο, αναθέτοντας του

και ένα αναγνωριστικό. Η παραπάνω διαδικασία συνεχίζεται μέχρι να συναντηθεί η λέξη "END" στο τέλος του αρχείου κειμένων, όπου και τερματίζει ο αλγόριθμος.

```
import nltk
import os
import string
path = "src"
dest_path = "dest/"
def List_to_write(filename,abstract_list):
    list_of_words = abstract_list
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in list_of_words]
    list_of_words = [word.upper() for word in list_of_words]
    list_of_words_size = len(list_of_words)
    rewrite_file(filename,list_of_words)
    return list_of_words
def rewrite_file(filename,list_of_words_to_write):
    filename="dest/"+filename
    fd = open(filename,'w')
    for w in list_of_words_to_write:
        fd.write("%s\n"% w)
    return 0
filecount = 0
count = 0
for file in os.listdir(path):
    filecount+=1
    full_path = path + file
    if os.path.isfile(full_path):
        fd = open(full_path, 'rt')
        Text = fd.readline()
        text=Text.split()
        filename = text[0]
        line=fd.readline()
        sline=line.split()
        while(line!="END"):
            if(sline[0]!="/"):
                print(sline)
                abstract = []
                if len(sline)>0:
                    abstract.extend(sline)
                    line = fd.readline(500)
                    sline = line.split()
                while sline[0] != "/" :
                    abstract.extend(sline)
                    line = fd.readline(500)
                    sline = line.split()
                List_to_write(filename,abstract)
            else:
                count+=1
                Text = fd.readline()
                text=Text.split()
                filename = text[0]
                line=fd.readline()
                sline=line.split()
                if len(sline)==0 or sline[0]=="END":
                    print(count)
                    break
        print("END")
print(count)
```

Αλγόριθμος 4.36: NPL text parser

Όσο αφορά τα ερωτήματα, η διαδικασία είναι ακόμα πιο απλή. Τα ερωτήματα στο αρχείο ερωτημάτων δεν ξεπερνάνε την μία γραμμή, άρα ο αλγόριθμος που υλοποιήθηκε

απλά διαβάζει μόνο τις γραμμές με τα ερωτήματα και τα αποθηκεύει σε μία λίστα, η οποία στην συνέχεια αποθηκεύεται σε ένα αρχείο. Όπως και πριν, ο αλγόριθμος τερματίζει όταν συναντήσει την λέξη "END", η οποία έχει προστεθεί στο τέλος από το αρχείο. Η υλοποίηση φαίνεται στο 4.37.

```
queries=[]
with open('query-text', 'r') as fd:
    line =fd.readline()
    while line and line !="END":
        line = fd.readline()
        queries.append(line)
        fd.readline()
        fd.readline()
    fw=open('queries.txt','w')
    fw.write(str(queries))
    fw.close()
```

Αλγόριθμος 4.37: NPL query parser

Τέλος, για την συλλογή NPL μένει να αναλυθεί η υλοποίηση για τα συναφή κείμενα για κάθε ερώτημα. Η υλοποίηση φαίνεται στο 4.38. Αρχικά, διαβάζονται οι γραμμές με τα συναφή κείμενα για ένα ερώτημα και κάθε αναγνωριστικό κειμένου αποθηκεύεται σε μία λίστα. Όταν ο αλγόριθμος συναντάει τον χαρακτήρα "/", αποθηκεύει την παραπάνω λίστα σε μία λίστα λιστών και ξεκινάει να διαβάζει τα αναγνωριστικά των συναφών κειμένων για το επόμενο ερώτημα. Η διαδικασία συνεχίζεται μέχρι να συναντηθεί η λέξη "END", η οποία έχει προστεθεί στο τέλος του αρχείου.

```
relevant=[]
sline2 =[]
with open('rlv-ass', 'r') as fd:
    line =fd.readline()
    sline = line.split()
    while sline[0]!="END":
        while sline[0]!="/":
            line = fd.readline()
            sline = line.split()
            sline2 = sline2 + sline
        line = fd.readline()
        sline = line.split()
        relevant.append(sline2)
        sline2=[]
        if len(sline)==0 or sline[0]=="END":
            break
    fw=open('relev.txt','w')
    fw.write(str(relevant))
    fw.close()
```

Αλγόριθμος 4.38: NPL relevant texts parser

Ανάλυση στην συλλογή Cranfield

Όπως και στην συλλογή NPL, θα εξεταστεί αρχικά η υλοποίηση για την ανάλυση των κειμένων. Η υλοποίηση φαίνεται στο 4.39. Το αρχείο κειμένων της συλλογής

Cranfield χρησιμοποιεί μερικές ετικέτες, οι οποίες δηλώνουν το είδος του κειμένου που ακολουθεί, όπως για παράδειγμα την ετικέτα ".T" για τον τίτλο ή την ετικέτα ".W" για το κείμενο. Σκοπός του αλγόριθμου είναι να εξάγει σε αρχεία τους τίτλους και τα κείμενα, αποφεύγοντας ετικέτες όπως για παράδειγμα ".A" που ακολουθούνται από το όνομα του συγγραφέα. Σύμφωνα με αυτό, οι ενδιαφέροντες ετικέτες είναι οι ".T" και η ".W". Με την βοήθεια δομών επανάληψις και δομών ροής, ο αλγόριθμος εξάγει το απαιτούμενο κείμενο, το επεξεργάζεται και το αποθηκεύει σε αρχεία αναθέτοντας τους και ένα αναγνωριστικό. Η παραπάνω διαδικασία συνεχίζεται μέχρι να συναντηθεί η λέξη "END" που έχει προστεθεί στο τέλος του αρχείου κειμένων.

```
import nltk
import os
import string
path = "src/"
dest_path = "dest"
def List_to_write(filename,abstract_list):
    list_of_words = abstract_list
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in list_of_words]
    rewrite_file(filename,list_of_words)
    return list_of_words
def rewrite_file(filename,list_of_words_to_write):
    filename="dest/"+str(filename)
    fd = open(filename,'w')
    for w in list_of_words_to_write:
        fd.write("%s\n"% w)
    return 0
filecount = 0
count = 0
for file in os.listdir(path):
    filecount+=1
    full_path = path + file
    if os.path.isfile(full_path):
        fd = open(full_path, 'rt')
        filename = count
        line=fd.readline()
        sline=line.split()
        while(sline[0]!="END"):
            if sline[0]==".T":
                abstract = []
                line = fd.readline()
                sline = line.split()
                while sline[0] != ".A" :
                    abstract.extend(sline)
                    line = fd.readline()
                    sline = line.split()
                while sline[0] !=".I":
                    line = fd.readline()
                    sline = line.split()
                    if sline[0]==".W":
                        line = fd.readline()
                        sline = line.split()
                        while sline[0] != ".I" :
                            abstract.extend(sline)
                            line = fd.readline()
                            sline = line.split()
                string_punct_no_dot = string.punctuation.translate({ord
                    ('.'): None})
```

```
                table = str.maketrans('', '', string_punct_no_dot)
                stripped = [w.translate(table) for w in abstract]
                List_to_write(filename,stripped)

            else:
                count+=1
                print(count)
                filename = count
                line=fd.readline()
                sline=line.split()
        print("END")
print(count)
```

Αλγόριθμος 4.39: Cranfield text parser

Σειρά έχει η ανάλυση του αρχείου ερωτημάτων. Όπως και το αρχείο κειμένων, έτσι και το αρχείο ερωτημάτων χρησιμοποιεί ετικέτες, αν και είναι πολύ λιγότερες και απλούστερες. Η μόνη ενδιαφέρον ετικέτα είναι η ".W", γιατί περιέχει το κείμενο των ερωτημάτων. Ακολουθώντας την ίδια λογική με το αρχείο κειμένων, εξάγεται το κείμενο των ερωτημάτων και αποθηκεύεται σε μία λίστα. Όταν ο αλγόριθμος συναντήσει την λέξη "END" που έχει προστεθεί στο τέλος του κειμένου, αποθηκεύει στην λίστα με τα ερωτήματα σε ένα αρχείο και τερματίζει. Η υλοποίηση φαίνεται στο 4.40.

```
import os
import string
queries=[]
line2 =""
sline=[1]
with open('cran.qry', 'r') as fd:
    line =fd.readline()
    sline = line.split()
    while(sline[0]!="END"):
        while sline[0] ==".I":
            line = fd.readline()
            sline = line.split()
            if sline[0]==".W":
                line = fd.readline()
                sline = line.split()
                while sline[0] != ".I":
                    line2 = line2 + line
                    line = fd.readline()
                    sline = line.split()
                    if len(sline)>0 and sline[0] == "END":
                        break
                print(line2)
                queries.append(line2)
                line2=""
                line = ""
                if len(sline)>0 and sline[0] == "END":
                    break
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in queries]
    fw=open('queries.txt','w')
    fw.write(str(stripped))
    fw.close()
```

Αλγόριθμος 4.40: Cranfield query parser

Τέλος, για το αρχείο των συναφών κειμένων για κάθε ερώτημα χρησιμοποιήθηκε ο αλγόριθμος 4.41. Η μορφή του αρχείο με τα συναφή κείμενα είναι αρκετά διαφορετική

από τα άλλα αρχεία. Κάθε γραμμή περιέχει τρεις αριθμούς. Ο πρώτος είναι ένας δείκτης που δηλώνει τον αριθμό του ερωτήματος στο οποίο ανήκει το συναφή κείμενο, ο δεύτερος είναι το αναγνωριστικό του συναφή κειμένου και ο τρίτος είναι μάλλον η βαρύτητα του συναφούς κειμένου. Στην παρών διπλωματική δεν χρησιμοποιήθηκε η βαρύτητα, άρα ενδιαφέρον έχουν μόνο οι δύο πρώτοι αριθμοί. Με την βοήθεια μίας μεταβλητής μετρητή, αποθηκεύονται τα κείμενα ενός ερωτήματος σε μία λίστα και στην συνέχεια αυτή η λίστα αποθηκεύεται σε μία λίστα λιστών. Όπως και στις προηγούμενες περιπτώσεις ο αλγόριθμος σταματάει όταν συναντήσει την λέξη "END" που έχει προστεθεί στο τέλος του αρχείου των συναφών κειμένων και αποθηκεύει την τελική λίστα λιστών σε ένα αρχείο στο δίσκο.

```
relevant=[]
sline2 =[]
counter = 1
with open('cranqrel', 'r') as fd:
    line =fd.readline()
    sline = line.split()
    while sline[0]!="END":
        if sline[0] == str(counter):
            while sline[0] == str(counter):
                sline2.append(sline[1])
                line = fd.readline()
                sline = line.split()
                if sline[0] != str(counter):
                    relevant.append(sline2)
                    sline2=[]
                    counter+=1
            if len(sline)==0 or sline[0]=="END":
                break
    fw=open('relev.txt','w')
    fw.write(str(relevant))
    fw.close()
```

Αλγόριθμος 4.41: Cranfield relevant texts parser

Ανάλυση στην συλλογή Time

Όπως και στις προηγούμενες συλλογές, πρώτα θα γίνει αναφορά στην ανάλυση του αρχείου κειμένων. Η υλοποίηση φαίνεται στο 4.42. Η συλλογή Time ξεχωρίζει τα κείμενα της μέσα στο αρχείο χρησιμοποιώντας την ετικέτα "*TEXT". Σύμφωνα με αυτό ο αλγόριθμος που χρησιμοποιήθηκε αποθηκεύει σε το κείμενο μετά από μία ετικέτα "*TEXT", μέχρι και την επόμενη, το επεξεργάζεται και το αποθηκεύει σε ένα αρχείο, αναθέτοντας του και ένα αναγνωριστικό. Η παραπάνω διαδικασία συνεχίζεται μέχρι να συναντηθεί μία ετικέτα "*STOP", η οποία έχει προστεθεί στο τέλος του αρχείου κειμένων.

```
import nltk
import os
import string
```

```
path = "src/"
dest_path = "dest/"
def List_to_write(filename,abstract_list):
    list_of_words = abstract_list
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in list_of_words]
    stop_words = set(stopwords.words('english'))
    list_of_words = [word.upper() for word in list_of_words]
    list_of_words_size = len(list_of_words)
    rewrite_file(filename,list_of_words)
    return list_of_words
def rewrite_file(filename,list_of_words_to_write):
    filename="dest/"+str(filename)
    fd = open(filename,'w')
    for w in list_of_words_to_write:
        fd.write("%s\n"% w)
    return 0
filecount = 0
count = 0
for file in os.listdir(path):
    filecount+=1
    full_path = path + file
    if os.path.isfile(full_path):
        fd = open(full_path, 'rt')
        filename = count
        line=fd.readline()
        sline=line.split()
        while(len(sline) == 0 or sline[0]!="*STOP"):
            if len(sline) == 0 or sline[0]!="*TEXT":
                abstract = []
                while len(sline) == 0 or sline[0] != "*TEXT" :
                    abstract.extend(sline)
                    line = fd.readline()
                    sline = line.split()
                    if len(sline)>0 and sline[0] == '*STOP':
                        break
                string_punct_no_dot = string.punctuation.translate({ord
                    ('.'): None})
                table = str.maketrans('', '', string_punct_no_dot)
                stripped = [w.translate(table) for w in abstract]
                List_to_write(filename,stripped)

            else:
                count+=1
                filename = count
                line=fd.readline()
                sline=line.split()
        print("END")
print(count)
```

Αλγόριθμος 4.42: Time text parser

Το αρχείο ερωτημάτων έχει την ίδια μορφή με το αρχείο κειμένων, με την μόνη διαφορά ότι αντί για την ετικέτα "*TEXT", χρησιμοποιεί την ετικέτα "*FIND". Ακολουθείται παρόμοια διαδικασία με το αρχείο κειμένων μόνο που το κείμενο (ερώτημα) αποθηκεύεται σε μία λίστα. Όταν ο αλγόριθμος συναντήσει την ετικέτα "*STOP", που έχει προστεθεί στο τέλος του κειμένου, αποθηκεύει την λίστα με τα ερωτήματα σε ένα αρχείο στο δίσκο και τερματίζει. Η υλοποίηση φαίνεται στο 4.43.

```
import os
import string

queries=[]
line2 =""
sline=[1]
with open('TIME.QUE', 'r') as fd:
    line =fd.readline()
    line =fd.readline()
    line =fd.readline()
    sline = line.split()
    while(len(sline) ==0 or sline[0]!="*STOP"):
        sline[0] = "1"
        while len(sline) == 0 or sline[0]!="*FIND":
            if len(sline)>0 and sline[0] == '*STOP':
                break
            line2 = line2 + line
            line = fd.readline()
            sline = line.split()
        queries.append(line2)
        line2=""
        line = ""
        if len(sline)>0 and sline[0] == '*STOP':
            break
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in queries]

    fw=open('queries.txt','w')
    fw.write(str(stripped))
    fw.close()
```

Αλγόριθμος 4.43: Time query parser

Η διαδικασία ανάλυσης του αρχείου με τα συναφή κείμενα για κάθε ερώτημα είναι ιδιαίτερά πιο απλή. Η μορφή του αρχείου είναι ότι για κάθε ερώτημα η πρώτη γραμμή περιέχει το αναγνωριστικό του ερωτήματος και η δεύτερη τα αναγνωριστικά των συναφών κειμένων για το ερώτημα. Έπειτα ακολουθεί μία κενή γραμμή και συνεχίζει με το επόμενο ερώτημα. Λαμβάνοντας αυτό υπόψιν, η κατασκευή ενός αλγόριθμου που να διαβάζει μόνο τις γραμμές με τα συναφή κείμενα είναι πολύ εύκολη. Τα αναγνωριστικά των συναφών κειμένων για ένα ερώτημα αποθηκεύονται σε μία λίστα, και έπειτα αυτή η λίστα αποθηκεύεται σε μία λίστα από λίστες. Η παραπάνω διαδικασία επαναλαμβάνεται, ώσπου ο αλγόριθμος να συναντήσει την λέξη "END", η οποία έχει προστεθεί στο τέλος του αρχείου με τα συναφή κείμενα. Στην συνέχεια, αποθηκεύει την τελική λίστα με τις λίστες για κάθε ερώτημα σε ένα αρχείο στο δίσκο και τερματίζει. Η υλοποίηση των παραπάνω φαίνεται στο 4.44.

```
relevant=[]
sline2 =[]
with open('TIME.REL', 'r') as fd:
    line =fd.readline()
    sline = line.split()
    while sline[0]!="END":
        line = fd.readline()
        sline = line.split()
        sline2 = sline2 + sline
        line = fd.readline()
        sline = line.split()
```

```
        relevant.append(sline2)
        sline2=[]
        line = fd.readline()
        sline = line.split()
        if len(sline)==0 or sline[0]=="END":
            break
    fw=open('relev.txt','w')
    fw.write(str(relevant))
    fw.close()
```

Αλγόριθμος 4.44: Time relevant texts parser

Τέλος, η συλλογή Time περιλαμβάνει και λίστα με ασήμαντες λέξεις. Η διαδικασία της ανάλυσης του αρχείου με τις ασήμαντες λέξεις είναι ίδια με αυτή για το αρχείο με τα συναφή κείμενα για τα ερωτήματα και η υλοποίηση φαίνεται στο 4.45.

```
stopwords=[]
sline2 =[]
with open('TIME.STP', 'r') as fd:
    line =fd.readline()
    sline = line.split()
    while sline[0]!="#END":
        sline2 = sline2 + sline
        line = fd.readline()
        sline = line.split()
        stopwords.append(sline2)
        sline2=[]
        line = fd.readline()
        sline = line.split()
        if len(sline)==0 or sline[0]=="END":
            break
    fw=open('stopwords.txt','w')
    fw.write(str(stopwords))
    fw.close()
```

Αλγόριθμος 4.45: Time stopword list parser

Chapter 5

Εντοπισμός Ασήμαντων Λέξεων

5.1 Εισαγωγή

Ο όρος stop words (ασήμαντες λέξεις) προτάθηκε πρώτα από τον Parkins (1963) (Flood, 1999), αν και ο πρώτος που αγνόησε συχνές λέξεις στο κείμενο ήταν ο H.P. Luhn(1960) (Luhn, 1960). Οι 16 λέξεις που αγνόησε ο Luhn ήταν οι:

a, an, and, as, at, by, for, from
if, in, of, on, or, the, to, with

Σε αυτό το κεφάλαιο θα γίνει αναφορά στο αν οι ασήμαντες λέξεις (stopwords) πρέπει να αγνοηθούν ή υπό συγκεκριμένες συνθήκες μπορούν να βοηθήσουν την ανάκτηση. Επίσης, θα προταθούν μερικές μέθοδοι που βασίζονται στα μοντέλα που αναφέρθηκαν στο προηγούμενο κεφάλαιο και επιτρέπουν τον εντοπισμό ενός ποσοστού των ασήμαντων λέξεων για μία συλλογή.

5.2 Ασήμαντες Λέξεις-Ανάκτηση Πληροφορίας

Το ερώτημα που πρέπει να απαντηθεί είναι γιατί λέξεις σαν τις παραπάνω πρέπει να αγνοηθούν? Σύμφωνα με τους (Rajaraman and Ullman, 2011) οι ασήμαντες λέξεις έχουν πολύ μεγάλη συχνότητα εμφάνισης στα κείμενα και σαν αποτέλεσμα δεν "κουβαλάνε" κάποιο βαθμό σημαντικότητας. Αξίζει επίσης να σημειωθεί, ότι το παραπάνω είναι φανερό και μέσο του νόμου του Zipf, δηλαδή ότι λέξεις που εμφανίζονται πάρα πολλές φορές σε μία συλλογή (corpus) κειμένων έχουν μικρή νοηματική αξία, σε σχέση με λέξεις που εμφανίζονται σχετικά σπάνια στα κείμενα. Η αφαίρεση των παραπάνω λέξεων βελτιώνει τα αποτελέσματα τις ανάκτησης της πληροφορίας σύμφωνα με τους (Ceri et al., 2013), (Rijsbergen, 1979), (Altmann and Gerlach, 2016).

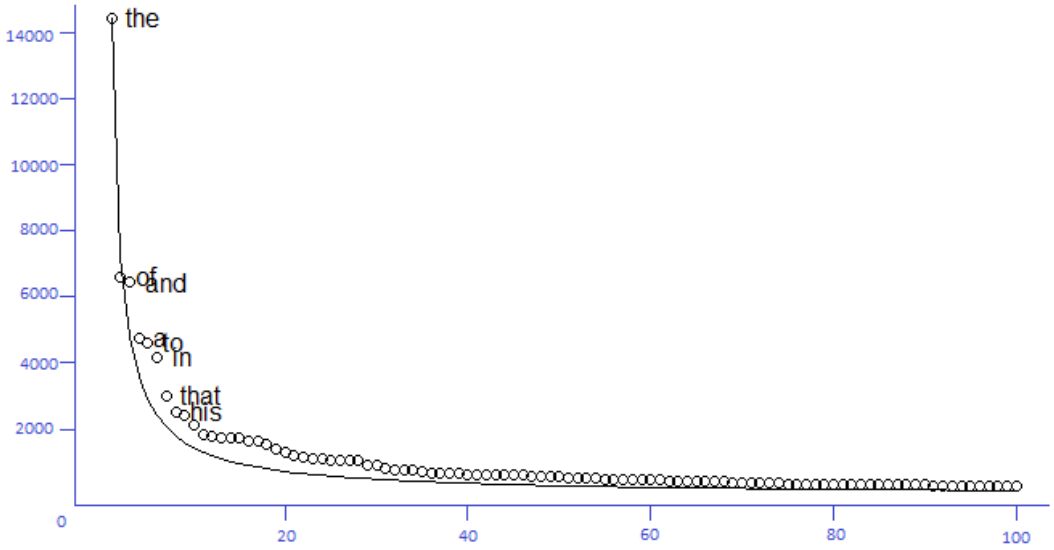

Figure 5.1: Ο νόμος του Zipf

Ωστόσο, πειράματα που έγιναν με τα μοντέλα που προτάθηκαν στο προηγούμενο κεφάλαιο έδειξαν ότι για τις συλλογές και τις μεθόδους που χρησιμοποιήθηκαν, τα αποτελέσματα χειροτέρευσαν όταν αγνοήθηκαν οι ασήμαντες λέξεις. Παρατηρήθηκε, ότι τα γραφήματα κειμένων που παραγόταν, αποτελούνταν στην πραγματικότητα από πολλά μικρά υπογραφήματα, τα οποία ήταν μάλιστα ξένα μεταξύ τους. Αντίθετα, όταν οι ασήμαντες λέξεις δεν αγνοούνταν τα γραφήματα κειμένων αποτελούνταν από υπογραφήματα ενωμένα με ακμές και κόμβους, που λειτουργούσαν ως "γέφυρες" για τα επιμέρους υπογραφήματα. Η υπόθεση που ακολουθήθηκε είναι ότι αυτά τα γραφήματα κειμένων με τις γέφυρες έβγαζαν καλύτερα αποτελέσματα γιατί ήταν πιο συμπαγή, περιείχαν πιο πολύ πληροφορία όσο αφορά την δομή του κειμένου και αποτύπωναν καλύτερα τις συσχετίσεις που είχαν οι λέξεις μεταξύ τους.

5.3 Περιγραφή των μεθόδων

Στη συνέχεια, εξετάζονται μερικές προσεγγίσεις όσο αφορά τον εντοπισμό των ασήμαντων λέξεων μιας συλλογή. Προτείνονται δύο προσεγγίσεις. Η πρώτη βασίζεται στην υπόθεση, ότι οι ασήμαντες λέξεις, επειδή σύμφωνα με τον ορισμό τους έχουν μεγάλη συχνότητα και εμφανίζονται στη πλειοψηφία των κειμένων μιας συλλογής, θα πρέπει να κυριεύουν τον αριθμό των κόμβων που εμφανίζονται στο κύριο πυρήνα των γραφημάτων των κειμένων. Η δεύτερη, υποθέτει ότι είναι δυνατόν χρησιμοποιώντας ένα μικρό κομμάτι των κειμένων στης συλλογής, που θα ονομάζεται δείγμα και μία δι-

αδικασία αξιολόγησης των όρων της συλλογής να ταξινομηθούν οι όροι σε μία λίστα με τους κορυφαίους όρους να αποτελούν την λίστα με τις ασήμαντες λέξεις της συλλογής.

5.3.1 Ασήμαντες Λέξεις στον κύριο πυρήνα

Όπως αναφέρθηκε και στην εισαγωγή, η υπόθεση είναι ότι λόγου της υψηλής συχνότητας των ασήμαντων λέξεων μέσα στα κείμενα, ο κύριος πυρήνας θα πρέπει να αποτελείται κυρίως από αυτά. Η διαδικασία που ακολουθήθηκε είναι:

- Κατασκευάζονται τα γραφήματα κειμένων για κάθε κείμενο της συλλογής.
- Εντοπίζονται οι όροι-κόμβοι που αποτελούν τον κύριο πυρήνα.
- Συγκρίνεται η λίστα των όρων του κύριου πυρήνα με την γνωστή λίστα των ασήμαντων λέξεων. (Είτε με την λίστα που παρέχεται από την βιβλιοθήκη NLTK(Steven Bird and Loper, 2009), είτε με τις ασήμαντες λέξεις της συλλογής που έχουν επιλεχθεί από ειδικούς στην περίπτωση που αυτές είναι διαθέσιμες.)
- Υπολογίζονται διάφορες μετρικές, όπως για παράδειγμα το precision και το recall.

Με την συγκεκριμένη προσέγγιση εντοπίστηκαν κατά μέσο όρο περίπου το 30% - 40% των ασήμαντων λέξεων κάθε κειμένου. Μία σημαντική παρατήρηση είναι ότι από τα αποτελέσματα είναι φανερό ότι ο κύριος πυρήνας αποτελείται και από πολλές λέξεις κλειδιά (keywords), αλλά πιθανόν και από άλλες τυχαίες λέξεις.

Λαμβάνοντας το παραπάνω υπόψιν σε συνδυασμό με τον ορισμό των ασήμαντων λέξεων, δηλαδή ότι οι ασήμαντες λέξεις εμφανίζονται με μικρή ή και μεγάλη συχνότητα στα περισσότερα κείμενα της συλλογής, η παραπάνω μέθοδο επεκτάθηκε ώστε να γίνει έλεγχος για ασήμαντες λέξεις και στο κύριο πυρήνα του συνολικού γραφήματος της συλλογής. Η διαδικασία είναι η ίδια μόνο που αντί για τα γραφήματα κειμένων, χρησιμοποιήθηκε το τελικό συνολικό γράφημα της συλλογής. Με αυτή την επέκταση εντοπίστηκε το 67% του κύριου πυρήνα του συνολικού γραφήματος αποτελούνταν από ασήμαντες λέξεις. Ο αριθμός των λέξεων στον κύριο πυρήνα ήταν συνολικά 260.

5.3.2 Ασήμαντες Λέξεις σε δείγμα της συλλογής

Σκοπός αυτής της μεθόδου είναι να χρησιμοποιηθεί ένα μικρό κομμάτι κειμένων της συλλογής, το οποίο θα ονομάζεται δείγμα, και με μία διαδικασία αξιολόγησης των όρων των κειμένων να γίνει εντοπισμός όσο πιο πολλών ασήμαντων λέξεων της συλλογής είναι δυνατόν. Θεωρητικά ακόμα και με ένα αρκετά μικρό δείγμα είναι δυνατόν

να εντοπιστεί ένα πολύ μεγάλο ποσοστό των ασήμαντων λέξεων της συλλογής. Αυτό οφείλεται στον ορισμό των ασήμαντων λέξεων, που σύμφωνα με τον οποίο, οι ασήμαντες λέξεις θα εμφανίζονται με πολύ μεγάλη συχνότητα στα περισσότερα κείμενα.

Αρχικά επιλέγεται το δείγμα, D που θα χρησιμοποιηθεί για την διαδικασία αξιολόγησης. Στη συνέχεια κατασκευάζονται τα γραφήματα κειμένων του δείγματος και υπολογίζεται ο βαθμός αξιολόγησης S για κάθε όρο. Ο βαθμός αξιολόγησης αποτελείτε από δύο σκέλη. Το πρώτο είναι ένα βάρος, που ονομάζεται βάρος γραφήματος, το οποίο παράγεται από το γράφημα και ισούται με το άθροισμα των επιμέρους βαρών των ακμών που σχετίζονται με το κόμβο, κανονικοποιημένο με το μέγεθος του παραθύρου $\mathrm{W_{size}}$ του μοντέλου που χρησιμοποιείται. Το δεύτερο βάρος, το οποίο θα ονομάζεται βάρος συμμετοχής W_p, είναι παρόμοιο με το document frequency ενός όρου. Δηλαδή, κάθε κείμενο τους δείγματος αναλύεται και την πρώτη φορά που συναντάται κάποιος όρος, έστω i, το αντίστοιχο βάρος συμμετοχής W_{p_i} του αυξάνεται κατά ένα. Το βάρος συμμετοχής πολλαπλασιάζεται με μία μεταβλητή α, μεγαλύτερη της μονάδας, με σκοπό να μειωθεί η επιρροή των keywords στην κατάταξη. Τελικά, ο βαθμός αξιολόγησης δίνεται από το παρακάτω τύπο:

$$S_i = \frac{\sum_j^N e_{i,j}}{W_{size}} + \alpha \times W_{p_i} \tag{5.1}$$

Όπου το $\mathrm{S_i}$ δηλώνει το βαθμό αξιολόγησης, το $e_{i,j}$ είναι οι ακμές που περιλαμβάνουν τους κόμβους i, j και το W_{size} είναι το μέγεθος του παραθύρου που χρησιμοποιείται.

5.3.3 Ασήμαντες Λέξεις στο ευρετήριο

Όπως έχει ήδη αναφερθεί, οι ασήμαντες λέξεις χαρακτηρίζονται από υψηλή συχνότητα εμφάνισης στα κείμενα. Αυτό έχει ως αποτέλεσμα, οι κόμβοι, στο συνολικό γράφημα, που αντιστοιχούν σε ασήμαντες λέξεις να έχουν πολλούς περισσότερους γείτονες (ακμές) συγκριτικά με τους άλλους κόμβους. Λαμβάνοντας αυτό υπόψιν σε συνδυασμό με τον τρόπο που υπολογίζονται τα βάρη από το συνολικό γράφημα (3.7), και συμπεραίνετε εύκολα ότι τα βάρη που αντιστοιχούν σε ασήμαντες λέξεις θα πρέπει να είναι αρκετά χαμηλότερα σε σχέση με τους υπόλοιπους κόμβους. Σύμφωνα με αυτό, είναι δυνατόν εξετάζοντας το ευρετήριο που παράγεται από κάποια προτεινόμενη μέθοδο να εντοπιστούν κάποιες ασήμαντες λέξεις. Ένα σημαντικό πλεονέκτημα αυτής της μεθόδου είναι ότι χρειάζεται πολύ λίγη επιπλέον εργασία αφού το ευρετήριο παράγεται, έτσι και αλλιώς για την ανάκτηση της πληροφορίας.

5.4 Περιγραφή υλοποίησης

5.4.1 Εισαγωγή

Σε αυτή την ενότητα θα περιγραφεί η διαδικασία με την οποία υλοποιήθηκαν οι μέθοδοι, που με τις οποίες ανιχνεύονται ασήμαντες λέξεις μέσα στα κείμενα της συλλογής. Οι μέθοδοι εφαρμόστηκαν σε δύο συλλογές, την CF Collection (Shaw et al., 1991), για την οποία χρησιμοποιήθηκε η λίστα των αγγλικών ασήμαντων λέξεων του NTLK (Steven Bird and Loper, 2009), καθώς και για την Time Collection (*Test collections* n.d.), οι οποία παρείχε δικιά της λίστα με ασήμαντες λέξεις, οι οποία έχει εξεταστεί από εμπειρογνώμονες. Στον κώδικα που ακολουθεί έχει χρησιμοποιηθεί ενδεικτικά η λίστα ασήμαντων λέξεων της βιβλιοθήκης NLTK.

5.4.2 Ασήμαντες Λέξεις στον κύριο πυρήνα

Η υλοποίηση της συγκεκριμένης μεθόδου είναι εξαιρετικά απλή, αφού ο κύριος όγκος της εργασίας που χρειάζεται εκτελείται ήδη από τις μεθόδους που λαμβάνουν υπόψιν τους σημαντικούς κόμβους, όπως αυτές έχουν περιγραφτεί στο προηγούμενο κεφάλαιο. Επειδή, χρειάζεται ο κύριος πυρήνας, αυτή η μέθοδο είναι συμβατή μόνο με συνδυασμούς μεθόδων που περιλαμβάνουν τον εντοπισμό του κύριου πυρήνα (Main-Core). Αρχικά, ο αλγόριθμος εκτελείται κανονικά ως και το σημείο που εντοπίζονται οι σημαντικοί κόμβοι. Αφού εντοπιστούν οι κόμβοι που αποτελούν τον κύριο πυρήνα, καλείται μία συνάρτηση, που ονομάζεται stopwordsStats και δέχεται ως είσοδο την λίστα με τους όρους-κόμβους του κύριου πυρήνα, την λίστα με τους όρους του κειμένου και το όνομα του αρχείου. Η συνάρτηση συγκρίνει τους όρους-κόμβους του κύριου πυρήνα με την λίστα των ασήμαντων λέξεων που παρέχεται. Τέλος, υπολογίζει τις μετρικές ακρίβεια-ανάκληση και αποθηκεύει τα αποτελέσματα σε ένα αρχείο στην μνήμη.

Για την επέκταση στο συνολικό γράφημα που αναφέρθηκε, αφού ολοκληρωθεί η κατασκευή του συνολικού γραφήματος της συλλογής, εντοπίζεται ο κύριος πυρήνας του, με την βοήθεια της συνάρτησης k_core της βιβλιοθήκης networkx, και στην συνέχεια γίνεται κλήση της συνάρτησης stopwordsStats που αναφέρθηκε παραπάνω.

```
def stopwordsStats(kcore, term_list, file):
    stopwords = stopwords.words('english')
    stopword_count = 0
    stopwords_in_file = 0
    for i in kcore.nodes:
        if term_list[i] in stopwords:
            stopword_count += 1
    for i in term_list:
        if i in stopwords:
```

```
            stopwords_in_file += 1
    stopwords_in_file_per = float(stopword_count/stopwords_in_file)
    stopwords_per = float(stopword_count/len(kcore.nodes))
    fw=open('stopwords_stats.txt','a')
    string_to_write = "File " + str(file) + " stopwords in kcore
        percentage : " + str(stopwords_per) + " and stopwords
        percentage in file: "+ str(stopwords_in_file_per) + "\n"
    fw.write(string_to_write)
    fw.close()
    fw=open('stopwords_kcore_stats.txt','a')
    fw.write(str(stopwords_per))
    fw.close()
    fw=open('stopwords_file_stats.txt','a')
    fw.write(str(stopwords_in_file_per))
    fw.close()
```

Αλγόριθμος 5.1: stopwordsStats Function.

5.4.3 Ασήμαντες Λέξεις σε δείγμα της συλλογής

Εισαγωγή

Σε αυτό το υποκεφάλαιο θα περιγραφεί η υλοποίηση για τον εντοπισμό ασήμαντων λέξεων χρησιμοποιώντας μόνο ένα δείγμα κειμένων από την συλλογή. Ο αλγόριθμος έχει υλοποιηθεί με τέτοιο τρόπο, ώστε να χρειάζεται μόνο η κλήση κάποιων συναρτήσεων σε συγκεκριμένα σημεία των αλγορίθμων ανάκτησης πληροφορίας που αναφέρθηκαν στο προηγούμενο κεφάλαιο. Αυτό σημαίνει ότι οι δύο αλγόριθμοι μπορούν να τρέχουν παράλληλα. Ωστόσο όμως, επειδή ο αλγόριθμος εντοπισμού των ασήμαντων λέξεων είναι ευαίσθητος στις τιμές του μεγέθους του παραθύρου, τα πειράματα εκτελέστηκαν μόνο με την χρήση της μεθόδου σταθερού παραθύρου για όλη την συλλογή. Επιλέχθηκε η συγκεκριμένη μέθοδο για την ευελιξία και την ευκολία που προσφέρει στην επιλογή μεγέθους παραθύρου. Λαμβάνοντας υπόψιν τα παραπάνω, η υλοποίηση για τον εντοπισμό ασήμαντων λέξεων σε δείγμα της συλλογής χωρίζεται σε 4 στάδια.

Καθορισμός του δείγματος κειμένων από τη συλλογή

Υπό κανονικές συνθήκες το δείγμα κειμένων της συλλογής επιλέγεται τυχαία. Για πειραματικούς λόγους, ωστόσο, είναι απαραίτητη η ικανότητα αναπαραγωγής του ίδιου δείγματος κειμένων και αποτελεσμάτων. Σύμφωνα με αυτό, και επειδή τα ονόματα των κειμένων είναι αριθμημένα, χρησιμοποιήθηκε μία συνάρτηση κατακερματισμού, η οποία χωρίζει τα κείμενα σε κουβάδες(λίστες) σύμφωνα με το όνομά τους. Η συνάρτηση ονομάζεται BucketHash, δέχεται ως είσοδο μία λίστα με τα ονόματα των κειμένων και επιστρέφει τα χωρισμένα σε λίστες κείμενα. Η συνάρτηση καλείται στην αρχή του αλγορίθμου της ανάκτησης πληροφορίας. Πραγματοποιήθηκαν πειράματα με τέσσερις, έξι και οκτώ κουβάδες.

```
def BucketHash(file_list):
    bucket_0 = []
    bucket_1 = []
    bucket_2 = []
    bucket_3 = []
    bucket_4 = []
    bucket_5 = []
    bucket_6 = []
    bucket_7 = []
    for name in file_list:
        name = name[0]
        name = int(name[9:])
        hashed_name = name %8
        if hashed_name == 0:
            bucket_0.append(name)
        elif hashed_name == 1:
            bucket_1.append(name)
        elif hashed_name == 2:
            bucket_2.append(name)
        elif hashed_name == 3:
            bucket_3.append(name)
        elif hashed_name == 4:
            bucket_4.append(name)
        elif hashed_name == 5:
            bucket_5.append(name)
        elif hashed_name == 6:
            bucket_6.append(name)
        elif hashed_name == 7:
            bucket_7.append(name)
        else:
            print("Something went wrong\n")
    return bucket_0, bucket_1, bucket_2, bucket_3, bucket_4, bucket_5,
        bucket_6, bucket_7
```

Αλγόριθμος 5.2: Bucket hashing.

Αφού ολοκληρωθεί η διαδικασία χωρισμού των κειμένων σε κουβάδες, επιλέγεται ένας κουβάς ως το δείγμα κειμένων της συλλογής.

Υπολογισμός του βαθμού αξιολόγησης

Για κάθε κείμενο του δείγματος, αμέσως μετά τον υπολογισμό του πίνακα γειτνίασης, καλείται μία συνάρτηση, που ονομάζεται calculateSummationMatrix και υπολογίζει το άθροισμα των ακμών των όρων για το συγκεκριμένο κείμενο. Ταυτόχρονα, γίνεται κανονικοποίηση με το μέγεθος του παραθύρου. Οι κανονικοποιήμενοι πίνακες αθροίσματος αποθηκεύονται μαζί με τα ονόματα των κειμένων που τους παρήγαγαν σε μία λίστα, που ονομάζεται file_sum_mat.

```
def calculateSummationMatrix(adj_matrix, file,file_sum_mat,terms,
    window):
    adj_mat_sum = adj_matrix.sum(axis = 1, dtype = 'float')
    for i in range(0,len(adj_mat_sum)):
        adj_mat_sum[i] = float(adj_mat_sum[i]/(window-1))
        if isinstance( adj_mat_sum[i], float):
            adj_mat_sum[i] = math.ceil(adj_mat_sum[i])
    sum_mat = list(zip(terms,adj_mat_sum))
    sorted_sum_mat = sorted(sum_mat, key = lambda x: x[1],reverse=True)
```

```
    file_sum_mat.append([file,sorted_sum_mat])
    return file_sum_mat
```

Αλγόριθμος 5.3: File summation matrix calculation.

Το επόμενο στάδιο είναι ο υπολογισμός του βαθμού αξιολόγησης για κάθε όρο του δείγματος συνολικά. Οπότε αφού δημιουργηθεί το συνολικό γράφημα, καλείται μία συνάρτηση με όνομα calculateStopwordWeight.

```
def calculateStopwordWeight(file_sum_mat, collection_terms):
    a = 5
    collection_sum_mat = numpy.zeros(shape=(1,len(collection_terms)))
    collection_vote_mat = numpy.zeros(shape=(1,len(collection_terms)))
    weight_matrix = numpy.zeros(shape=(1,len(collection_terms)))
    for i in range(1,len(file_sum_mat)):
        file_terms_mat = file_sum_mat[i][1]
        collection_sum_mat, collection_vote_mat = calculateWeight(
            collection_sum_mat, collection_vote_mat, file_terms_mat,
            collection_terms)
        #collection_matrices[0] = collection_sum_mat |
            collection_matrices[1] = collection_vote_mat
    for x in range(0,len(collection_sum_mat[0][:])):
        weight_matrix[0][x] = collection_sum_mat[0][x] +
            collection_vote_mat[0][x]*a
    indices = [x for x in range(len(weight_matrix[0][:]))]
    zipped_weight_matrix = list(zip(indices,*weight_matrix))
    sorted_weight_matrix = sorted(zipped_weight_matrix, key = lambda x:
        x[1],reverse=True)
    return sorted_weight_matrix
```

Αλγόριθμος 5.4: Stopword weight calculation.

Η συνάρτηση διατρέχει την λίστα file_sum_mat και αναλαμβάνει με την βοήθεια μίας δεύτερης συνάρτησης ονομαζόμενης calculateWeight να υπολογίσει τα δύο σκέλη του βαθμού αξιολόγησης για κάθε όρο του δείγματος.

```
def calculateWeight(collection_sum_mat, collection_vote_mat,
    file_terms_mat, collection_terms):
    for x in range(0,len(file_terms_mat)):
        index = collection_terms[file_terms_mat[x][0]]
        collection_sum_mat[0][index] += file_terms_mat[x][1]
        collection_vote_mat[0][index] +=1
    return collection_sum_mat, collection_vote_mat
```

Αλγόριθμος 5.5: calculateWeight function.

Κάθε βαθμός αξιολόγησης και όρος που το παρήγαγε, αποθηκεύονται σε μία λίστα. Τέλος, η λίστα ταξινομείται σε φθίνουσα σειρά.

Ανάλυση των αποτελεσμάτων

Το τελευταίο στάδιο είναι η ανάλυση της παραπάνω λίστας και η αξιολόγηση των αποτελεσμάτων. Αυτό επιτυγχάνεται με την βοήθεια μίας συνάρτησης παρόμοιας με

αυτή του προηγούμενου υποκεφαλαίου. Διατρέχεται η λίστα μέχρι ένα σημείο και μετριούνται πόσοι κορυφαίοι όροι αποτελούν ασήμαντες λέξεις. Τέλος, υπολογίζονται μετρικές όπως ακρίβεια-ανάκληση.

```
def stopwordsStats(stopword_matrix, collection_terms):
    stopwords = stopwords.words('english')
    stopword_count = 0
    val_list = list(collection_terms.values())
    key_list = list(collection_terms.keys())
    new_matrix = stopword_matrix[0:1000][0:1000]
    for i in range(0,1000):
        position = val_list.index(new_matrix[i][0])
        if key_list[position] in stopwords:
            stopword_count += 1
    stopwords_in_collection = float(stopword_count/340)
    stopwords_per = float(stopword_count/len(new_matrix))
    string_to_write = " stopwords in matrix percentage : " + str(
       stopwords_per) + " and stopwords percentage in collection: "+
       str(stopwords_in_collection) + " sta 1000\n"
    print(string_to_write)
```

Αλγόριθμος 5.6: Sampling result evaluation function.

5.4.4 Ασήμαντες Λέξεις στο ευρετήριο

Αρχικά, πρέπει να διαβαστεί το επιθυμητό ευρετήριο. Αυτό επιτυγχάνεται με την χρήση μίας συνάρτησης, η οποία αναλαμβάνει να διαβάσει το ευρετήριο από την μνήμη και εξάγει χρήσιμη πληροφορία από αυτό. Η συνάρτηση ονομάζεται load_inv_index και δέχεται ως είσοδο το όνομα ενός από τα παραγόμενα ευρετήρια του αλγορίθμου ανάκτησης. Το ευρετήριο είναι πρακτικά ένα αρχείο csv, όπου οι στήλες του χωρίζονται με τον χαρακτήρα ";". Για διευκόλυνσή, το ευρετήριο διαβάζεται με την βοήθεια μίας βιβλιοθήκης που ονομάζεται csv. Επιστρέφονται δύο λίστες που περιέχουν τα ονόματα των όρων και τα αντίστοιχα βάρη τους.

```
def load_inv_index(*args):
    arg = list(args)
    if not arg:
        invindex = 'inverted index.dat'
    else:
        invindex = arg[0]
    ids = []
    trms = []
    W = []
    plist = []
    with open(invindex, 'r') as csvf:
        reader = csv.reader(csvf, delimiter=";")
        for row in reader:
            if row[0] not in ids:
                ids.append(row[0])
                trms.append(row[1])
                W.append(row[2])
                plist.append(row[3].split(','))
    csvf.close()
    # print(len(ids))
```

```
    return ids, trms, W, plist
```

Αλγόριθμος 5.7: Reading the inverted index

Στην συνέχεια, οι δύο επιστρεφόμενες λίστες συνδυάζονται και ταξινομούνται φθίνουσα, σύμφωνα με τα βάρη.

```
zipped_list = list(zip(trms,W))
sorted_zipped_list = sorted(zipped_list, key = lambda x: x[1],reverse=
    False)
```

Αλγόριθμος 5.8: Sorting the two lists

Τέλος, όπως και στη προηγούμενες μεθόδους εντοπισμού ασήμαντων λέξεων χρησιμοποιείται μία συνάρτηση που ονομάζεται stopwordsStats, η οποία συγκρίνει την παραπάνω λίστα με την λίστα των ασήμαντων λέξεων και υπολογίζει της μετρικές ακρίβεια-ανάκληση.

```
def stopwordsStats(stopword_matrix):
    new_matrix = []
    stopwords_list =  stopwords.words('english')
    stopword_count = 0
    for i in range(0,200):
        new_matrix.append(stopword_matrix[i][0])
    for i in range(0,200):
        if new_matrix[i] in stopwords_list:
            stopword_count += 1
    stopwords_in_collection = float(stopword_count/340)
    stopwords_per = float(stopword_count/len(new_matrix))
    string_to_write = " stopwords in matrix percentage : " + str(
       stopwords_per) + " and stopwords percentage in collection: "+
       str(stopwords_in_collection) + " sta 200\n"
    print(string_to_write)
```

Αλγόριθμος 5.9: stopwordStats function

Chapter 6

Αποτελέσματα

6.1 Εισαγωγή

Στο σημείο αυτό γίνεται η παρουσίαση των αποτελεσμάτων καθώς και ο σχολιασμός τους. Και τελικά εντοπίζονται μερικά αρνητικά σημεία του μοντέλου και προτείνονται τρόποι και ιδέες προς μελέτη και βελτίωση του.

Όμως, προτού γίνει η παρουσίαση θα πρέπει να αναλυθεί το πειραματικό πλαίσιο στο οποίο έγιναν οι μετρήσεις καθώς και τι εκφράζουν αυτές. Αρχικά, για κάθε ερώτημα μίας συλλογής υπολογίζεται η μέση ακρίβεια για κάθε μια από τις υλοποιήσεις. Σαν βάση σύγκρισης χρησιμοποιείται η υλοποίηση του Set-Based μοντέλου. Ο στόχος της πειραματικής διαδικασίας είναι να εντοπιστεί ο αριθμός ερωτημάτων, που κάθε άλλη υλοποίηση απαντάει με μεγαλύτερη ακρίβεια σε σχέση με αυτό. Η σχέση αυτή εκφράζεται με την έννοια της διαφοράς τους. Άρα για κάθε ερώτημα υπολογίζεται η διαφορά του εκάστοτε μοντέλου με το Set-Based μοντέλο. Αν η τιμή της διαφοράς είναι μεγαλύτερη του 0 τότε το μοντέλο απαντάει καλύτερα από ότι το Set-Based μοντέλο, αν η διαφορά είναι 0 ίδια και αν είναι μικρότερη του 0 χειρότερα.

6.2 Πειραματικά αποτελέσματα και σχολιασμός

Συνολικά, όπως έχει ήδη αναφερθεί, τα πειράματα εκτελέστηκαν πάνω σε τέσσερις συλλογές. Πριν παρουσιαστούν τα αποτελέσματα, θα γίνει αναφορά σε μερικά τεχνικά χαρακτηριστικά των συλλογών. Επίσης, θα γίνει αναφορά και στην ονομασία των πειραμάτων.

Η πρώτη συλλογή που χρησιμοποιήθηκε είναι η Cystic Fibrosis (Shaw et al., 1991), οι οποία περιέχει 1209 κείμενα και 100 ερωτήματα. Συνολικά έχει μέγεθος 1.47 Megabyte. Στην συγκεκριμένη συλλογή έγιναν τα περισσότερα πειράματα και με όλες τις μεθόδους που έχουν προταθεί. Έγινε προεπεξεργασία στην συλλογή αφαιρώντας σημεία στίξης (εκτός από τις τελείες). Επίσης, έγιναν πειράματα χωρίς να αφαιρεθούν

Όνομα	Docs	Queries	Size (in MB)	Stopwords	K-Core Decomposition	K-Core With Windows	Stopword Detection
CF	1239	100	1.47	Yes & No	Yes	Yes	No
NPL	11.429	93	3.1	Yes	No	Yes	No
Time	423	83	1.5	Yes	No	No	Yes
Cran	1400	225	1.6	Yes	No	No	No

Table 6.1: Collections and types of experiments used on each one.

οι ασήμαντες λέξεις αλλά και με αφαίρεση τους.

Η δεύτερη συλλογή που χρησιμοποιήθηκε ήταν η NPL Collection (*Test collections* n.d.), η οποία περιέχει 11,429 κείμενα και 93 ερωτήματα. Το μέγεθος της είναι συνολικά 3.1 Megabyte και έγινε προεπεξεργασία αφαιρώντας τα σημεία στίξης (εκτός από τις τελείες). Τα πειράματα εκτελέστηκαν χωρίς την αφαίρεση των ασήμαντων λέξεων.

Η τρίτη συλλογή ήταν η Time Collection (*Test collections* n.d.) που περιέχει 423 κείμενα, 83 ερωτήματα και έχει μέγεθος 1.5 Megabyte. Ιδιαίτερα σημαντικό είναι ότι η συγκεκριμένη συλλογή παρέχει λίστα ασήμαντων λέξεων (340 λέξεις συνολικά) και στην συγκεκριμένη συλλογή έγιναν τα πειράματα με τις μεθόδους εντοπισμού ασήμαντων λέξεων. Έγινε προεπεξεργασία στην συλλογή αφαιρώντας σημεία στίξης (εκτός από τις τελείες). Επίσης, έγιναν πειράματα χωρίς να αφαιρεθούν οι ασήμαντες λέξεις αλλά και με αφαίρεση τους.

Η τελευταία συλλογή ήταν η Cranfield Collection (*Test collections* n.d.), που αποτελείται από 1,400 κείμενα, 225 ερωτήματα και έχει μέγεθος 1.6 Megabyte. Η προεπεξερασία που εκτελέστηκε είναι η αφαίρεση των σημείων στίξης. Όπως και στις άλλες συλλογές τα πειράματα έγιναν διατηρώντας τις ασήμαντες λέξεις.

Ο πίνακας 6.1 δείχνει μερικά χαρακτηριστικά των πειραμάτων σε σχέση με τις συλλογές.

Τα ονόματα των πειραμάτων ακολουθούν το παρακάτω πρότυπο:

$$<\text{b}><\text{h}><\text{c}><\text{s}><\text{p}>$$

Όπου b είναι το μπόνους που δίνεται στα βάρη των κόμβων που ανήκουν σε σημαντικούς κόμβους, h είναι το κατώφλι απαλοιφής στην περίπτωση που χρησιμοποιείται απαλοιφή ακμών, c είναι το μέγεθος του σταθερού παραθύρου σε λέξεις, s είναι το μέγεθος του παραθύρου τις παραγράφου σε λέξεις και p είναι το ποσοστό του μεγέθους του κειμένου που θα χρησιμοποιηθεί ως μέγεθος παραθύρου.

Λόγου του μεγάλου αριθμού των πειραμάτων (εκτελέστηκαν συνολικά 350+ πειράματα), στα διαγράμματα έχουν παραλειφθεί οι ακριβές ονομασίες των πειραμάτων. Αντίθετα, θα δίνονται περιγραφές για τα είδη των πειραμάτων που εκτελούνται και θα σημειώνονται παρατηρήσεις για τα σύνολα των πειραμάτων. Τα πειράματα έχουν

χωρισθεί σε σετ ανάλογα τον σκοπό εκτέλεσης τους. Αυτό σημαίνει ότι διάφορες μέθοδοι χρησιμοποιήθηκαν σε παραπάνω από ένα σετ, και πολλές φορές τα καλύτερα αποτελέσματα μίας μεθόδου δεν βρίσκονται στο πρώτο σετ που εμφανίστηκε η μέθοδο. Στο τέλος της παρουσίασης των αποτελεσμάτων κάθε συλλογής, θα περιλαμβάνεται και ένας πίνακας με τα καλύτερα αποτελέσματα κάθε μεθόδου.

6.2.1 Αποτελέσματα στην CF συλλογή

Η παρουσίαση των αποτελεσμάτων θα ξεκινήσει από τη CF συλλογή. Στην συγκεκριμένη συλλογή έγιναν πειράματα και συγκρίσεις με όλες τις διαθέσιμες μεθόδους. Επίσης, έγιναν πειράματα χρησιμοποιώντας τεχνικές διατήρησης σημαντικών κόμβων, με το ολισθούμενο παράθυρο των (Rousseau and Vazirgiannis, 2013a) και με την αφαίρεση και συμπερίληψη ασήμαντων λέξεων.

GSB - K-Core - Σταθερό παράθυρο

Τα πρώτα πειράματα που εκτελέστηκαν. Σκοπός τους ήταν να γίνει μία πρώτη εκτίμηση της χρήσης των παραθύρων σε σχέση με τις μεθόδους που περιγράφονται από τους Καλογερόπουλο, Δούκα, Καναβό και Μακρή (Kalogeropoulos et al., 2020). Χρησιμοποιήθηκαν οι μέθοδοι GSB, Maincore, Density, Corerank και σταθερού παραθύρου για όλη την συλλογή (Constant Window). Οι τιμές του παραθύρου κυμάνθηκαν από 7 έως 44, δίνοντας έμφαση στις τιμές κάτω από 16. Δοκιμάστηκαν επίσης οι ακραίες τιμές των 60 και 100. Συνολικά εκτελέστηκαν 18 πειράματα και τα αποτελέσματα φαίνονται στο διάγραμμα 6.1. Στο κάτω μέρος του διαγράμματος φαίνονται τα μεγέθη των διάφορων παραθύρων που χρησιμοποιήθηκαν.

Οι πτώσεις απόδοσης από την μέθοδο σταθερού παραθύρου συλλογής ήταν για μεγάλες τιμές μεγέθους παραθύρου. Παρατηρήθηκε ότι για τα κείμενα τις συλλογής CF, η απόδοση ήταν μέγιστη με παράθυρο μεγέθους 9 στα 79 από τα 100 ερωτήματα καλύτερα από το Set-Based μοντέλο. Οι μεγάλες πτώσεις αντιστοιχούν σε παράθυρα μεγέθους 36, 40, 44, 60, 100 με αυτή την σειρά. Μία ακόμα σημαντική παρατήρηση είναι ότι η απόδοση, αφού περάσει το μέγιστο, δεν μειώνεται συνεχώς αλλά μερικές φορές αυξάνεται. Αυτό μας οδηγεί στο συμπέρασμα ότι για την απόδοση δεν παίζει ρόλο μόνο το μέγεθος παραθύρου, αλλά και τα σημεία που κόβεται το κείμενο από τα παράθυρα. Επίσης, ας γίνει αναφορά στο γεγονός ότι ακόμα και με παράθυρο μεγέθους 100, που στην συλλογή CF είναι μεγαλύτερο από τα πιο πολλά κείμενα, η μέθοδος σταθερού παραθύρου συλλογής αποδίδει καλύτερα από τις μεθόδους που έχουν περιγραφεί από τους (Kalogeropoulos et al., 2020). Τέλος, πρέπει να σημειωθεί, αν και δεν φαίνεται στο διάγραμμα, ότι αν το μέγεθος του παραθύρου συλλογής είναι ίσο ή μεγαλύτερο

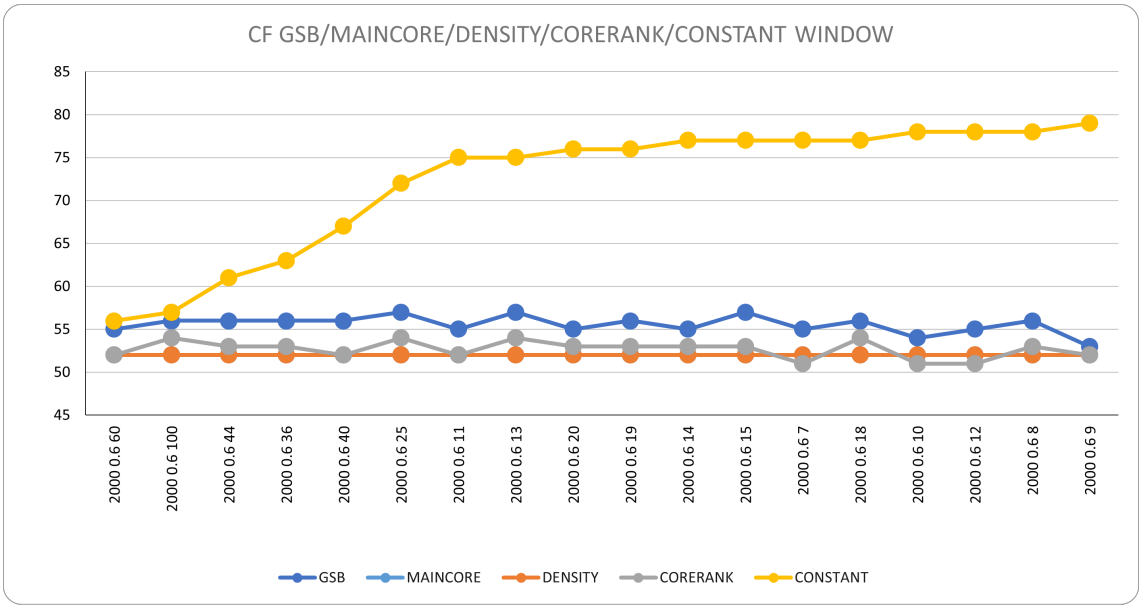

Figure 6.1: GSB - K-Core - Σταθερό παράθυρο συλλογής

από το μέγεθος του μεγαλύτερου κειμένου της συλλογής, τότε η μέθοδο σταθερού παραθύρου συλλογής συμπεριφέρεται σαν το μοντέλο GSB, αφού για κάθε κείμενο πλέον υπάρχει μόνο ένα παράθυρο, το οποίο μάλιστα περιλαμβάνει όλο το κείμενο.

GSB - K-Core - Σταθερό παράθυρο κειμένου

Όπως αναφέρθηκε και στο κεφάλαιο 4.5.7, η μέθοδο σταθερού παραθύρου συλλογής παρουσιάζει κάποια προβλήματα, κυρίως στην ευελιξία της, αφού κάθε κείμενο ασχέτου μεγέθους χωρίζεται με το ίδιο μέγεθος παράθυρου. Για αυτό τον λόγο ήταν απαραίτητη η επινόηση της μεθόδου σταθερού μεγέθους παραθύρου για κάθε κείμενο. Το σύνολο των πειραμάτων που ακολουθούν εκτελέστηκαν, με σκοπό να εκτιμηθεί η απόδοση της συγκεκριμένης μεθόδου. Συνολικά εκτελέστηκαν 12 πειράματα σε αυτό το σύνολο πειραμάτων. Οι τιμές του ποσοστού του μεγέθους του κειμένου που χρησιμοποιείτε ως μέγεθος παραθύρου, κυμαίνονται από 0.005(0.5%) έως 0.1(10%). Χρησιμοποιήθηκαν οι μέθοδοι GSB, Maincore, Density, Corerank και σταθερού παραθύρου για κάθε κείμενο (Constant Window %). Τα αποτελέσματα φαίνονται στο διάγραμμα 6.2. Στο κάτω μέρος του διαγράμματος φαίνονται τα ποσοστά των κειμένων που χρησιμοποιήθηκαν ως παράθυρα.

Όπως και στην περίπτωση του σταθερού παραθύρου συλλογής, τα αποτελέσματα είναι εξαιρετικά καλύτερα από της μεθόδους των (Kalogeropoulos et al., 2020). Για τα κείμενα τις συλλογής CF, παρατηρείται ότι καλύτερη απόδοση, ως τώρα, είχε το ποσοστό μεγέθους κειμένου 0.0105(1.05%), απαντώντας σε 80 από τα 100 ερωτήματα, καλύτερα από το Set-Based μοντέλο. Ακόμα, φαίνεται ότι η μέθοδος σταθερού παρα-

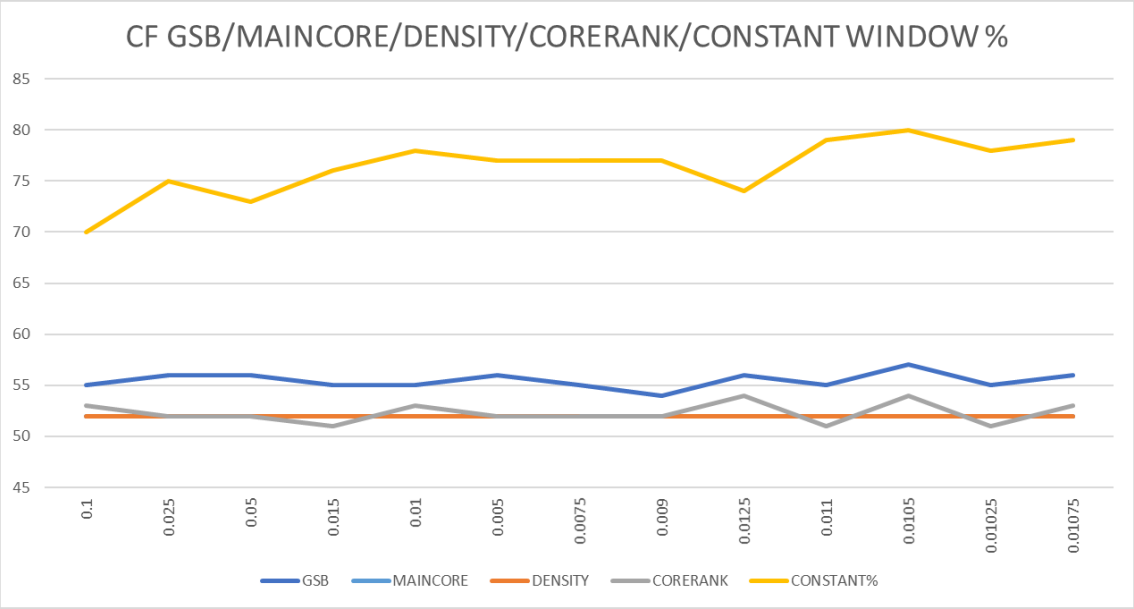

Figure 6.2: GSB - K-Core - Σταθερό παράθυρο κειμένου

θύρου για κάθε κείμενο, να είναι πιο ανθεκτική σε μεγάλες διακυμάνσεις τιμών των παραθύρων σε σχέση με την μέθοδο σταθερού παραθύρου συλλογής. Αυτό συμπεραίνετε από το γεγονός ότι η μείωση της απόδοσης, για παράδειγμα, από την καλύτερη περίπτωση σε σχέση με την περίπτωση που χρησιμοποιείται το 10% του μεγέθους των κειμένων δεν είναι όσο ραγδαία όσο, όταν χρησιμοποιείτε παράθυρο συλλογής μεγέθους 9 και 100, ακόμα και αν και στις δύο περιπτώσεις η αναλογία αύξησης μεγέθους είναι η ίδια. Πρέπει να σημειωθεί, ωστόσο ότι οι καλύτερες περιπτώσεις και των δύο μεθόδων έχουν παραπλήσια αποτελέσματα. Τέλος, παρατηρήθηκε ότι στις καλύτερες περιπτώσεις τα μεγέθη των παραθύρων που παράγονταν βρισκόταν σχεδόν πάντα στο κατώφλι του μεγέθους του παραθύρου, με αποτέλεσμα να θέτονταν ίσο με 5. Αυτό σημαίνει ότι προσέγγιζαν την μέθοδο σταθερού παραθύρου συλλογής για μέγεθος ίσο με 5. Η συγκεκριμένη παρατήρηση θα επεκταθεί περισσότερο σε παρακάτω υποκεφάλαια.

GSB - K-Core - Σταθερό παράθυρο κειμένου - Σταθερό παράθυρο παραγράφου

Σε αυτή την ομάδα πειραμάτων, έγινε προσπάθεια σύγκρισης της μεθόδου σταθερού παραθύρου παραγράφου με τις μεθόδους των Καλογερόπουλο, Δούκα, Καναβό και Μακρή (Kalogeropoulos et al., 2020). Παράλληλα, συνεχίστηκε ο πειραματισμός με την μέθοδο σταθερού παραθύρου κειμένου. Συνολικά εκτελέστηκαν 24 πειράματα και χρησιμοποιήθηκαν οι μέθοδοι GSB, Maincore, Density, Corerank, σταθερού παραθύρου για κάθε κείμενο (Constant Window %) και σταθερού παραθύρου παραγράφου (Sentence - Paragraph Window). Οι τιμές του παραθύρου παραγράφου κυμάνθηκαν

από 150 έως και 200, αλλά κυρίως χρησιμοποιήθηκε το παράθυρο μεγέθους 150. Αντίθετα, δόθηκε έμφαση στο πειραματισμό με τις τιμές των μεταβλητών σημαντικότητας. Επίσης, όπως αναφέρθηκε επεκτάθηκε ο πειραματισμός με την μέθοδο σταθερού παραθύρου για κάθε κείμενο με τιμές ποσοστών από 0.01055 (1.055%) έως και 0.02357 (2.357%). Τα αποτελέσματα φαίνονται στο διάγραμμα 6.3. Στο κάτω μέρος του διαγράμματος φαίνονται τα ποσοστά των κειμένων που χρησιμοποιήθηκαν ως παράθυρα, το μέγεθος του παραθύρου παραγράφου και οι μεταβλητές σημαντικότητας με αυτή την σειρά. Υπενθυμίζεται ότι η μεταβλητή σημαντικότητας α αντιστοιχεί στο κομμάτι της πρότασης, ενώ η η μεταβλητή σημαντικότητας β αντιστοιχεί στο κομμάτι της παραγράφου.

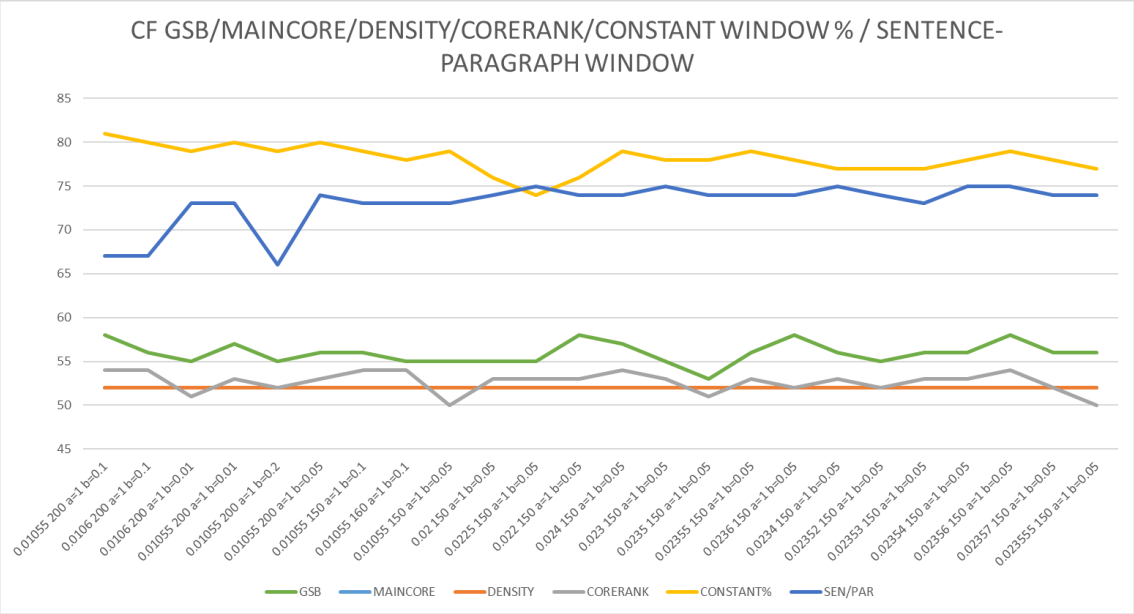

Figure 6.3: GSB - K-Core - Σταθερό παράθυρο κειμένου - Σταθερό παράθυρο παραγράφου

Αρχικά, θα γίνει αναφορά σε μερικές παρατηρήσεις όσο αφορά το σταθερό παράθυρο κειμένου. Η βέλτιστη τιμή ποσοστού, στα πειράματα που εκτελέστηκαν συνολικά, στην συλλογή CF, είναι ίση με 0.01055 (1.055%), απαντώντας καλύτερα σε 81 από τα 100 ερωτήματα. Όπως και για τιμή ποσοστού ίση με 0.0105, η συγκεκριμένη τιμή προσεγγίζει την μέθοδο σταθερού παραθύρου συλλογής για μέγεθος ίσο με 5, για τους ίδιους λόγους που αναφέρθηκαν στο προηγούμενο υποκεφάλαιο. Επεκτείνοντας από την παρατήρηση του προηγούμενου υποκεφαλαίου, παρατηρείται ότι για τιμές ποσοστού μεγαλύτερες από 0.02 (2%) τα αποτελέσματα προσεγγίζουν τις καλύτερες περιπτώσεις που αναφέρθηκαν, ακόμα και αν τα μεγέθη των παραθύρων που παράγονται δεν προσεγγίζουν την μέθοδο σταθερού παραθύρου συλλογής. Αντίθετα, κυμαίνονται, κυρίως, σε τιμές μεγέθους μεταξύ του 8 και του 14. Για άλλη μία φορά, γίνεται φανερό ότι

εκτός από το μέγεθος του παραθύρου, τα αποτελέσματα εξαρτιούνται και από τα σημεία που χωρίζεται κάθε κείμενο σε παράθυρα.

Αν και τα αποτελέσματα του σταθερού παραθύρου παραγράφου στην CF, δεν είναι όσο καλά όσο με τις άλλες μεθόδους, είναι ακόμα πολύ καλύτερα από τις μεθόδους των Καλογερόπουλο, Δούκα, Καναβό και Μακρή (Kalogeropoulos et al., 2020). Στο πείραμα με τιμές "0.023 150 a=1 b=0.05", παράχθηκαν τα καλύτερα αποτελέσματα, απαντώντας σε 75 στα 100 ερωτήματα καλύτερα από το Set-based μοντέλο. Η πιο σημαντική παρατήρηση που μπορεί να αντληθεί είναι ότι τα αποτελέσματα βελτιώνονται σημαντικά όσο μειώνεται η μεταβλητή σημαντικότητας β, δηλαδή όσο πέφτει η επιρροή του γραφήματος της παραγράφου στο τελικό ενιαίο γράφημα.

GSB - K-Core με παράθυρα, χωρίς απαλοιφή - Παράθυρο κειμένου - Παράθυρο παραγράφου

Το κίνητρο πίσω από αυτό το σετ πειραμάτων, ήταν ποια θα ήταν τα αποτελέσματα αν, στις μεθόδους διατήρησης σημαντικών κόμβων, τα γραφήματα κειμένων παράγονταν χρησιμοποιώντας παράθυρα. Ο διαχωρισμός των κειμένων σε παράθυρα στις μεθόδους Maincore, Density και Corerank έγινε χρησιμοποιώντας τη βέλτιστη τιμή για την βέλτιστη μέθοδο ως αυτό το σημείο. Αυτό σημαίνει ότι τα γραφήματα κειμένων παράχθηκαν με την μέθοδο σταθερού παραθύρου κειμένου για τιμή ποσοστού ίση με 0.01055 (1.055%). Σε αυτό το σετ πειραμάτων, δεν χρησιμοποιήθηκε απαλοιφή ακμών πριν τον εντοπισμό των σημαντικών κόμβων. Συνολικά εκτελέστηκαν 21 πειράματα και χρησιμοποιήθηκαν οι μέθοδοι GSB, Maincore, Density, Corerank, σταθερού παραθύρου για κάθε κείμενο (Constant Window %) και σταθερού παραθύρου παραγράφου (Sentence - Paragraph Window). Τα αποτελέσματα φαίνονται στο διάγραμμα 6.4. Στο κάτω μέρος του διαγράμματος φαίνονται το μπόνους που δίνεται στους σημαντικούς κόμβους, τα ποσοστά των κειμένων που χρησιμοποιήθηκαν ως παράθυρα και το μέγεθος του παραθύρου παραγράφου με αυτή την σειρά. Οι μεταβλητές σημαντικότητας που χρησιμοποιήθηκαν για την μέθοδο σταθερού παραθύρου παραγράφου α και β, ήταν ίσες με 1 και 0.05 αντίστοιχα.

Σε αυτό το σημείο γίνεται φανερό πόσο πιο πολύ αυξάνεται η απόδοση χρησιμοποιώντας τα πιο συνεκτικά και ισχυρότερα σημασιολογικά γραφήματα κειμένων που παράγονται με την χρήση μεθόδων παραθύρων. Και για τις τρεις μεθόδους διατήρησης κόμβων, τα αποτελέσματα βελτιώνονται σημαντικά, απαντώντας, για οποιαδήποτε τιμές εισόδου, τουλάχιστον σε 75 στα 100 ερωτήματα καλύτερα από το Set-based μοντέλο. Αν και τα αποτελέσματα τους δεν είναι όσο καλά, όσο στην απλή μέθοδο σταθερού παραθύρου κειμένου, είναι αρκετά κοντά. Συμπεραίνουμε, δηλαδή, ότι η χρήση μεθόδων διατήρησης σημαντικών κόμβων με παράθυρα, παραδόξως, μειώνουν την ποιότητα των

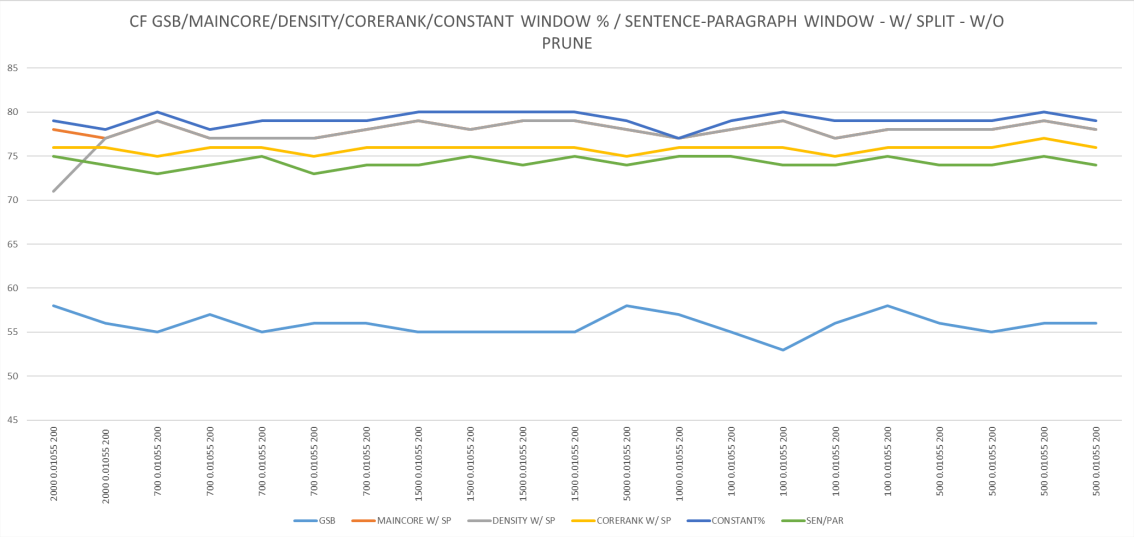

Figure 6.4: GSB - K-Core με παράθυρα, χωρίς απαλοιφή - Παράθυρο κειμένου - Παράθυρο παραγράφου

ευρετηρίων και ως αποτέλεσμα και της διαδικασίας της ανάκτησης.

GSB - K-Core με παράθυρα, με απαλοιφή - Παράθυρο κειμένου - Παράθυρο παραγράφου

Για λόγους διατήρησης της ακεραιότητας της πειραματικής διαδικασίας, στο συγκεκριμένο σετ πειραμάτων εκτελέστηκαν τα πειράματα του προηγούμενου σετ, με την μόνη διαφορά ότι πλέον θα γίνει και απαλοιφή ακμών πριν τον εντοπισμό των σημαντικών κόμβων. Οι παράμετροι της παραγωγής των γραφημάτων κειμένου είναι οι ίδιοι με αυτές του προηγούμενου υποκεφαλαίου. Συνολικά εκτελέστηκαν 21 πειράματα και χρησιμοποιήθηκαν οι μέθοδοι GSB, Maincore, Density, Corerank, σταθερού παραθύρου για κάθε κείμενο (Constant Window %) και σταθερού παραθύρου παραγράφου (Sentence - Paragraph Window). Τα αποτελέσματα φαίνονται στο διάγραμμα 6.4. Στο κάτω μέρος του διαγράμματος φαίνονται το μπόνους που δίνεται στους σημαντικούς κόμβους, τα κατώφλια απαλοιφής ακμών, τα ποσοστά των κειμένων που χρησιμοποιήθηκαν ως παράθυρα και το μέγεθος του παραθύρου παραγράφου με αυτή την σειρά. Οι μεταβλητές σημαντικότητας που χρησιμοποιήθηκαν για την μέθοδο σταθερού παραθύρου παραγράφου α και β, ήταν ίσες με 1 και 0.05 αντίστοιχα.

Οι παρατηρήσεις που μπορούν να αντληθούν από τα αποτελέσματα είναι οι ίδιες με πριν. Τα αποτελέσματα βελτιώνονται σημαντικά, αν και δεν φτάνουν αυτά από την μέθοδο χωρίς διατήρηση σημαντικών κόμβων. Ιδιαίτερο ενδιαφέρον παρουσιάζει το γεγονός ότι με την απαλοιφή ακμών, τα αποτελέσματα σε γενικές γραμμές είναι ολίγον χειρότερα, από τα αποτελέσματα όταν δεν χρησιμοποιείται απαλοιφή. Αυτό μάλλον

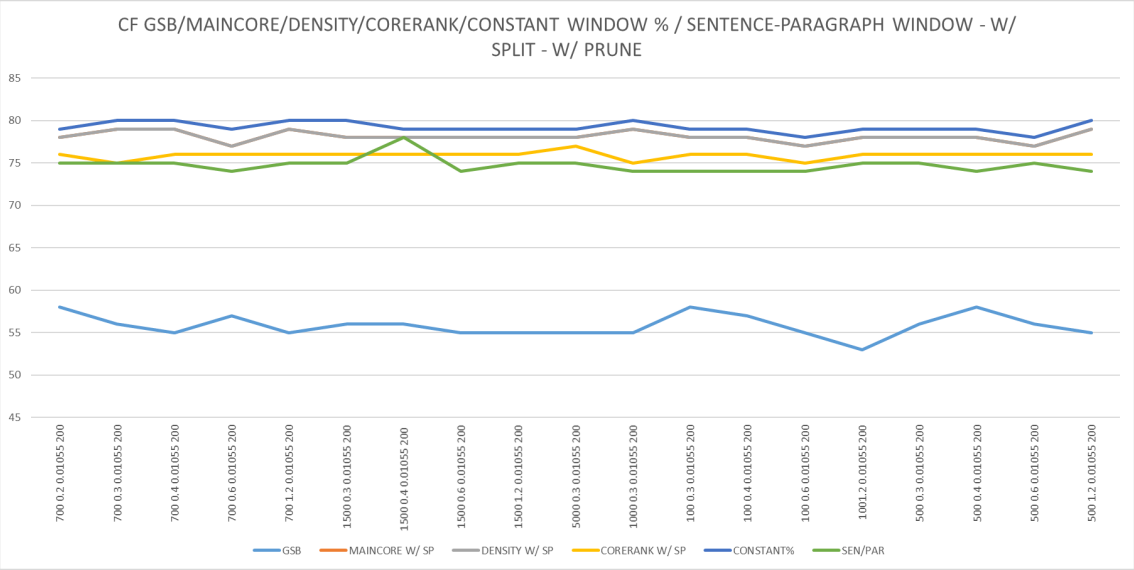

Figure 6.5: GSB - K-Core με παράθυρα, με απαλοιφή - Παράθυρο κειμένου - Παράθυρο παραγράφου

οφείλεται στο γεγονός ότι με την απαλοιφή ακμών χάνεται πληροφορία στα γραφήματα κειμένων. Στην περίπτωση του GSB, επειδή τα γραφήματα κειμένων είναι πλήρη αυτή η πληροφορία είναι αμελητέα, και ίσως μάλιστα, βλαβερή για την ανάκτηση. Όταν χρησιμοποιούνται παράθυρα, ωστόσο, τα γραφήματα είναι τόσο συνεκτικά και σημασιολογικά ισχυρά που αυτή η απώλεια πληροφορίας, οδηγεί σε μείωση της απόδοσης, έστω και μικροσκοπικής.

GSB - K-Core με παράθυρα, χωρίς απαλοιφή - Ολισθούμενο παράθυρο των (Rousseau and Vazirgiannis, 2013a) - Παράθυρο παραγράφου

Στο συγκεκριμένο σετ, εκτελέστηκαν πειράματα για να εκτιμηθεί η απόδοση του ολισθούμενο παράθυρου των (Rousseau and Vazirgiannis, 2013a) σε σχέση με τις προτεινόμενες μεθόδους. Χρησιμοποιήθηκαν πέρα από το GSB, οι εκδόσεις των Maincore, Density και Corerank με παράθυρα και χωρίς απαλοιφή ακμών. Στην συγκεκριμένη περίπτωση, ωστόσο, το παράθυρο λειτουργούσε με τον τρόπο που περιγράφηκε από τους (Rousseau and Vazirgiannis, 2013a). Παράλληλα, χρησιμοποιήθηκε η απλή έκδοση του ολισθούμενου παραθύρου, δηλαδή χωρίς τεχνικές διατήρησης σημαντικών κόμβων, και μία νέα έκδοση του παραθύρου παραγράφου, όπου και τα δύο επιμέρους γραφήματα κατασκευάζονται επίσης με ολισθούμενο παράθυρο. Τέλος, πρέπει να σημειωθεί ότι τα μεγέθη παραθύρων στις μεθόδους Maincore, Density, Corerank και την απλή έκδοση του ολισθούμενου παραθύρου παράχθηκαν ως ποσοστό του μεγέθους του κειμένου σε λέξεις (όπως στο σταθερό παράθυρο κειμένου). Το ίδιο ισχύει και για το επίπεδο προτάσεως της μεθόδου σταθερού παραθύρου παραγράφου. Συνο-

λικά εκτελέστηκαν 19 πειράματα και τα αποτελέσματα φαίνονται στο διάγραμμα 6.6. Στο κάτω μέρος του διαγράμματος, απεικονίζονται, πρώτα, το μπόνους που αποδίδεται στους σημαντικούς κόμβους, έπειτα, τα ποσοστά των κειμένων που χρησιμοποιήθηκαν ως παράθυρα και τέλος, το μέγεθος του παραθύρου παραγράφου, ενώ οι μεταβλητές σημαντικότητας για αυτό ήταν 1 και 0.1 για το α και β αντίστοιχα.

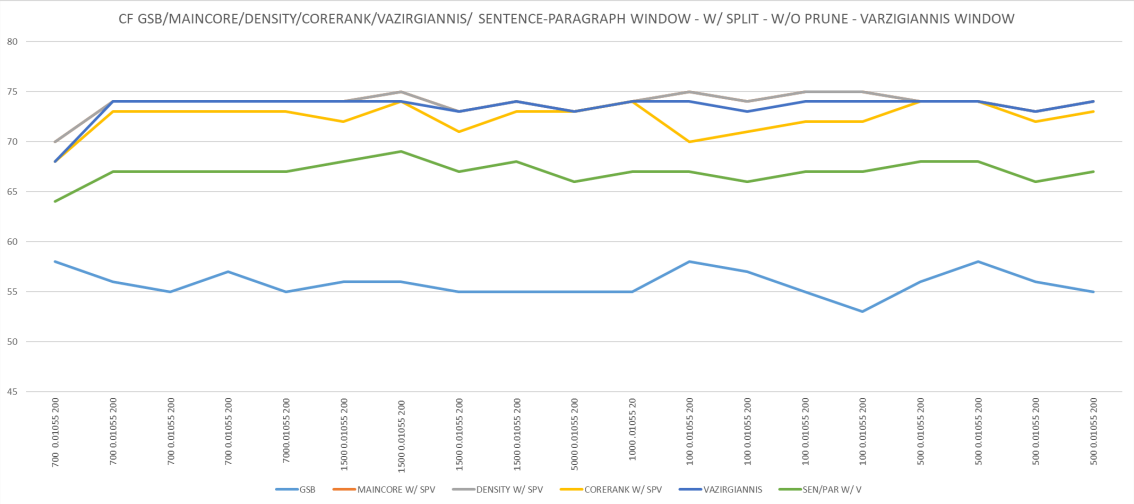

Figure 6.6: GSB - K-Core με παράθυρα, χωρίς απαλοιφή - Ολισθούμενο παράθυρο των Vazirgiannis/Rousseau - Παράθυρο παραγράφου

Κατευθείαν είναι φανερό ότι ενώ η χρήση του ολισθούμενου παράθυρου βελτιώνει τα αποτελέσματα αισθητά σε σχέση με το GSB, αυτά δεν είναι όσο υψηλά όσο με τα παράθυρα των προτεινόμενων μεθόδων. Ενδιαφέρον παρουσιάζει το γεγονός ότι οι μέθοδοι maincore και density παρήγαγαν τα ίδια ακριβώς αποτελέσματα. Τα αποτελέσματα των δύο αυτών μεθόδων ήταν και τα καλύτερα, απαντώντας στο μέγιστο σε 75 στα 100 ερωτήματα καλύτερα από το Set-based μοντέλο. Είναι επίσης ενδιαφέρον το γεγονός ότι το ολισθούμενο παράθυρο χωρίς διατήρηση σημαντικών κόμβων, παρήγαγε χειρότερα αποτελέσματα από τις μεθόδους που χρησιμοποιούν τεχνικές διατήρησης σημαντικών κόμβων, το οποίο είναι το αντίθετο από αυτό που παρατηρήθηκε στα προηγούμενα σετ πειραμάτων με τα παράθυρα των προτεινόμενων μεθόδων. Τέλος, είναι αναγκαίο να σημειωθεί ότι τα αποτελέσματα των μεθόδων με την χρήση ολισθούμεου παραθύρου των (Rousseau and Vazirgiannis, 2013a), ήταν ολοκληρωτικά χειρότερα από τα αποτελέσματα των ίδιων μεθόδων, όταν χρησιμοποιήθηκαν τα παράθυρα των προτεινομένων μεθόδων.

GSB - K-Core - Σταθερό παράθυρο κειμένου - Σταθερό παράθυρο παραγράφου - Σταθερό παράθυρο κειμένου με ποινή στο συνολικό γράφημα

Στο συγκεκριμένο σετ πειραμάτων, εκτιμήθηκε η απόδοση των μεθόδων όταν εφαρμόζεται ποινή στο συνολικό γράφημα. Χρησιμοποιήθηκαν το μοντέλο GSB, οι απλές μορφές των density και corerank, η μέθοδο σταθερού παραθύρου κειμένου, το σταθερό παράθυρο παραγράφου και το σταθερό παράθυρο κειμένου με ποινή στο συνολικό γράφημα. Εκτελέστηκαν συνολικά 18 πειράματα, των οποίων τα αποτελέσματα φαίνονται στο διάγραμμα 6.7. Το υπόμνημα του διαγράμματος είναι οργανωμένο με την παρακάτω σειρά. Η πρώτη τιμή δηλώνει την ποινή που εφαρμόστηκε στα βάρη των ακμών στο συνολικό γράφημα, η δεύτερη τιμή το ποσοστό των μεγεθών των κειμένων που χρησιμοποιήθηκαν ως μέγεθος παραθύρου και η τρίτη το μέγεθος παραθύρου παραγράφου. Οι μεταβλητές σημαντικότητας α και β, ισούται με 1 και 0.05 αντίστοιχα.

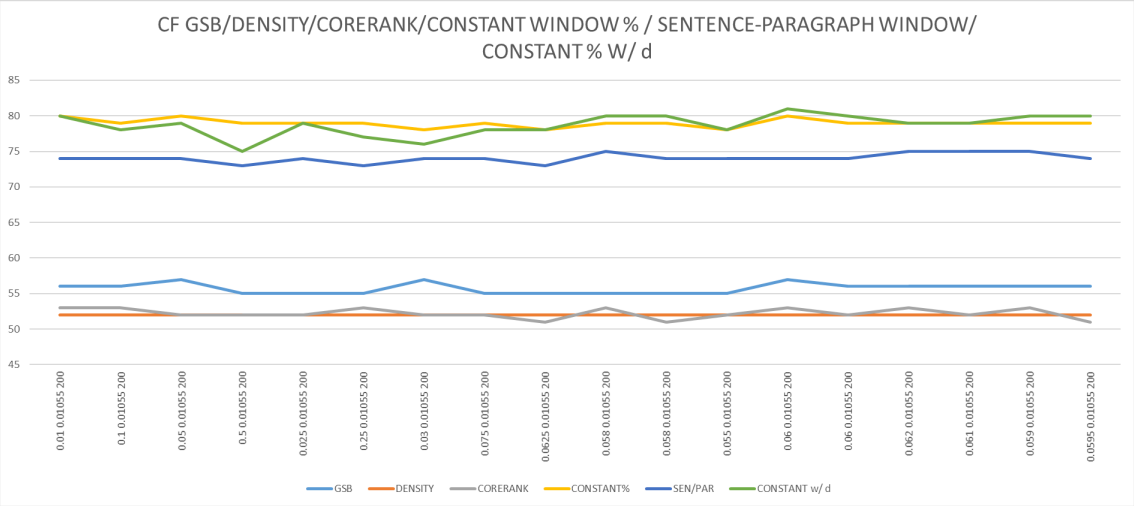

Figure 6.7: GSB - K-Core - Σταθερό παράθυρο κειμένου - Σταθερό παράθυρο παραγράφου - Σταθερό παράθυρο κειμένου με ποινή στο συνολικό γράφημα

Τα αποτελέσματα των μεθόδων που έχουν συζητηθεί ήδη δεν παρουσιάζουν ιδιαίτερο ενδιαφέρον στο συγκεκριμένο σετ πειραμάτων. Το επίκεντρο του συγκεκριμένου σετ είναι τα αποτελέσματα της μεθόδου σταθερού παραθύρου κειμένου, όταν χρησιμοποιείται ποινή στις ακμές του συνολικού γραφήματος της συλλογής. Για την συλλογή CF, η μέθοδο σταθερού παραθύρου κειμένου, ποσοστού 0.01055, με ποινή στο συνολικό γράφημα, ίση με 0.06, έβγαλε το υψηλότερο αριθμό απαντημένων ερωτημάτων καλύτερα από το Set-based, με 81 στα 100 ερωτήματα. Ακόμα, μία πολύ σημαντική παρατήρηση, η οποία δυστυχώς δεν φαίνεται στο διάγραμμα, είναι ότι αν γίνει σύγκριση των αποτελεσμάτων της μεθόδου που χρησιμοποιεί την ποινή, με την απλή έκδοση της αντί του Set-based, διαπιστώνεται ότι τα αποτελέσματα είναι καλύτερα ποιοτικά,

ακόμα και αν πολλές φορές σε σχέση με το Set-based, οι δύο μέθοδοι παράγουν ίδια αποτελέσματα.

Αποτελέσματα χωρίς ασήμαντες λέξεις

Σε αυτό το σημείο θα συζητηθούν τα αποτελέσματα της ανάκτησης στην συλλογή CF όταν αφαιρέθηκαν οι αγγλικές ασήμαντες λέξεις της βιβλιοθήκης NLTK(Steven Bird and Loper, 2009). Συνολικά αφαιρέθηκαν 127 λέξεις. Συγκρίθηκαν οι μέθοδοι σταθερού παραθύρου για κάθε κείμενο και σταθερού παραθύρου παραγράφου. Τα αποτελέσματα φαίνονται στο διάγραμμα 6.8.

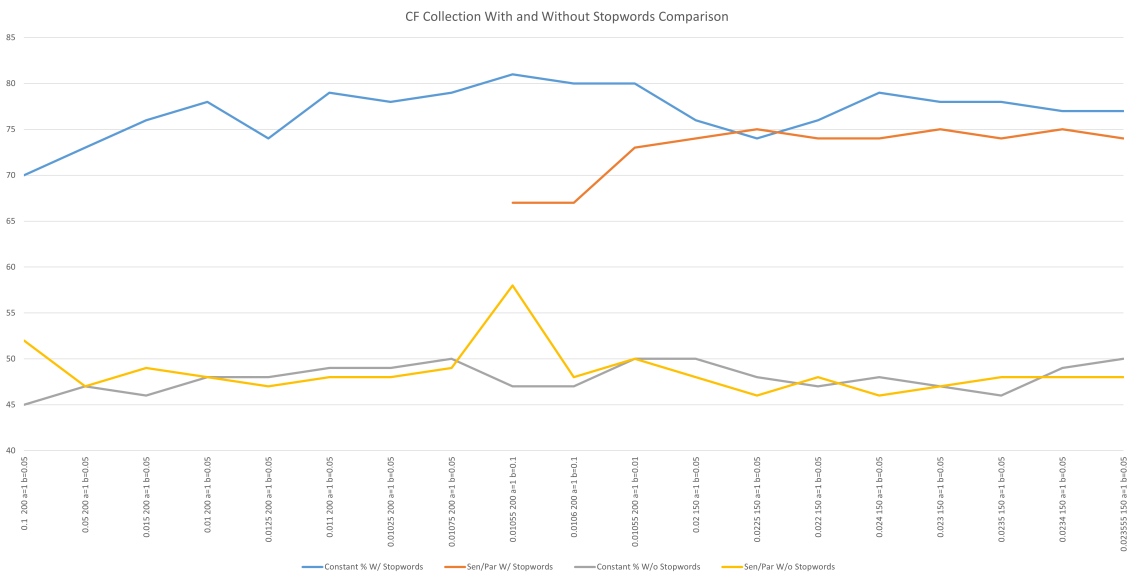

Figure 6.8: Σύγκρισή μεθόδων με παράθυρα με και χωρίς ασήμαντες λέξεις.

Τα αποτελέσματα μιλάνε από μόνα τους. Όταν αφαιρούνται οι ασήμαντες λέξεις η απόδοση των μεθόδων μειώνεται δραματικά. Η μέθοδοι χωρίς τις ασήμαντες λέξεις αποδίδουν χαμηλότερα ακόμα και από τις μεθόδους που δεν χρησιμοποιούν παράθυρα. Τα αποτελέσματα αυτά σε συνδυασμό με τα αποτελέσματα που παρατηρήθηκαν όταν εισάγεται ποινή στο συνολικό γράφημα επιβεβαιώνουν την υπόθεση ότι η συμπερίληψη των ασήμαντων λέξεων με τις προτεινόμενες μεθόδους ενισχύουν την απόδοση της ανάκτησης της πληροφορίας σημαντικά.

6.2.2 Αποτελέσματα στην NPL συλλογή

Η παρουσίαση των αποτελεσμάτων συνεχίζεται με την NPL συλλογή. Στην συγκεκριμένη συλλογή έγιναν πειράματα και συγκρίσεις με το GSB, το Corerank, την μέθοδο σταθερού παραθύρου κειμένου, την μέθοδο σταθερού παραθύρου παραγράφου, καθώς και λίγα με την μέθοδο σταθερού παραθύρου συλλογής. Σε όλες τις μεθόδους

εφαρμόστηκε ποινή ακμών στο συνολικό γράφημα συλλογής με τιμή 0.06. Συνολικά εκτελέστηκαν 25 πειράματα, με τις τιμές των παραθύρων να κυμαίνονται από 3 μέχρι 10 για το σταθερό παράθυρο συλλογής, 150 έως 200 για το σταθερό παράθυρο παραγράφου και με 0.01055 ως το ποσοστό κειμένου για την μέθοδο σταθερού παραθύρου κειμένου. Δεν πραγματοποιήθηκαν περισσότερα πειράματα, λόγου του μεγέθους των κειμένων.Η συλλογή NPL χαρακτηρίζεται από πολλά μικρά κείμενα (<10 λέξεων), οπότε ο ορισμός των μεγεθών των παραθύρων ήταν αρκετά περιορισμένος. Τέλος, υπενθυμίζεται ότι η συλλογή NPL περιλαμβάνει 71 ερωτήματα αντί των 100 τις CF. Ακόμα, στην συγκεκριμένη συλλογή υπήρχε ιδιαίτερα υψηλός αριθμός ερωτημάτων στα οποία το Set-based και οι μέθοδοι απάντησαν το ίδιο με αποτέλεσμα τον αυξημένο αριθμό των μηδενικών στα αποτελέσματα. Για αυτό το λόγο δημιουργήθηκαν δύο διαγράμματα, ένα που περιέχει τα αποτελέσματα χωρίς τα μηδενικά (6.9) και ένα μόνο με τα μηδενικά (6.10).

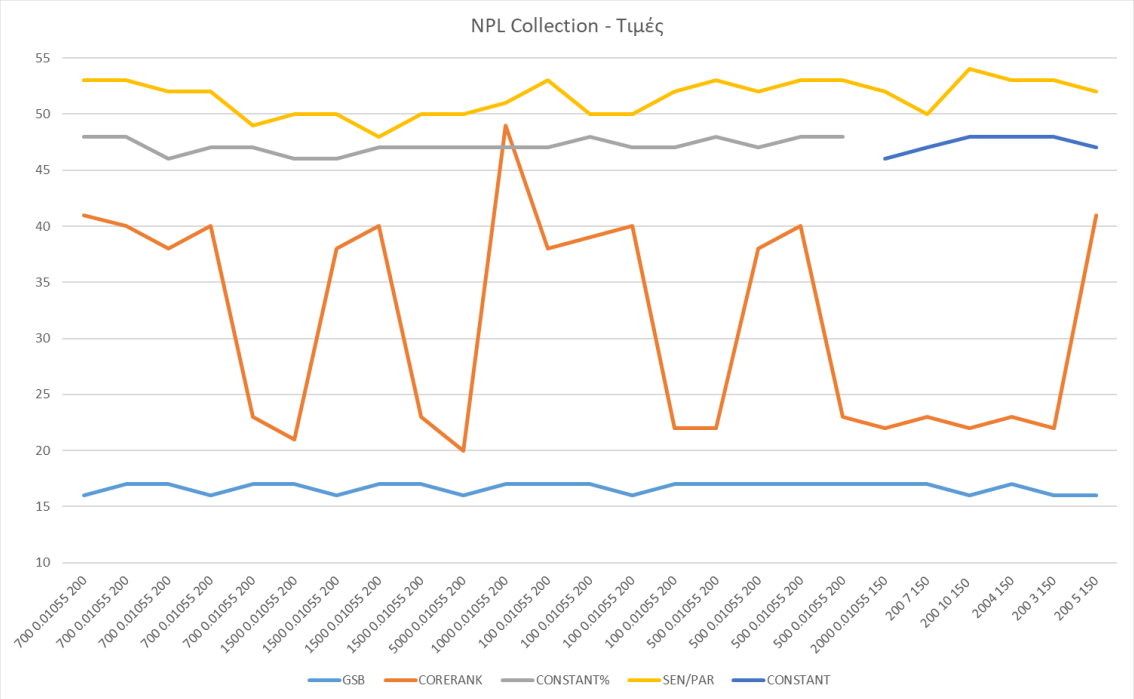

Figure 6.9: Αποτελέσματα στην NPL συλλογή

Παρατηρείται ότι ακόμα και με πολύ μικρά κείμενα οι προτεινόμενοι μέθοδοι παράγουν καλύτερα αποτελέσματα από το GSB και τη Corerank. Από όλες τις μεθόδους καλύτερη απόδοση έχει η μέθοδο σταθερού παράθυρου παραγράφου, με βέλτιστη τιμή απαντήσεων, 54 στα 71 ερωτήματα καλύτερα από το Set-based και 10 στα 71 ίδια με το Set-based. Επίσης, πρέπει να αναφερθεί ότι οι μέθοδοι που χρησιμοποιούν παράθυρα είχαν πολύ λιγότερο ποσοστό μηδενικών από τις άλλες μεθόδους. Ακόμα, είναι αναμενόμενο ότι με τόσο μικρά κείμενα οι μέθοδοι σταθερού παραθύρου κειμένου και

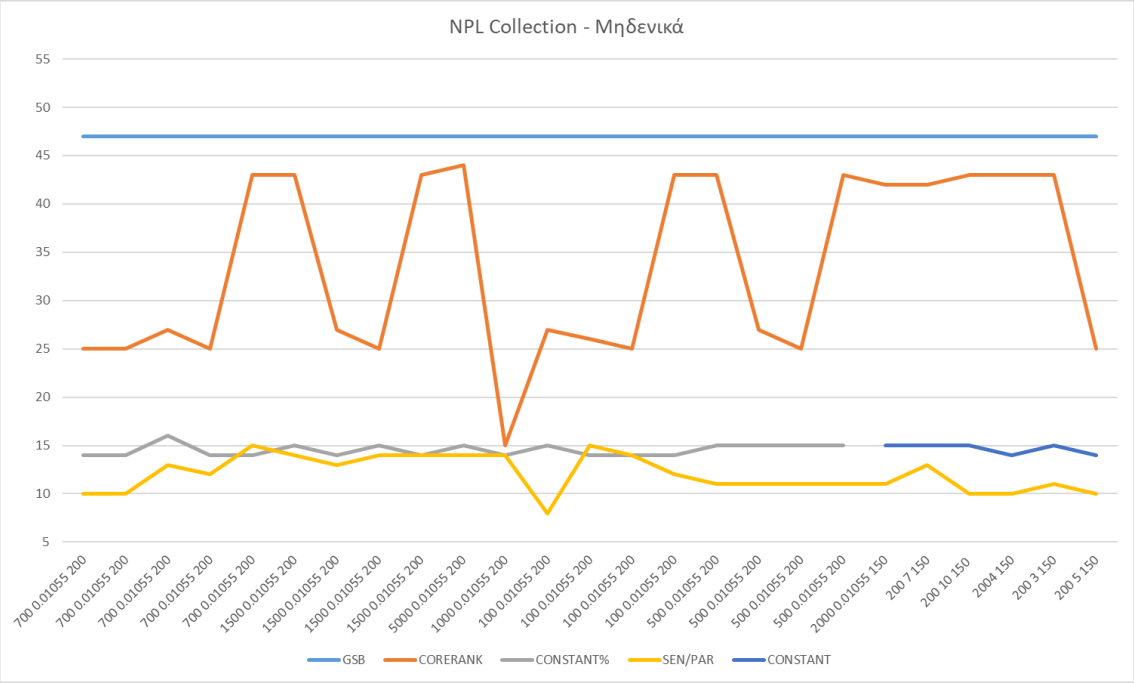

Figure 6.10: Αποτελέσματα στην NPL συλλογή - Μηδενικά

σταθερού παραθύρου συλλογής, συμπίπτουν.

6.2.3 Αποτελέσματα στην Cranfield συλλογή

Η επόμενη συλλογή για την οποία θα παρουσιαστούν τα αποτελέσματα είναι η Cranfield. Για την συγκεκριμένη συλλογή χρησιμοποιήθηκαν οι μέθοδοι GSB, σταθερού παραθύρου κειμένου, παράθυρο μεγέθους πρότασης, σταθερό παράθυρο συλλογής και σταθερό παράθυρο παραγράφου. Όπως και πριν για όλες τις μεθόδους εφαρμόστηκε ποινή στο συνολικό γράφημα με τιμή 0.06. Ο αριθμός των πειραμάτων που εκτελέστηκαν ήταν 54. Τα μεγέθη των παραθύρων κυμαίνονται από 2 μέχρι 20 για το σταθερό παράθυρο συλλογής, 150 και 200 για το σταθερό παράθυρο παραγράφου και 0.001 έως 0.1 ποσοστό μεγέθους κάθε κειμένου για το σταθερό παράθυρο κειμένου. Για την κατασκευή του γραφήματος επιπέδου πρότασης για την μέθοδο σταθερού παραθύρου παραγράφου χρησιμοποιήθηκε η μέθοδο σταθερού παραθύρου κειμένου. Ακόμα, ως μεταβλητές σημαντικότητας χρησιμοποιήθηκαν οι τιμές 1 και 0,05 για τα α και β αντίστοιχα. Τέλος, όπως αναφέρθηκε ήδη στο υποκεφάλαιο της υλοποίησης, χρησιμοποιήθηκε ο tokenizer της βιβλιοθήκης NTLK (punkt). Τα κείμενα της συλλογής Cranfield θυμίζουν σε μέγεθος και αριθμό αυτά της CF. Επίσης, η συλλογή Cranfield περιλαμβάνει συνολικά 225 ερωτήματα αντί για 100 της CF, ή 71 της NPL. Τα αποτελέσματα απεικονίζονται στο διάγραμμα 6.11 και στο υπόμνημα βρίσκονται πρώτα το μέγεθος παραθύρου για την μέθοδο σταθερού παραθύρου συλ-

λογής, έπειτα, το μέγεθος παραθύρου παραγράφου και τέλος, το ποσοστό κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου για τη μέθοδο σταθερού παραθύρου κειμένου.

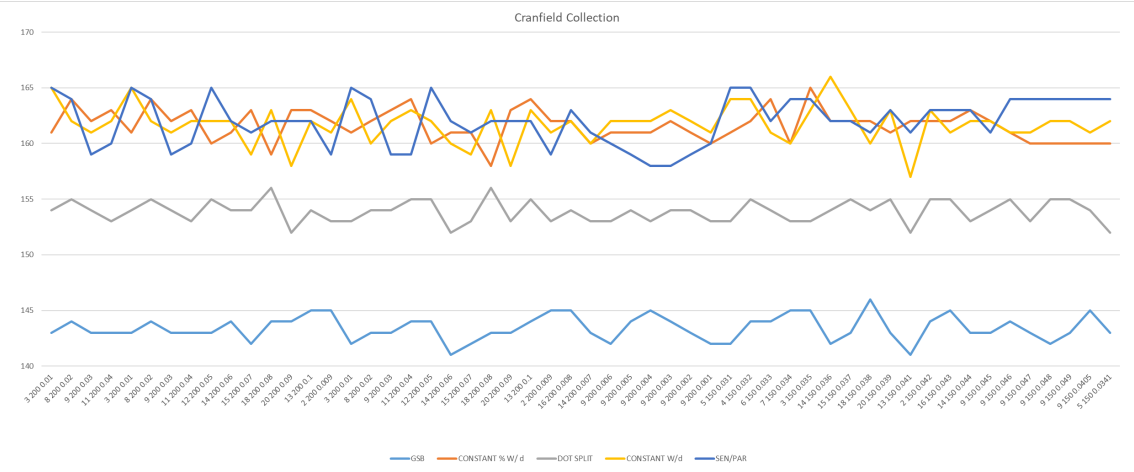

Figure 6.11: Αποτελέσματα στην Cranfield συλλογή

Τα αποτελέσματα που παρατηρούνται είναι παρόμοια με αυτά της CF. Για άλλη μία φορά επιβεβαιώνεται ότι οι προτεινόμενες μέθοδοι αποδίδουν σημαντικά καλύτερα από το GSB. Ακόμα και η μέθοδος μεγέθους πρότασης αν και τα αποτελέσματα της είναι χαμηλότερα από τις άλλες μεθόδους, ακόμα αποδίδει αισθητά καλύτερα από το GSB. Κατά μέσο όρο τα βέλτιστα αποτελέσματα προήλθαν από την μέθοδο σταθερού παραθύρου παραγράφου, αν και η κορυφαία τιμή ήταν από την μέθοδο σταθερού παραθύρου συλλογής για παράθυρο μεγέθους 14, απαντώντας σε 166 στα 225 ερωτήματα καλύτερα από το Set-based μοντέλο. Αν και δεν φαίνεται στο διάγραμμα πρέπει να σημειωθεί ότι οι μέθοδοι απάντησαν σε αρκετά ερωτήματα ίδια με το Set-based μοντέλο. Κατά μέσο όρο αυτά τα ερωτήματα(μηδενικά) ήταν μεταξύ 5 έως 10 ανάλογα το πείραμα που εκτελούνταν. Λαμβάνοντας και αυτά υπόψιν, μπορεί να σημειωθεί μία ενδιαφέρον παρατήρηση. Τα μοντέλα σταθερού παραθύρου συλλογής, σταθερού παραθύρου κειμένου και σταθερού παραθύρου παραγράφου, απαντούν καλύτερα ή τουλάχιστον ίδια με το Set-based, σε περισσότερα από το 72% των ερωτημάτων κάθε συλλογής. Αυτό ισχύει και για την CF($\approx 79\% - 80\%$) και για την NPL($\approx 87\%$) και για την Cranfield($\approx 72\% - 76\%$).

6.2.4 Αποτελέσματα στην Time συλλογή

Αποτελέσματα Ανάκτησης στην συλλογή Time

Τέλος, χρησιμοποιήθηκε η συλλογή Time. Η συλλογή Time αποτελείται από 424 κείμενα, των οποίων το μέγεθος κυμαίνεται από μερικές λέξεις έως και 4000+ λέξεις στα μεγαλύτερα κείμενα. Χρησιμοποιήθηκαν οι μέθοδοι GSB, σταθερό παράθυρο

συλλογής, σταθερό παράθυρο κειμένου, σταθερό παράθυρο παραγράφου και παράθυρο πρότασης. Οι τιμές των παραθύρων ήταν μεταξύ των 2 με 20 για το σταθερό παράθυρο συλλογής, 200 για το παράθυρο παραγράφου και 0.005 έως 0.1 για ποσοστό κειμένου ως παράθυρο για την μέθοδο σταθερού παραθύρου κειμένου. Το γράφημα κειμένου σε επίπεδο πρότασης για την μέθοδο σταθερού παραθύρου παραγράφου, κατασκευάστηκε χρησιμοποιώντας την μέθοδο σταθερού παραθύρου κειμένου. Ακόμα, οι μεταβλητές σημαντικότητας α και β ήταν ίσες με 1 και 0.05 αντίστοιχα. Τέλος, όπως και στη προηγούμενη συλλογή εφαρμόστηκε ποινή στις ακμές του συνολικού γραφήματος σε όλες τις μεθόδους. Η Time συλλογή περιλαμβάνει συνολικά 83 ερωτήματα. Εκτελέστηκαν 119 πειράματα και τα αποτελέσματα τους φαίνονται στο διάγραμμα 6.12. Το υπόμνημα του διαγράμματος είναι οργανωμένο με την παρακάτω σειρά. Η πρώτη τιμή δηλώνει το μέγεθος του παραθύρου για το σταθερό παράθυρο συλλογής, η δεύτερη το μέγεθος παραθύρου παραγράφου για την μέθοδο σταθερού παραθύρου παραγράφου και η τρίτη το ποσοστό των κειμένων που χρησιμοποιήθηκε ως μέγεθος παραθύρου για την μέθοδο σταθερού παραθύρου κειμένου. Τέλος, όταν αλλάζει η τιμή της ποινής που εφαρμόζεται στο συνολικό γράφημα, σημειώνεται πάνω στο όνομα του πειράματος με την λέξη pen.

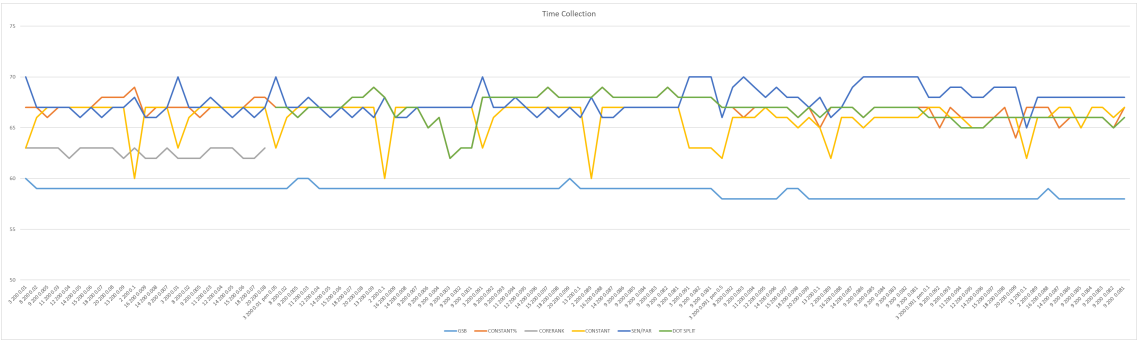

Figure 6.12: Αποτελέσματα στην Time συλλογή

Τα αποτελέσματα επιβεβαιώνουν για μία άλλη φορά όσα έχουν ειπωθεί ως τώρα. Γενικά, στην μέση περίπτωση καλύτερα αποτελέσματα είχε η μέθοδο σταθερού παραθύρου παραγράφου. Επίσης, η ίδια μέθοδο παρήγαγε και το βέλτιστο αποτέλεσμα για αυτή την συλλογή, απαντώντας καλύτερα σε 70 από τα 83 ερωτήματα, σε σχέση με το Set-based μοντέλο. Όπως έχει παρατηρηθεί και στις άλλες συλλογές, είναι ξανά φανερό ότι εκτός από το μέγεθος του παραθύρου, σημαντικό ρόλο παίζει και το σημείο που χωρίζεται το κείμενο σε παράθυρα. Είναι αναγκαίο να σημειωθεί ότι για μερικά πολύ μεγάλα κείμενα, το παράθυρο που παραγόταν από το ποσοστό του μεγέθους του κειμένου ήταν υπερβολικά μεγάλο (400 λέξεις) και πολλές φορές ήταν μεγαλύτερο από το παράθυρο παραγράφου. Ως αποτέλεσμα από αυτό, ίσως να είναι αναγκαίο να ορισθεί και άνω όριο για τα παραγόμενα παράθυρα με την μέθοδο αυτή. Τέλος, η συλλογή

Time ακολουθεί το μοτίβο που περιγράφηκε στην προηγούμενη συλλογή. Οι μέθοδοι με παράθυρα απαντάνε καλύτερα σε ποσοστό τουλάχιστον 80% των ερωτημάτων.

Εντοπισμός Ασήμαντων λέξεων στην συλλογή Time

Η συλλογή Time παρέχει εκτός από τα κείμενα και τις λίστες με τα ερωτήματα και τα σχετικά κείμενα, μία λίστα με τις ασήμαντες λέξεις της συλλογής. Για αυτό το λόγο τα πειράματα που αφορούν τον εντοπισμό ασήμαντων λέξεων εκτελέστηκαν στην συγκεκριμένη συλλογή. Λόγου της φύσεως των μεθόδων που βασίζονται στον εντοπισμό ασήμαντων λέξεων στον κύριο πυρήνα ή στο ευρετήριο, δεν ήταν αναγκαία η εκτέλεση μεγάλου αριθμού πειραμάτων για αυτές τις μεθόδους.

Αρχικά, για τον εντοπισμό των ασήμαντων λέξεων στον κύριο πυρήνα, ακολουθήθηκε η διαδικασία που περιγράφεται στο αντίστοιχο κεφάλαιο. Συνολικά εντοπίστηκαν το 30 % - 40% των ασήμαντων λέξεων. Αυτά τα αποτελέσματα μας οδήγησαν στο συμπέρασμα ότι ο κύριος πυρήνας κάθε κειμένου αποτελείτε από μερικές ασήμαντες λέξεις, μερικές λέξεις κλειδιά, αλλά και μερικές τυχαίες λέξεις που δεν ανήκουν σε καμία κατηγόρια. Το ποσοστό που αναφέρεται παραπάνω είναι ο μέσος όρος των ποσοστών των ασήμαντων λέξεων που βρέθηκαν στο κύριο πυρήνα κάθε κειμένου. Είναι απαραίτητο να αναφερθεί ότι τα αποτελέσματα είχαν ιδιαίτερη μεγάλη διασπορά. Μερικοί κύριοι πυρήνες περιείχαν 10% ασήμαντες λέξεις, ενώ άλλοι περιείχαν 95% ασήμαντες λέξεις. Όπως είναι αναμενόμενο τα μεγαλύτερα ποσοστά προήλθαν από σχετικά μικρά κείμενα, ωστόσο τα μικρά ποσοστά δεν προήλθαν από τα μεγαλύτερα κείμενα της συλλογής αλλά από μεσαία σε μέγεθος κείμενα. Τα μεγαλύτερα κείμενα της συλλογής συνήθως περιείχαν 30-35% ασήμαντες λέξεις στο κύριο πυρήνα. Η επέκταση ωστόσο στον έλεγχο για ασήμαντες λέξεις στο κύριο πυρήνα του συνολικού γραφήματος έδειξε ότι το 67% του κύριου πυρήνα αποτελούνταν από ασήμαντες λέξεις. Μία αισθητή βελτίωση σε σχέση με την αρχική μέθοδο. Ιδέες για προέκταση αυτής της μεθόδου θα συζητηθούν στο κεφάλαιο που ακολουθεί.

Στην συνέχεια, συζητούνται τα αποτελέσματα που προήλθαν από τον εντοπισμό ασήμαντων λέξεων στο ευρετήριο. Με την συγκεκριμένη μέθοδο εντοπίστηκαν συνολικά το 52% των ασήμαντων λέξεων τις συλλογής. Μία αισθητή βελτίωση σε σχέση με την προηγούμενη μέθοδο. Ένα σημαντικό χαρακτηριστικό είναι ότι τα αποτελέσματα, λόγου του τρόπου που υπολογίζονται τα βάρη από το συνολικό γράφημα, είναι σχετικά αδιάφορα με τις μεταβλητές που χρησιμοποιούνται για την κατασκευή του ευρετηρίου, μέχρι κάποιο όριο φυσικά. Στο πίνακα φαίνεται 6.2 η ακρίβεια και η ανάκληση που υπολογίστηκε όταν εξετάζονται μικρότερα ή μεγαλύτερα κομμάτια του ευρετηρίου.

Τέλος, αναλύονται τα αποτελέσματα από την μέθοδο εντοπισμού ασήμαντων λέξεων χρησιμοποιώντας ένα δείγμα κειμένων της συλλογής, τα οποία ήταν και τα πιο

Αριθμός Λέξεων	Ακρίβεια	Ανάκληση
10	1.0	0,02
25	1.0	0.07
50	0,9	0,13
100	0,73	0,21
200	0,6	0,35
340	0,52	0,52
450	0,47	0,62
1000	0,286	0,841

Table 6.2: Results for stopword decection using the invereted file

ενθαρρυντικά. Πριν παρουσιαστούν τα αποτελέσματα, ωστόσο, είναι αναγκαίο να γίνει αναφορά στα πειράματα που εκτελέστηκαν. Η εν λόγω μέθοδος είναι η πιο παραμετροποιήσιμη, με παράμετρους όπως το μέγεθος του δείγματος, η μέθοδο με την οποία κατασκευάζονται τα γραφήματα κειμένων, το μέγεθος παράθυρου που χρησιμοποιείται και οι αναλογία σημαντικότητας ανάμεσα στα δύο σκέλη του βαθμού αξιολόγησης.

Η πιο σημαντική παράμετρος από τις παραπάνω είναι το μέγεθος του δείγματος. Σκοπός είναι να μεγιστοποιηθεί ο αριθμός των ασήμαντων λέξεων που εντοπίζονται στα κορυφαία αποτελέσματα της κατάταξης, ενώ ταυτόχρονα να ελαχιστοποιηθεί ο αριθμός των κειμένων που αποτελούν το δείγμα. Για αυτό τον λόγο πραγματοποιήθηκαν πειράματα με διάφορα μεγέθη δειγμάτων. Όπως έχει αναφερθεί ήδη στην υλοποίηση της μεθόδου, χρησιμοποιήθηκε bucket hashing. Τα μεγέθη δείγματος που εξετάστηκαν ήταν τα $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{6}$, $\frac{1}{8}$ του αριθμού των κειμένων που περιέχει η συλλογή, και εξετάστηκαν όλα τα δείγματα-κουβάδες που παράχθηκαν. Ο διαχωρισμός στους κουβάδες έγινε σύμφωνα με τον αριθμό στο όνομα κάθε κειμένου.

Ως μέθοδο χρησιμοποιήθηκε η **μέθοδο σταθερού παραθύρου συλλογής**, κυρίως για την απλότητα της στην κανονικοποίηση, αφού χρησιμοποιείται πάντα το ίδιο παράθυρο, και την απόδοση της. Το μέγεθος παραθύρου που χρησιμοποιήθηκε ήταν 9 (ή 1) και για τα περισσότερα πειράματα η αναλογία σημαντικότητας ήταν 1:1. Χρησιμοποιήθηκε και η αναλογία 1:5, με το 1 να αντιστοιχεί στο βάρος συμμετοχής και το 5 στο βάρος γραφήματος.

Η αξιολόγηση των αποτελεσμάτων έγινε υπολογίζοντας τις μετρικές ακρίβεια - ανάκληση στα 10, 25, 50, 100, 200 , 340, 450, 1000. Ο αριθμός των ασήμαντων λέξεων της συλλογής ήταν συνολικά 340. Τα πειράματα που εκτελέστηκαν ήταν πάνω στα δείγματα-κουβάδες που παράχθηκαν από 6 κατηγορίες πειραμάτων. Αυτές φαίνονται στο πίνακα 6.3.

Πρέπει να αναφερθεί ότι εξαιτίας των μεθόδων της ανάκτησης, τα κείμενα περιείχαν και ως όρο την τελεία, την οποία μάλιστα η μέθοδο εντοπισμού ασήμαντων λέξεων την

Αριθμός	Μέγεθος Δείγματος (% Συλλογής)	Αριθμός Δειγμάτων	Μέγεθος Παραθύρου	Αναλογία Σκελών Βάρους
1	33.333%	3	9	1:1
2	25%	4	9	1:1
3	12,5%	8	9	1:1
4	16,666%	6	9	1:1
5	16,666%	6	1	1:1
6	16,666%	6	9	1:5

Table 6.3: Types of experiments for stopword detection using samples.

εντόπιζε ως ασήμαντη λέξη, και μάλιστα ήταν από τις πρώτες στην κατάταξη. Επειδή, δεν περιέχεται η τελεία στην λίστα των ασήμαντων λέξεων, τα αποτελέσματα πιθανόν έχουν μία πολύ μικρή απόκλιση. Αυτή η απόκλιση ωστόσο, παίζει αρκετά σημαντικό ρόλο στα αποτελέσματα όταν ελέγχονται σχετικά μικρός αριθμός λέξεων (≤ 100). Είναι πολύ σημαντικό να αναφερθεί ότι με την αφαίρεση των τελείων τα αποτελέσματα μπορούν μόνο να βελτιωθούν, αφού η επόμενη λέξη προς εξέταση, είτε θα είναι ασήμαντη, άρα θα υπάρχει βελτίωση, είτε κάποια άλλη λέξη, οπότε τα αποτελέσματα μένουν ίδια γιατί η τελεία δεν περιέχεται στην λίστα των ασήμαντων λέξεων της συλλογής.

Στην συνέχεια, ακολουθούν διαγράμματα ακρίβειας για τα καλύτερα (6.13) και χειρότερα (6.14) δείγματα των 6 πειραματικών ομάδων που αναφέρθηκαν. Ο αριθμός των λέξεων που εξετάστηκαν ήταν ίσος με τον αριθμός των ασήμαντων λέξεων της συλλογής, δηλαδή 340. Αν ληφθεί αυτό υπόψιν και σύμφωνα με τους τύπους υπολογισμού ακρίβειας και ανάκλησης (1.4 και 1.5), οι τιμές που υπολογίζονται για τις δύο μετρικές είναι ίδιες, με αποτέλεσμα τα διαγράμματα να είναι και αυτά ίδια.

Τα αποτελέσματα είναι πολύ καλύτερα σε σχέση με τις προηγούμενες μεθόδους. Το καλύτερο δείγμα παράχθηκε στην πειραματική ομάδα 5, εντοπίζοντας το 64,7% των ασήμαντων λέξεων όλης της συλλογής. Αντίστοιχα το χειρότερο δείγμα παράχθηκε στην πειραματική ομάδα 3, εντοπίζοντας μόλις το 54,4% των ασήμαντων λέξεων της συλλογής.

Ιδιαίτερο ενδιαφέρον φέρνουν οι ακρίβειες που υπολογίστηκαν εξετάζοντας άλλο αριθμό λέξεων από τις 340. Για παράδειγμα στο πίνακα 6.4 φαίνονται τα αποτελέσματα για τα δείγματα της πειραματική ομάδας 6. Επίσης, τα ίδια φαίνονται και στο διάγραμμα 6.15.

Όταν εξετάζονται έως και 100 λέξεις, τα αποτελέσματα είναι αισθητά καλύτερα, ακόμα και αν εμπεριέχεται η τελεία ως όρος των κειμένων. Παρόμοια αποτελέσματα έβγαλαν και οι άλλες πειραματικές ομάδες.

Ανακεφαλαιώνοντας,η μέθοδο, εντοπίζει το 54% - 64% των ασήμαντων λέξεων της συλλογής, με μέσο όρο γύρω στο 59% - 60%. Όταν δεν εξετάζεται ο μέγιστος αριθμός

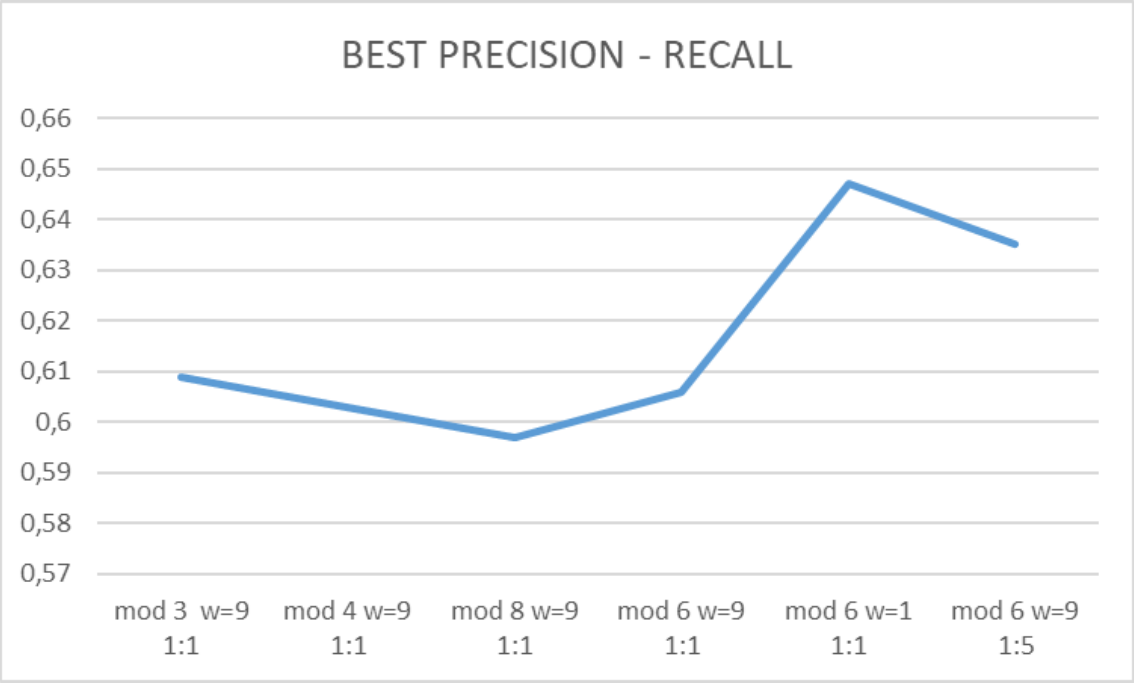

Figure 6.13: Best precision - recall with sampling

Αριθμός Λέξεων	Δείγμα 1	Δείγμα 2	Δείγμα 3	Δείγμα 4	Δείγμα 5	Δείγμα 6
10	0,9	0,9	0,9	0,9	0,9	0,9
25	0,96	0,96	0,96	0,96	0,96	0,96
50	0,96	0,94	0,94	0,96	0,92	0,96
100	0,9	0,89	0,87	0,87	0,87	0,89
200	0,72	0,73	0,71	0,725	0,735	0,735
340	0,61	0,63	0,57	0,59	0,60	0,57
450	0,52	0,54	0,50	0,508	0,50	0,50
1000	0,287	0,287	0,289	0,282	0,287	0,28

Table 6.4: Precision for experiment group 6

των ασήμαντων λέξεων αλλά λιγότερες, τα αποτελέσματα βελτιώνονται σημαντικά.

Τέλος, πρέπει να αναφερθεί ότι στο παράρτημα θα συμπεριληφθούν πίνακες με όλες τις μετρήσεις για όλα τα πειράματα που έχουν γίνει και για την ανάκτηση αλλά και για τον εντοπισμό ασήμαντων λέξεων.

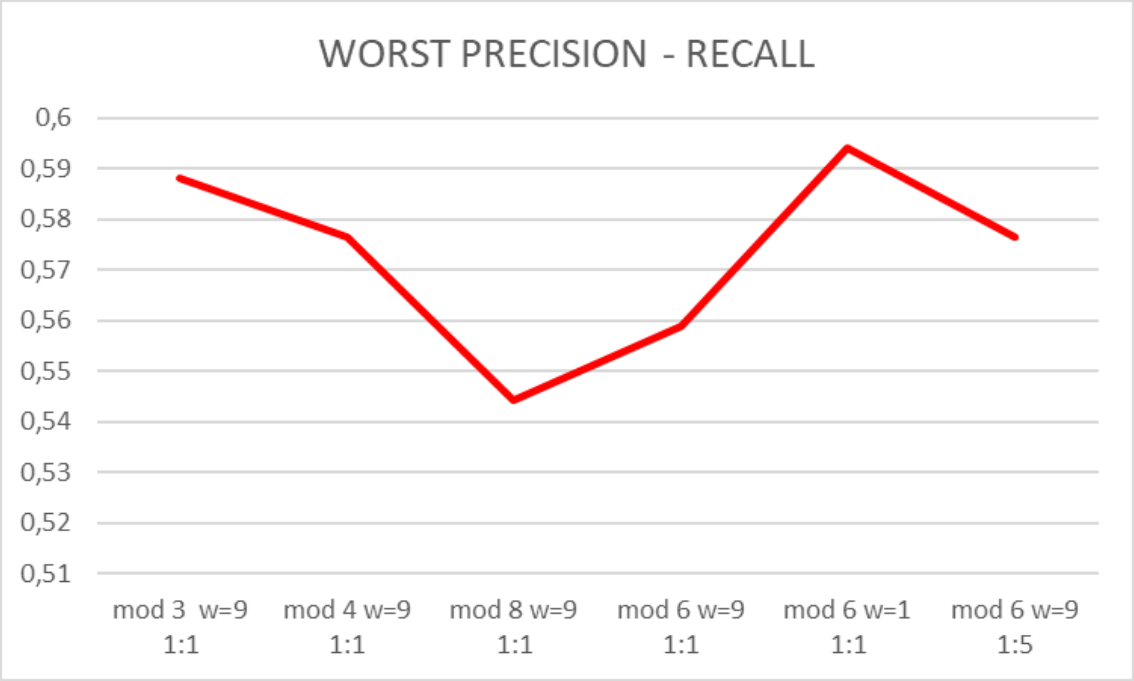

Figure 6.14: Worst precision - recall with sampling

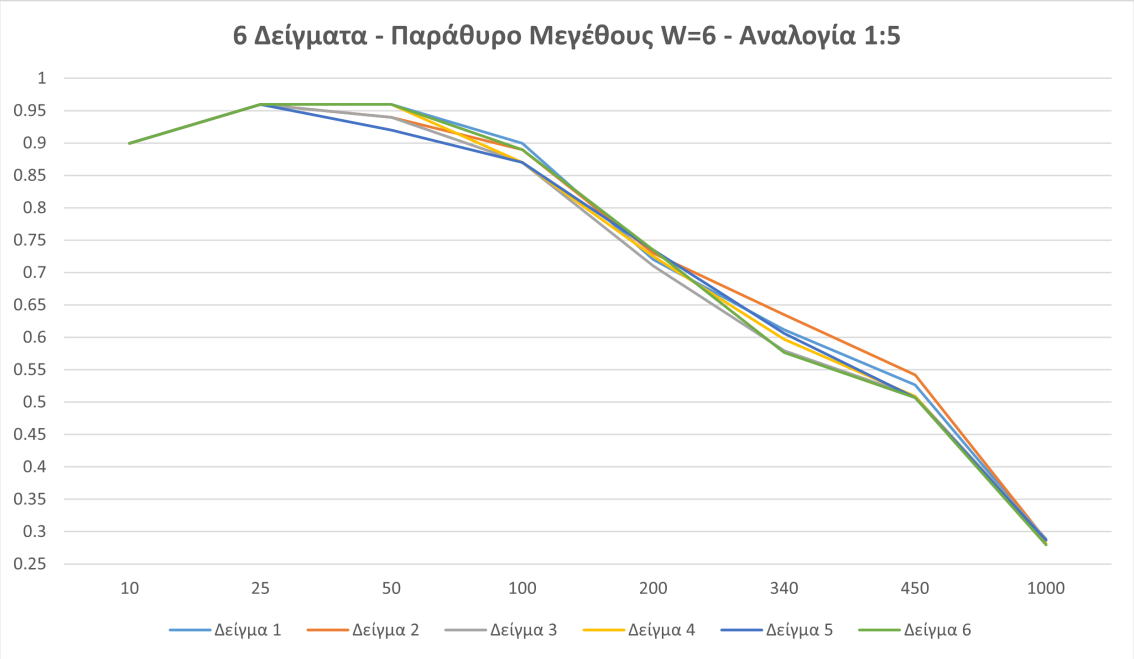

Figure 6.15: Precision for experiment group 6

Chapter 7

Συμπεράσματα και Θέματα Μελλοντικής Μελέτης

7.1 Συμπεράσματα

Σε αυτό το σημείο, γίνεται μία σύντομη σύνοψη των αποτελεσμάτων του προηγούμενου κεφαλαίου και σημειώνονται μερικά συμπεράσματα που προκύπτουν από αυτά. Στην συνέχεια, γίνεται αναφορά σε μερικές ιδέες προς βελτίωση των προτεινόμενων μεθόδων και σε διάφορες παρατηρήσεις. Η παραπάνω ανάλυση θα γίνει πρώτα στα αποτελέσματα της ανάκτησης και έπειτα στα αποτελέσματα από τον εντοπισμό των ασήμαντων λέξεων.

7.1.1 Ανάκτηση Πληροφορίας

Όπως αναφέρθηκε στην εισαγωγή, αρχικά θα γίνει μία σύνοψη των αποτελεσμάτων της ανάκτησης. Οι προτεινόμενες μέθοδοι απέδωσαν αισθητά καλύτερα σε όλες τις συλλογές σε σχέση, τόσο με το Set-based μοντέλο, όσο και με το GSB, και τις μεθόδους ανάλυσης των κειμένων σε πυρήνες. Εστιάζοντας περισσότερο στις προτεινόμενες μεθόδους, δεν υπήρχε μεγάλη διακύμανση στα αποτελέσματα στην συλλογή CF. Η μέθοδο σταθερού παραθύρου για κάθε κείμενο με ποινή στο συνολικό γράφημα, ωστόσο, απέδωσε λίγο καλύτερα από τις άλλες, απαντώντας στην βέλτιστη περίπτωσή σε 81 από τα 100 ερωτήματα καλύτερα από το Set-based μοντέλο, ακολουθούμενη από την μέθοδο σταθερού παραθύρου για κάθε κείμενο χωρίς ποινή στο συνολικό γράφημα απαντώντας καλύτερα σε 80 από τα 100 ερωτήματα. Η σύνοψη των αποτελεσμάτων φαίνονται στον πίνακα 7.1. Οι ονομασίες των μεθόδων είναι αυτές που εμφανίζονται στα διαγράμματα για να είναι πιο εύκολη η ανάγνωση του πίνακα.

Προχωρώντας στην βιβλιοθήκη NPL, τα καλύτερα αποτελέσματα παράχθηκαν από την μέθοδο σταθερού παραθύρου παραγράφου με ποινή στο συνολικό γράφημα, απαντώντας στην βέλτιστη περίπτωση σε 54 από τα 71 ερωτήματα καλύτερα. Η σύνοψη των αποτελεσμάτων στην NPL φαίνονται στο πίνακα 7.2. Οι ονομασίες των μεθόδων είναι ίδιες με αυτές των διαγραμμάτων με σκοπό την πιο εύκολη ανάγνωση του πίνακα.

Μέθοδο	Ερωτήματα	Ποσοστό Ερωτημάτων
GSB	58	58%
MainCore	52	52%
Density	52	52%
Corerank	54	54%
Constant	79	79%
Constant %	80	80%
Sen/Par	78	78%
MainCore w/SP	79	79%
Density w/SP	79	79%
Corerank w/SP	77	77%
Vazirgiannis	74	74%
Constant % w/d	81	81%

Table 7.1: Σύνοψη των αποτελεσμάτων στην CF συλλογή.

Μέθοδο	Ερωτήματα	Ποσοστό Ερωτημάτων
GSB	17	23%
Corerank	41	57%
Constant	48	67%
Constant %	48	67%
Sen/Par	54	76%

Table 7.2: Σύνοψη των αποτελεσμάτων στην NPL συλλογή.

Επόμενη συλλογή είναι η Cranfield. Στην συγκεκριμένη συλλογή απέδωσε καλύτερα η μέθοδο σταθερού παραθύρου συλλογής με ποινή στο συνολικό γράφημα. Απάντησε καλύτερα στην βέλτιστη περίπτωση σε 166 από τα 225 ερωτήματα. Δεύτερη ήταν η μέθοδο σταθερού παραθύρου παραγράφου με ποινή στο συνολικό γράφημα που απάντησε στην βέλτιστη περίπτωση σε 165 από τα 225 ερωτήματα. Η σύνοψη των αποτελεσμάτων στην Cranfield φαίνονται στο πίνακα 7.3. Οι ονομασίες των μεθόδων είναι ίδιες με αυτές των διαγραμμάτων με σκοπό την πιο εύκολη ανάγνωση του πίνακα.

Τέλος, μένει η συλλογή Time. Στην συγκεκριμένη συλλογή καλύτερα απέδωσε η μέθοδο σταθερού παραθύρου παραγράφου με ποινή στο συνολικό γράφημα, απαντώντας στην βέλτιστη περίπτωση σε 70 από τα 83 ερωτήματα καλύτερα από το Set-based

Μέθοδο	Ερωτήματα	Ποσοστό Ερωτημάτων
GSB	145	64%
Dot Split	156	69%
Constant	166	74%
Constant %	165	73%
Sen/Par	165	73%

Table 7.3: Σύνοψη των αποτελεσμάτων στην Cranfield συλλογή.

Μέθοδο	Ερωτήματα	Ποσοστό Ερωτημάτων
GSB	60	72%
Corerank	63	75%
Dot Split	69	83%
Constant	67	80%
Constant %	69	83%
Sen/Par	70	84%

Table 7.4: Σύνοψη των αποτελεσμάτων στην συλλογή Time.

μοντέλο. Η σύνοψη των αποτελεσμάτων στην Time φαίνονται στο πίνακα 7.4. Οι ονομασίες των μεθόδων είναι ίδιες με αυτές των διαγραμμάτων με σκοπό την πιο εύκολη ανάγνωση του πίνακα.

Από τα παραπάνω είναι φανερό ότι οι προτεινόμενες μέθοδοι είναι αποδοτικότερες σε σχέση με τις υπόλοιπες μεθόδους που αναφέρονται στην παρούσα διπλωματική, τόσο σε συλλογές με μεγάλα (Time) ή/και μεσαία (CF, Cranfield, Time) κείμενα, όσο και σε συλλογές με πολύ μικρά κείμενα (NPL). Για την ακρίβεια πρέπει να αναφερθεί ότι οι μέθοδοι αποδίδουν καλύτερα, από το Set-based, σε ένα ποσοστό τουλάχιστον του 70% των ερωτημάτων άσχετα από τη συλλογή που χρησιμοποιείται, εκτός από την NPL, όπου αυτό το ποσοστό επιτυγχάνεται μόνο αν συμπεριληφθούν και τα ερωτήματα που οι μέθοδοι αποδίδουν το ίδιο με το Set-based.

Τα αποτελέσματα αποδεικνύουν επίσης τον ισχυρισμό ότι οι λέξεις μέσα σε ένα κείμενο δεν σχετίζονται ισάξια μεταξύ τους. Δηλαδή, ο βαθμός συσχέτισης μίας λέξης στην αρχή ενός κειμένου με μία λέξη στο τέλος του, είναι πολύ χαμηλότερος από τον βαθμό συσχέτισης δύο λέξεων στην ίδια πρόταση ή σε ένα μικρό κομμάτι του κειμένου (παράθυρο). Η μεθοδολογία των προτεινόμενων αλγορίθμων δείχνουν επίσης ότι αυτές οι σχέσεις μπορούν να αξιοποιηθούν για την βελτίωση της διαδικασίας της ανάκτησης.

Οι παραπάνω σχέσεις είναι τόσο σημαντικές για την ανάκτηση που και μόνο με διαχωρισμό των κειμένων σε παράθυρα σταθερού μεγέθους, τα αποτελέσματα βελτιώνονται σημαντικά άσχετα της μεθόδου που υλοποιείται. Αυτό φαίνεται όταν γίνεται σύγκριση των αποτελεσμάτων των μεθόδων ανάλυσης των κειμένων σε πυρήνες με και χωρίς παράθυρα.

Τέλος, από τα πειράματα που έγιναν με ή χωρίς ασήμαντες λέξεις συμπεραίνετε ότι οι ασήμαντες λέξεις βοηθάνε σημαντικά στην ανάκτηση της πληροφορίας. Αυτό μπορεί να οφείλεται στο γεγονός ότι με την βοήθεια των ασήμαντων λέξεων, τα γραφήματα κειμένων αποτυπώνουν καλύτερα την συνοχή του κειμένου, και γενικά τα γραφήματα κειμένων αποτελούνται από λιγότερα υπο-γραφήματα, τα οποία είναι ξένα μεταξύ τους.

Ιδέες για επέκταση των μοντέλων

Αρχικά, μία πρώτη ιδέα είναι να γίνει προσπάθεια αποτύπωσης της ροής των κειμένων στα γραφήματα, όπως γίνεται ήδη με την δομή και τα παράθυρα. Αυτό μπορεί να επιτευχθεί με την χρήση κατευθυνόμενων γραφημάτων αντί μη κατευθυνόμενων. Επίσης, όπως έχει αναφερθεί ήδη τα γραφήματα των κειμένων που παράγονται αποτελούνται πολλές φορές από υπο-γραφήματα τα οποία είναι μεταξύ τους ξένα με αποτελέσματα η αποτύπωση της δομής του κειμένου να είναι ελλιπής. Αυτό γίνεται επειδή τα παράθυρα ενός κειμένου είναι μεταξύ τους εντελώς ανεξάρτητα. Λαμβάνοντας αυτό υπόψιν μία ακόμα επέκταση που θα μπορούσε να γίνει πάνω στις μεθόδους είναι η συμπερίληψη μίας ακμής, που θα ονομάζεται γέφυρα, μεταξύ δύο κόμβων, ενός γραφήματος κειμένου, που αντιστοιχούν στην τελευταία λέξη ενός παράθυρου και στην πρώτη λέξη του επόμενου παράθυρου. Επίσης, όπως χρησιμοποιείται στην μέθοδο σταθερού παραθύρου παραγράφου, ένα γράφημα που αντιστοιχεί στο επίπεδο πρότασης και ένα γράφημα που αντιστοιχεί στο επίπεδο παραγράφου, θα μπορούσε να δημιουργηθεί μία μέθοδο που να λαμβάνει μεγαλύτερα κομμάτια του κειμένου υπόψιν, όπως για παράδειγμα κεφάλαια ή ίσως και όλο το κείμενο. Ακόμα, θα μπορούσε να εξερευνηθούν διάφοροι τρόποι αξιοποίησης του είδους των λέξεων και των σχέσεων των λέξεων με τεχνικές τύπου POS (Part-of-speech) tagging.

Όσο αφορά τα ίδια τα παράθυρα, μπορούν να γίνουν διάφορες επεκτάσεις όσο αφορά τον υπολογισμό των μεγεθών των παραθύρων ή και τον τρόπο που παράγονται τα παράθυρα. Για παράδειγμα, όπως φάνηκε στην συλλογή Time, πολλές φορές τα παραγόμενα μεγέθη παραθύρων για την μέθοδο σταθερού παράθυρου κειμένου, ήταν μεγαλύτερα από 400 λέξεις και ως συνέπεια δεν μπορούν ακριβώς να κατηγοριοποιηθούν ως παράθυρα προτάσεων. Οπότε, είναι δυνατόν όπως χρησιμοποιείται ένα κατώτερο όριο για τα μεγέθη των παραθύρων, να χρησιμοποιηθεί και ένα ανώτερο όριο. Επίσης, για την μέθοδο σταθερού παραθύρου παραγράφου, θα μπορούσαν να επεκταθεί κατάλληλα ώστε τα δύο επιμέρους γραφήματα να μπορούν να παραχθούν με οποιαδήποτε μέθοδο με παράθυρα.

Τέλος, μία σημαντική επέκταση που θα μπορούσε να γίνει είναι να εγκαταλειφθεί η δημιουργία αντεστραμμένων ευρετηρίων και η διαδικασία της ανάκτησης της πληροφορίας να γίνεται απαντώντας τα ερωτήματα κατευθείαν από το συνολικό γράφημα.**

7.1.2 Εντοπισμός ασήμαντων λέξεων

Ανακεφαλαιώνοντας, για τον εντοπισμό ασήμαντων λέξεων χρησιμοποιήθηκαν τρεις μέθοδοι, με διάφορους βαθμούς επιτυχίας. Οι δύο πρώτες μέθοδοι έχουν το πλεονέκτημα ότι τα στοιχεία που εξετάζουν για τον εντοπισμό των ασήμαντων λέξεων, δηλαδή

τους κύριους πυρήνες και τα ευρετήρια, παράγονται ως μέρος της κανονικής διαδικασίας της ανάκτησης πληροφορίας, με αποτέλεσμα να χρειάζεται πολύ λίγη επιπλέον δουλειά για τον εντοπισμό τον ασήμαντων λέξεων. Υπενθυμίζεται ότι η μέθοδο εντοπισμού ασήμαντων λέξεων στο κύριο πυρήνα του γραφήματος κειμένου εντόπισε, στην μέση περίπτωση, το 30% - 40% των ασήμαντων λέξεων του κάθε κειμένου, με την επέκταση στο συνολικό γράφημα να εντοπίζει το 67% των ασήμαντων λέξεων (στο κύριο πυρήνα) και η μέθοδο εντοπισμού ασήμαντων λέξεων στο ευρετήριο εντόπισε περίπου το 52% των ασήμαντων λέξεων. Η τρίτη μέθοδο, στην οποία δόθηκε και η περισσότερη προσοχή εντόπισε, στην βέλτιστη περίπτωση, το 63% των ασήμαντων λέξεων.

Από τα παραπάνω μπορούν να αντληθούν διάφορα συμπεράσματα, τόσο για την απόδοση των μεθόδων, όσο και την φύση των ασήμαντων λέξεων. Αρχικά, από τα αποτελέσματα της πρώτης μεθόδου είναι φανερό ότι στον κύριο πυρήνα των γραφημάτων των κειμένων, εκτός από ασήμαντες λέξεις, υπάρχουν και άλλα είδη λέξεων. Επίσης, ιδιαίτερο ενδιαφέρον φέρνουν οι διακυμάνσεις στον αριθμό των εντοπισμένων λέξεων που παρατηρήθηκαν τόσο σε κείμενα διαφορετικών μεγεθών, όσο και σε κείμενα ίδιου μεγέθους.***

Για την δεύτερη μέθοδο εκμεταλλεύτηκε η υπόθεση ότι οι ασήμαντες λέξεις έχουν υψηλή συχνότητα εμφάνισης στα κείμενα (document frequency) και ως αποτέλεσμα οι κόμβοι τους, στο συνολικό γράφημα, έχουν περισσότερες ακμές, και σε συνδυασμό με την διαδικασία παραγωγής των βαρών των όρων στο ευρετήριο πραγματοποιήθηκε ο εντοπισμός τους. Τα αποτελέσματα επιβεβαιώνουν ως ένα βαθμό τον παραπάνω ισχυρισμό.

Τέλος, με την τρίτη μέθοδο αποδείξαμε ότι χρησιμοποιώντας μόνο ένα σχετικά μικρό δείγμα κειμένων της συλλογής είναι δυνατόν να εντοπιστούν ένα μεγάλο ποσοστό από τις ασήμαντες λέξεις. Αυτό με την σειρά του, επίσης επιβεβαιώνει τον ισχυρισμό της προηγούμενης παραγράφου. Ότι δηλαδή, οι ασήμαντες λέξεις έχουν υψηλή συχνότητα εμφάνισης στα κείμενα και ότι εμφανίζονται στα πιο πολλά κείμενα της συλλογής, με μερικές εξαιρέσεις.

Ιδέες για επέκταση των μοντέλων εντοπισμού ασήμαντων λέξεων

Αρχικά, ακολουθώντας την φιλοσοφία της πρώτης μεθόδου θα μπορούσαν να ερευνηθούν και άλλοι πυρήνες εκτός από τον κύριο για ασήμαντες λέξεις.

Επιπρόσθετα, για την μέθοδο εντοπισμού ασήμαντων λέξεων χρησιμοποιώντας δείγμα της συλλογής, θα μπορούσε αρχικά να χρησιμοποιηθούν κάποια από τις άλλες μεθόδους που έχουν προταθεί, με κατάλληλη διαχείρισή των διαφορετικών μεγεθών των παραθύρων στην κανονικοποίηση. Ακόμα, είναι δυνατόν να γίνουν βελτιώσεις ή και αλλαγές στο τρόπο που υπολογίζεται ο βαθμός αξιολόγησης κάθε λέξης. Τέλος, η

πιο ενδιαφέρον ιδέα για επέκταση είναι η βελτίωση του τρόπου με τον οποίο επιλέγεται το δείγμα κειμένων της συλλογής. Ως τώρα το δείγμα επιλέγεται τυχαία ή για πειραματικούς λόγους ψευδοτυχαία. Με κατάλληλη ανάλυση των κειμένων, είτε ως προς παραμέτρους, σαν το μέγεθος τους, είτε ως προς το περιεχόμενο τους (πιο δύσκολο), ίσως να είναι δυνατόν να επιλεχθεί ένα δείγμα κειμένων τέτοιο ώστε να παράξει τα βέλτιστα δυνατά αποτελέσματα εντοπισμού.

Βιβλιογραφία

Altmann, E. G., & Gerlach, M. (2016). Statistical laws in linguistics. *Creativity and universality in language* (pp. 7–26). Springer International Publishing. https://doi.org/10.1007/978-3-319-24403-7_2

Baeza-Yates, R., & Ribeiro-Neto, B. (2008). *Modern information retrieval: The concepts and technology behind search* (2nd ed.). Addison-Wesley Publishing Company.

Bordag, S., Heyer, G., & Quasthoff, U. (2003). Small worlds of concepts and other principles of semantic search. In T. Böhme, G. Heyer, & H. Unger (Eds.), *Innovative internet community systems* (pp. 10–19). Springer Berlin Heidelberg.

Bush, V. (1996). As we may think. *Interactions*, *3*(2), 35–46. https://doi.org/10.1145/227181.227186

Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., & Quarteroni, S. (2013). The information retrieval process. *Web information retrieval* (pp. 13–26). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39314-3_2

Choudhury, M., Thomas, M., Mukherjee, A., Basu, A., & Ganguly, N. (2007). How difficult is it to develop a perfect spell-checker? a cross-linguistic analysis through complex network approach. *Proceedings of the Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing*, 81–88. https://aclanthology.org/W07-0212

Ferrer i Cancho, R., Capocci, A., & Caldarelli, G. (2007). Spectral methods cluster words of the same class in a syntactic dependency network. *International Journal of Bifurcation and Chaos*, *17*(07), 2453–2463. https://doi.org/10.1142/S021812740701852X

Ferrer i Cancho, R., Solé, R. V., & Köhler, R. (2004). Patterns in syntactic dependency networks. *Phys. Rev. E*, *69*, 051915. https://doi.org/10.1103/PhysRevE.69.051915

Flood, B. J. (1999). Historical note: The start of a stop list at biological abstracts. *Journal of the American Society for Information Science*, *50*(12), 1066–1066.

https://doi.org/https://doi.org/10.1002/(SICI)1097-4571(1999)50:12⟨1066::AID-ASI5⟩3.0.CO;2-A

Harman, D. (1993). Overview of the first trec conference. *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 36–47. https://doi.org/10.1145/160688.160692

Kalogeropoulos, N.-R., Doukas, I., Makris, C., & Kanavos, A. (2020). A graph-based extension for the set-based model implementing algorithms based on important nodes. In I. Maglogiannis, L. Iliadis, & E. Pimenidis (Eds.), *Artificial intelligence applications and innovations. aiai 2020 ifip wg 12.5 international workshops* (pp. 143–154). Springer International Publishing.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. [Doklady Akademii Nauk SSSR, V163 No4 845-848 1965]. *Soviet Physics Doklady*, *10*(8), 707–710.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, *1*(4), 309–317. https://doi.org/10.1147/rd.14.0309

Luhn, H. P. (1960). Key word-in-context index for technical literature (kwic index). *American Documentation*, *11*(4), 288–295. https://doi.org/https://doi.org/10.1002/asi.5090110403

Medeiros Soares, M., Corso, G., & Lucena, L. (2005). The network of syllables in portuguese. *Physica A: Statistical Mechanics and its Applications*, *355*(2), 678–684. https://doi.org/https://doi.org/10.1016/j.physa.2005.03.017

Possas, B., Ziviani, N., Meira, W., Jr., & Ribeiro-Neto, B. (2002). Set-based model: A new approach for information retrieval. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 230–237. https://doi.org/10.1145/564376.564417

Powers, D. (2008). Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. *Mach. Learn. Technol.*, *2*.

Rajaraman, A., & Ullman, J. D. (2011). Data mining. *Mining of massive datasets* (pp. 1–17). Cambridge University Press. https://doi.org/10.1017/CBO9781139058452.002

Rijsbergen, C. J. V. (1979). *Information retrieval* (2nd ed.). Butterworth-Heinemann.

Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, *3*(4), 333–389. https://doi.org/10.1561/1500000019

Rousseau, F., & Vazirgiannis, M. (2013a). Graph-of-word and tw-idf: New approach to ad hoc ir. *Proceedings of the 22Nd ACM International Conference on*

Information and Knowledge Management, 59–68. https://doi.org/10.1145/2505515.2505671

Rousseau, F., & Vazirgiannis, M. (2013b). Graph-of-word and tw-idf: New approach to ad hoc ir. *International Conference on Information and Knowledge Management, Proceedings*, 59–68. https://doi.org/10.1145/2505515.2505671

Rousseau, F., & Vazirgiannis, M. (2015). Main core retention on graph-of-words for single-document keyword extraction. *Advances in Information Retrieval*, 382–393.

Salton, G. [G.]. (1971). *The smart retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc.

Salton, G. [G.], Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, *18*(11), 613–620. https://doi.org/10.1145/361219.361220

Salton, G. [Gerard], & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, *24*(5), 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

Shaw, M., W., Wood, B., J., Wood, B., J., & Tibbo, R., H. (1991). The cystic fibrosis database: Content and research opportunities.

Singhal, A., & Google, I. (2001). Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, *24*.

Steven Bird, E. K., & Loper, E. (2009). *Natural language processing with python*. https://www.nltk.org/book/ch02.html

Test collections. (n.d.). http://ir.dcs.gla.ac.uk/resources/test_collections/

Tixier, A., Malliaros, F., & Vazirgiannis, M. (2016). A graph degeneracy-based approach to keyword extraction, 1860–1870. https://doi.org/10.18653/v1/D16-1191

Λίστα Αλγορίθμων

Κεφάλαιο 8

Παράρτημα

Πίνακες αποτελεσμάτων των μετρήσεων που πραγματοποιήθηκαν.

8.0.1 Αποτελέσματα Ανάκτησης Πληροφορίας

Πίνακες αποτελεσμάτων στην CF συλλογή

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μέγεθος του παραθύρου που χρησιμοποιήθηκε, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Σετ-βασεδ.

Test	GSB	Maincore	Density	Corerank	Constant
10	54	52	52	51	78
36	56	52	52	53	63
40	56	52	52	52	67
12	55	52	52	51	78
44	56	52	52	53	61
14	55	52	52	53	77
15	57	52	52	53	77
11	55	52	52	52	75
9	53	52	52	52	79
8	56	52	52	53	78
13	57	52	52	54	75
7	55	52	52	51	77
20	55	52	52	53	76
18	56	52	52	54	77
19	56	52	52	53	76
25	57	52	52	54	72
60	55	52	52	52	56
100	56	52	52	54	57

Table 8.1: Πρώτη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	GSB	Maincore	Density	Corerank	Constant%
0.1	55	52	52	53	70
0.025	56	52	52	52	75
0.05	56	52	52	52	73
0.015	55	52	52	51	76
0.01	55	52	52	53	78
0.005	56	52	52	52	77
0.0075	55	52	52	52	77
0.009	54	52	52	52	77
0.0125	56	52	52	54	74
0.011	55	52	52	51	79
0.0105	57	52	52	54	80
0.01025	55	52	52	51	78
0.01075	56	52	52	53	79

Table 8.2: Δεύτερη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου και οι μεταβλητές σημαντικότητας για την μέθοδο σταθερού παραθύρου παραγράφου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	GSB	Maincore	Density	Corerank	Constant%	Sen/Par
0.01055 200 a=1 b=0.1	58	52	52	54	81	67
0.0106 200 a=1 b=0.1	56	52	52	54	80	67
0.0106 200 a=1 b=0.01	55	52	52	51	79	73
0.01055 200 a=1 b=0.01	57	52	52	53	80	73
0.01055 200 a=1 b=0.2	55	52	52	52	79	66
0.01055 200 a=1 b=0.05	56	52	52	53	80	74
0.01055 150 a=1 b=0.1	56	52	52	54	79	73
0.01055 160 a=1 b=0.1	55	52	52	54	78	73
0.01055 150 a=1 b=0.05	55	52	52	50	79	73
0.02 150 a=1 b=0.05	55	52	52	53	76	74
0.0225 150 a=1 b=0.05	55	52	52	53	74	75
0.022 150 a=1 b=0.05	58	52	52	53	76	74
0.024 150 a=1 b=0.05	57	52	52	54	79	74
0.023 150 a=1 b=0.05	55	52	52	53	78	75
0.0235 150 a=1 b=0.05	53	52	52	51	78	74
0.02355 150 a=1 b=0.05	56	52	52	53	79	74
0.0236 150 a=1 b=0.05	58	52	52	52	78	74
0.0234 150 a=1 b=0.05	56	52	52	53	77	75
0.02352 150 a=1 b=0.05	55	52	52	52	77	74
0.02353 150 a=1 b=0.05	56	52	52	53	77	73
0.02354 150 a=1 b=0.05	56	52	52	53	78	75
0.02356 150 a=1 b=0.05	58	52	52	54	79	75
0.02357 150 a=1 b=0.05	56	52	52	52	78	74
0.023555 150 a=1 b=0.05	56	52	52	50	77	74

Table 8.3: Τρίτη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μπόνους που δίνεται στους σημαντικούς κόμβους των γραφημάτων, το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based. Στην συγκεκριμένη ομάδα πειραμάτων δεν χρησιμοποιήθηκε απαλοιφή ακμών.

Test	GSB	Maincore W/ SP	Density W/ SP	Corerank W/ SP	Constant%	Sen/Par
2000 0.01055 200	58	78	71	76	79	75
2000 0.01055 200	56	77	77	76	78	74
700 0.01055 200	55	79	79	75	80	73
700 0.01055 200	57	77	77	76	78	74
700 0.01055 200	55	77	77	76	79	75
700 0.01055 200	56	77	77	75	79	73
700 0.01055 200	56	78	78	76	79	74
1500 0.01055 200	55	79	79	76	80	74
1500 0.01055 200	55	78	78	76	80	75
1500 0.01055 200	55	79	79	76	80	74
1500 0.01055 200	55	79	79	76	80	75
5000 0.01055 200	58	78	78	75	79	74
1000 0.01055 200	57	77	77	76	77	75
100 0.01055 200	55	78	78	76	79	75
100 0.01055 200	53	79	79	76	80	74
100 0.01055 200	56	77	77	75	79	74
100 0.01055 200	58	78	78	76	79	75
500 0.01055 200	56	78	78	76	79	74
500 0.01055 200	55	78	78	76	79	74
500 0.01055 200	56	79	79	77	80	75
500 0.01055 200	56	78	78	76	79	74

Table 8.4: Τέταρτη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μπόνους που δίνεται στους σημαντικούς κόμβους των γραφημάτων, το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based. Στην συγκεκριμένη ομάδα πειραμάτων χρησιμοποιήθηκε απαλοιφή ακμών.

Test	GSB	Maincore W/ SP	Density W/ SP	Corerank W/ SP	Constant%	Sen/Par
700 0.2 0.01055 200	58	78	78	76	79	75
700 0.3 0.01055 200	56	79	79	75	80	75
700 0.4 0.01055 200	55	79	79	76	80	75
700 0.6 0.01055 200	57	77	77	76	79	74
700 1.2 0.01055 200	55	79	79	76	80	75
1500 0.3 0.01055 200	56	78	78	76	80	75
1500 0.4 0.01055 200	56	78	78	76	79	78
1500 0.6 0.01055 200	55	78	78	76	79	74
1500 1.2 0.01055 200	55	78	78	76	79	75
5000 0.3 0.01055 200	55	78	78	77	79	75
1000 0.3 0.01055 200	55	79	79	75	80	74
100 0.3 0.01055 200	58	78	78	76	79	74
100 0.4 0.01055 200	57	78	78	76	79	74
100 0.6 0.01055 200	55	77	77	75	78	74
100 1.2 0.01055 200	53	78	78	76	79	75
500 0.3 0.01055 200	56	78	78	76	79	75
500 0.4 0.01055 200	58	78	78	76	79	74
500 0.6 0.01055 200	56	77	77	76	78	75
500 1.2 0.01055 200	55	79	79	76	80	74

Table 8.5: Πέμπτη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μπόνους που δίνεται στους σημαντικούς κόμβους των γραφημάτων, το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based. Στην συγκεκριμένη ομάδα πειραμάτων δεν χρησιμοποιήθηκε απαλοιφή ακμών.

Test	GSB	Maincore W/ SPV	Density W/ SPV	Corerank W/ SPV	Vazirgiannis	Sen/Par W/ SPV
700 0.01055 200	58	70	70	68	68	64
700 0.01055 200	56	74	74	73	74	67
700 0.01055 200	55	74	74	73	74	67
700 0.01055 200	57	74	74	73	74	67
7000.01055 200	55	74	74	73	74	67
1500 0.01055 200	56	74	74	72	74	68
1500 0.01055 200	56	75	75	74	74	69
1500 0.01055 200	55	73	73	71	73	67
1500 0.01055 200	55	74	74	73	74	68
5000 0.01055 200	55	73	73	73	73	66
1000 0.01055 200	55	74	74	74	74	67
100 0.01055 200	58	75	75	70	74	67
100 0.01055 200	57	74	74	71	73	66
100 0.01055 200	55	75	75	72	74	67
100 0.01055 200	53	75	75	72	74	67
500 0.01055 200	56	74	74	74	74	68
500 0.01055 200	58	74	74	74	74	68
500 0.01055 200	56	73	73	72	73	66
500 0.01055 200	55	74	74	73	74	67

Table 8.6: Έκτη ομάδα πειραμάτων

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται, ο βαθμός της ποινής που δόθηκε στις ακμές στο συνολικό γράφημα, το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	GSB	Density	Corerank	Constant%	Sen/Par	Constant w/ d
0.01 0.01055 200	56	52	53	80	74	80
0.1 0.01055 200	56	52	53	79	74	78
0.05 0.01055 200	57	52	52	80	74	79
0.5 0.01055 200	55	52	52	79	73	75
0.025 0.01055 200	55	52	52	79	74	79
0.25 0.01055 200	55	52	53	79	73	77
0.03 0.01055 200	57	52	52	78	74	76
0.075 0.01055 200	55	52	52	79	74	78
0.0625 0.01055 200	55	52	51	78	73	78
0.058 0.01055 200	55	52	53	79	75	80
0.058 0.01055 200	55	52	51	79	74	80
0.055 0.01055 200	55	52	52	78	74	78
0.06 0.01055 200	57	52	53	80	74	81
0.06 0.01055 200	56	52	52	79	74	80
0.062 0.01055 200	56	52	53	79	75	79
0.061 0.01055 200	56	52	52	79	75	79
0.059 0.01055 200	56	52	53	79	75	80
0.0595 0.01055 200	56	52	51	79	74	80

Table 8.7: Έβδομη ομάδα πειραμάτων

Στον παρακάτω πίνακα φαίνονται οι μετρήσεις για τα πειράματα με και χωρίς ασήμαντες λέξεις στην συλλογή CF. Στην στήλη Test φαίνεται το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου και οι βαθμοί σημαντικότητας για την μέθοδο σταθερού παραθύρου παραγράφου, με αυτήν την σειρά. Στις υπόλοιπες στήλες φαίνεται ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	Constant % W/ Stopwords	Sen/Par W/ Stopwords	Constant % W/o Stopwords	Sen/Par W/o Stopwords
0.1 200 a=1 b=0.05	70		45	52
0.05 200 a=1 b=0.05	73		47	47
0.015 200 a=1 b=0.05	76		46	49
0.01 200 a=1 b=0.05	78		48	48
0.0125 200 a=1 b=0.05	74		48	47
0.011 200 a=1 b=0.05	79		49	48
0.01025 200 a=1 b=0.05	78		49	48
0.01075 200 a=1 b=0.05	79		50	49
0.01055 200 a=1 b=0.1	81	67	47	58
0.0106 200 a=1 b=0.1	80	67	47	48
0.01055 200 a=1 b=0.01	80	73	50	50
0.02 150 a=1 b=0.05	76	74	50	48
0.0225 150 a=1 b=0.05	74	75	48	46
0.022 150 a=1 b=0.05	76	74	47	48
0.024 150 a=1 b=0.05	79	74	48	46
0.023 150 a=1 b=0.05	78	75	47	47
0.0235 150 a=1 b=0.05	78	74	46	48
0.0234 150 a=1 b=0.05	77	75	49	48
0.023555 150 a=1 b=0.05	77	74	50	48

Table 8.8: Πειράματα με και χωρίς ασήμαντες λέξεις

Πίνακες αποτελεσμάτων στην NPL συλλογή

Ο πρώτος πίνακας αντιστοιχεί στα ερωτήματα, τα οποία τα μοντέλα που εξετάζονται απάντησαν καλύτερα από το Set-based μοντέλο και ο δεύτερος πίνακας αντιστοιχεί στα ερωτήματα, στα οποία τα μοντέλα που εξετάζονται απάντησαν ίδια με το Set-based μοντέλο. Στους πίνακες, στην στήλη Test φαίνεται το μπόνους που δίνεται στους σημαντικούς κόμβους των γραφημάτων, το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, το μέγεθος παραθύρου παραγράφου, με αυτή την σειρά.

Test	GSB	Corerank	Constant %	Sen/Par	Constant
700 0.01055 200	16	41	48	53	
700 0.01055 200	17	40	48	53	
700 0.01055 200	17	38	46	52	
700 0.01055 200	16	40	47	52	
700 0.01055 200	17	23	47	49	
1500 0.01055 200	17	21	46	50	
1500 0.01055 200	16	38	46	50	
1500 0.01055 200	17	40	47	48	
1500 0.01055 200	17	23	47	50	
5000 0.01055 200	16	20	47	50	
1000 0.01055 200	17	49	47	51	
100 0.01055 200	17	38	47	53	
100 0.01055 200	17	39	48	50	
100 0.01055 200	16	40	47	50	
100 0.01055 200	17	22	47	52	
500 0.01055 200	17	22	48	53	
500 0.01055 200	17	38	47	52	
500 0.01055 200	17	40	48	53	
500 0.01055 200	17	23	48	53	
2000 0.01055 150	17	22		52	46
200 7 150	17	23		50	47
200 10 150	16	22		54	48
2004 150	17	23		53	48
200 3 150	16	22		53	48
200 5 150	16	41		52	47

Table 8.9: Πειράματα στην NPL συλλογή που τα μοντέλα απάντησαν καλύτερα από το Set-based μοντέλο

Test	GSB	Corerank	Constant %	Sen/Par	Constant
700 0.01055 200	47	25	14	10	
700 0.01055 200	47	25	14	10	
700 0.01055 200	47	27	16	13	
700 0.01055 200	47	25	14	12	
700 0.01055 200	47	43	14	15	
1500 0.01055 200	47	43	15	14	
1500 0.01055 200	47	27	14	13	
1500 0.01055 200	47	25	15	14	
1500 0.01055 200	47	43	14	14	
5000 0.01055 200	47	44	15	14	
1000 0.01055 200	47	15	14	14	
100 0.01055 200	47	27	15	8	
100 0.01055 200	47	26	14	15	
100 0.01055 200	47	25	14	14	
100 0.01055 200	47	43	14	12	
500 0.01055 200	47	43	15	11	
500 0.01055 200	47	27	15	11	
500 0.01055 200	47	25	15	11	
500 0.01055 200	47	43	15	11	
2000 0.01055 150	47	42		11	15
200 7 150	47	42		13	15
200 10 150	47	43		10	15
200 4 150	47	43		10	14
200 3 150	47	43		11	15
200 5 150	47	25		10	14

Table 8.10: Πειράματα στην NPL συλλογή που τα μοντέλα απάντησαν ίδια με το Set-based μοντέλο

Πίνακες αποτελεσμάτων στην Cranfield συλλογή

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μέγεθος του παραθύρου, το μέγεθος παραθύρου παραγράφου και το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, με αυτή την σειρά, και στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	GSB	Constant % W/ d	Dot Split	Constant W/d	Sen/Par
3 200 0.01	143	161	154	165	165
8 200 0.02	144	164	155	162	164
9 200 0.03	143	162	154	161	159
11 200 0.04	143	163	153	162	160
3 200 0.01	143	161	154	165	165
8 200 0.02	144	164	155	162	164
9 200 0.03	143	162	154	161	159
11 200 0.04	143	163	153	162	160
12 200 0.05	143	160	155	162	165
14 200 0.06	144	161	154	162	162
15 200 0.07	142	163	154	159	161
18 200 0.08	144	159	156	163	162
20 200 0.09	144	163	152	158	162
13 200 0.1	145	163	154	162	162
2 200 0.009	145	162	153	161	159
3 200 0.01	142	161	153	164	165
8 200 0.02	143	162	154	160	164
9 200 0.03	143	163	154	162	159
11 200 0.04	144	164	155	163	159
12 200 0.05	144	160	155	162	165
14 200 0.06	141	161	152	160	162
15 200 0.07	142	161	153	159	161
18 200 0.08	143	158	156	163	162
20 200 0.09	143	163	153	158	162
13 200 0.1	144	164	155	163	162
2 200 0.009	145	162	153	161	159
16 200 0.008	145	162	154	162	163
14 200 0.007	143	160	153	160	161
9 200 0.006	142	161	153	162	160
9 200 0.005	144	161	154	162	159
9 200 0.004	145	161	153	162	158
9 200 0.003	144	162	154	163	158
9 200 0.002	143	161	154	162	159
9 200 0.001	142	160	153	161	160
5 150 0.031	142	161	153	164	165
4 150 0.032	144	162	155	164	165
6 150 0.033	144	164	154	161	162
7 150 0.034	145	160	153	160	164
3 150 0.035	145	165	153	163	164
14 150 0.036	142	162	154	166	162
15 150 0.037	143	162	155	163	162
18 150 0.038	146	162	154	160	161
20 150 0.039	143	161	155	163	163

Test	GSB	Constant % W/ d	Dot Split	Constant W/d	Sen/Par
13 150 0.041	141	162	152	157	161
2 150 0.042	144	162	155	163	163
16 150 0.043	145	162	155	161	163
14 150 0.044	143	163	153	162	163
9 150 0.045	143	162	154	162	161
9 150 0.046	144	161	155	161	164
9 150 0.047	143	160	153	161	164
9 150 0.048	142	160	155	162	164
9 150 0.049	143	160	155	162	164
9 150 0.0405	145	160	154	161	164
5 150 0.0341	143	160	152	162	164

Table 8.11: Πειράματα στην συλλογή Cranfield

Πειράματα στην συλλογή Time

Στον παρακάτω πίνακα, στην στήλη Test φαίνεται το μέγεθος του παραθύρου, το μέγεθος παραθύρου παραγράφου και το ποσοστό του κειμένου που χρησιμοποιήθηκε ως μέγεθος παραθύρου, με αυτή την σειρά. Επίσης, εμφανίζονται αλλαγές στις μεταβλητές σημαντικότητας για την μέθοδο σταθερού παραθύρου παραγράφου και αλλαγές στην ποινή στο συνολικό γράφημα. Τέλος, στις υπόλοιπες στήλες ο αριθμός των ερωτημάτων που απάντησαν οι μέθοδοι καλύτερα από το Set-based.

Test	GSB	Constant W/d	Constant %	Corerank	Constant	Sen/Par	Dot Split
3 200 0.01 a = 1 & b = 0.1	60		67	63	63	70	
8 200 0.02	59		67	63	66	67	
9 200 0.005	59		66	63	67	67	
11 200 0.03	59		67	63	67	67	
12 200 0.04	59		67	62	67	67	
14 200 0.05	59		67	63	67	66	
15 200 0.06	59		67	63	67	67	
18 200 0.07	59		68	63	67	66	
20 200 0.08	59		68	63	67	67	
13 200 0.09	59		68	62	67	67	
2 200 0.1	59		69	63	60	68	
16 200 0.009	59		66	62	67	66	
14 200 0.008	59		67	62	67	66	
9 200 0.007	59		67	63	67	67	
3 200 0.01	59		67	62	63	70	
8 200 0.02	59		67	62	66	67	
9 200 0.005	59		66	62	67	67	
11 200 0.03	59		67	63	67	68	
12 200 0.04	59		67	63	67	67	
14 200 0.05	59		67	63	67	66	
15 200 0.06	59		67	62	67	67	
18 200 0.07	59		68	62	67	66	
20 200 0.08	59		68	63	67	67	
3 200 0.01 d =0.05	59	67	67		63	70	67
8 200 0.02	59	67	67		66	67	67
9 200 0.005	60	66	66		67	67	66
11 200 0.03	60	67	67		67	68	67
12 200 0.04	59	67	67		67	67	67
14 200 0.05	59	67	67		67	66	67
15 200 0.06	59	67	67		67	67	67
18 200 0.07	59	68	68		67	66	68
20 200 0.08	59	68	68		67	67	68
13 200 0.09	59	69	69		67	66	69
2 200 0.1	59	68	68		60	68	68
16 200 0.009	59	66	66		67	66	66
14 200 0.008	59	67	67		67	66	67
9 200 0.007	59	67	67		67	67	67
9 200 0.006	59	65	65		67	67	65
9 200 0.004	59	66	66		67	67	66

Test	GSB	Constant W/d	Constant %	Corerank	Constant	Sen/Par	Dot Split
9 200 0.003	59	62	62		67	67	62
9 200 0.002	59	63	63		67	67	63
9 200 0.001	59	63	63		67	67	63
3 200 0.091	59	68	68		63	70	68
8 200 0.092	59	68	68		66	67	68
9 200 0.093	59	68	68		67	67	68
11 200 0.094	59	68	68		67	68	68
12 200 0.095	59	68	68		67	67	68
14 200 0.096	59	68	68		67	66	68
15 200 0.097	59	69	69		67	67	69
18 200 0.098	59	68	68		67	66	68
20 200 0.099	60	68	68		67	67	68
13 200 0.1	59	68	68		67	66	68
2 200 0.089	59	68	68		60	68	68
16 200 0.088	59	69	69		67	66	69
14 200 0.087	59	68	68		67	66	68
9 200 0.086	59	68	68		67	67	68
9 200 0.085	59	68	68		67	67	68
9 200 0.084	59	68	68		67	67	68
9 200 0.083	59	68	68		67	67	68
9 200 0.082	59	69	69		67	67	69
9 200 0.081	59	68	68		67	67	68
3 200 0.091 d=0.001 a=1 & b=0.05	59	68	68		63	70	68
9 200 0.082	59	68	68		63	70	68
9 200 0.081	59	68	68		63	70	68
3 200 0.091 pen 0.5	58	67	67		62	66	67
8 200 0.092	58	67	67		66	69	67
9 200 0.093	58	66	66		66	70	67
11 200 0.094	58	67	67		66	69	67
12 200 0.095	58	67	67		67	68	67
14 200 0.096	58	67	67		66	69	67
15 200 0.097	59	67	67		66	68	67
18 200 0.098	59	66	66		65	68	66
20 200 0.099	58	67	67		66	67	67
13 200 0.1	58	65	65		65	68	66
2 200 0.089	58	67	67		62	66	67
16 200 0.088	58	67	67		66	67	67
14 200 0.087	58	67	67		66	69	67
9 200 0.086	58	66	66		65	70	66
9 200 0.085	58	67	67		66	70	67
9 200 0.084	58	67	67		66	70	67
9 200 0.083	58	67	67		66	70	67
9 200 0.082	58	67	67		66	70	67
9 200 0.081	58	67	67		66	70	67
3 200 0.091 pen 0.1	58	67	67		67	68	66
8 200 0.092	58	65	65		67	68	66
9 200 0.093	58	67	67		66	69	66

Test	GSB	Constant W/d	Constant %	Corerank	Constant	Sen/Par	Dot Split
11 200 0.094	58	66	66		66	69	65
12 200 0.095	58	66	66		65	68	65
14 200 0.096	58	66	66		65	68	65
15 200 0.097	58	66	66		66	69	66
18 200 0.098	58	67	67		66	69	66
20 200 0.099	58	64	64		66	69	66
13 200 0.1	58	67	67		62	65	66
2 200 0.089	58	67	67		66	68	66
16 200 0.088	59	67	67		66	68	66
14 200 0.087	58	65	65		67	68	66
9 200 0.086	58	66	66		67	68	66
9 200 0.085	58	66	66		65	68	66
9 200 0.084	58	66	66		67	68	66
9 200 0.083	58	66	66		67	68	66
9 200 0.082	58	65	65		66	68	65
9 200 0.081	58	67	67		67	68	66

Table 8.12: Πειράματα στην συλλογή Time

8.0.2 Αποτελέσματα Εντοπισμού Ασήμαντων Λέξεων με δείγμα της συλλογής

3 Δείγματα

mod 3 no bonus	Bucket 1	
Λέξεις	Precision	Recall
340	0.588235294	0.588235294
10	0.9	0.026470588
25	0.96	0.070588235
50	0.92	0.135294118
100	0.85	0.25
200	0.725	0.426470588
450	0.524444444	0.694117647
1000	0.293	0.861764706
	Bucket 2	
	Precision	Recall
340	0.608823529	0.608823529
10	0.9	0.026470588
25	0.96	0.070588235
50	0.92	0.135294118
100	0.86	0.252941176
200	0.705	0.414705882
450	0.528888889	0.7
1000	0.293	0.861764706
	Bucket 3	
	Precision	Recall
340	0.591176471	0.591176471
10	0.9	0.026470588
25	0.96	0.070588235
50	0.92	0.135294118
100	0.86	0.252941176
200	0.7	0.411764706
450	0.513333333	0.679411765
1000	0.287	0.844117647

Table 8.13: Αποτελέσματα με 3 δείγματα

mod 4 no bonus	Bucket 1	
Λέξεις	Precision	Recall
340	0.597058824	0.597058824
10	0.9	0.026470588
25	0.96	0.070588235
50	0.92	0.135294118
100	0.85	0.25
200	0.71	0.417647059
450	0.522222222	0.691176471
1000	0.289	0.85
	Bucket 2	
	Precision	Recall
340	0.602941176	0.602941176
10	0.9	0.026470588
25	0.96	0.070588235
50	0.94	0.138235294
100	0.89	0.261764706
200	0.7	0.411764706
450	0.515555556	0.682352941
1000	0.29	0.852941176
	Bucket 3	
	Precision	Recall
340	0.576470588	0.576470588
10	0.9	0.026470588
25	0.96	0.070588235
50	0.92	0.135294118
100	0.84	0.247058824
200	0.71	0.417647059
450	0.524444444	0.694117647
1000	0.287	0.844117647
	Bucket 4	
	Precision	Recall
340	0.588235294	0.588235294
10	0.9	0.026470588
25	0.96	0.070588235
50	0.96	0.141176471
100	0.85	0.25
200	0.7	0.411764706
450	0.508888889	0.673529412
1000	0.289	0.85

Table 8.14: Αποτελέσματα με 4 δείγματα

mod 8 no bonus	Bucket 1		Bucket 2	
Λέξεις	Precision	Recall	Precision	Recall
340	0.579411765	0.579411765	0.597058824	0.597058824
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.94	0.138235294
100	0.86	0.252941176	0.9	0.264705882
200	0.69	0.405882353	0.7	0.411764706
450	0.508888889	0.673529412	0.504444444	0.667647059
1000	0.284	0.835294118	0.285	0.838235294
	Bucket 3		Bucket 4	
	Precision	Recall	Precision	Recall
340	0.544117647	0.544117647	0.570588235	0.570588235
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.98	0.144117647
100	0.83	0.244117647	0.85	0.25
200	0.675	0.397058824	0.675	0.397058824
450	0.48	0.635294118	0.484444444	0.641176471
1000	0.278	0.817647059	0.272	0.8
	Bucket 5		Bucket 6	
	Precision	Recall	Precision	Recall
340	0.579411765	0.579411765	0.555882353	0.555882353
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.94	0.138235294
100	0.87	0.255882353	0.82	0.241176471
200	0.705	0.414705882	0.665	0.391176471
450	0.497777778	0.658823529	0.484444444	0.641176471
1000	0.268	0.788235294	0.274	0.805882353
	Bucket 7		Bucket 8	
	Precision	Recall	Precision	Recall
340	0.552941176	0.552941176	0.576470588	0.576470588
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.94	0.138235294
100	0.84	0.247058824	0.84	0.247058824
200	0.7	0.411764706	0.695	0.408823529
450	0.506666667	0.670588235	0.504444444	0.667647059
1000	0.275	0.808823529	0.284	0.835294118

Table 8.15: Αποτελέσματα με 8 δείγματα

mod 6 no bonus	Bucket 1		Bucket 2	
Λέξεις	Precision	Recall	Precision	Recall
340	0.6	0.6	0.605882353	0.605882353
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.94	0.138235294
100	0.85	0.25	0.87	0.255882353
200	0.7	0.411764706	0.705	0.414705882
450	0.517777778	0.685294118	0.526666667	0.697058824
1000	0.284	0.835294118	0.287	0.844117647
	Bucket 3		Bucket 4	
	Precision	Recall	Precision	Recall
340	0.567647059	0.567647059	0.579411765	0.579411765
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.94	0.138235294	0.94	0.138235294
100	0.86	0.252941176	0.86	0.252941176
200	0.685	0.402941176	0.69	0.405882353
450	0.5	0.661764706	0.493333333	0.652941176
1000	0.285	0.838235294	0.277	0.814705882
	Bucket 5		Bucket 6	
	Precision	Recall	Precision	Recall
340	0.579411765	0.579411765	0.558823529	0.558823529
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.96	0.141176471
100	0.84	0.247058824	0.87	0.255882353
200	0.71	0.417647059	0.71	0.417647059
450	0.497777778	0.658823529	0.5	0.661764706
1000	0.285	0.838235294	0.279	0.820588235

Table 8.16: Αποτελέσματα με 6 δείγματα

mod 6 no bonus w = 1	Bucket 1		Bucket 2	
Λέξεις	Precision	Recall	Precision	Recall
340	0.623529412	0.623529412	0.647058824	0.647058824
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.96	0.141176471	0.96	0.141176471
100	0.92	0.270588235	0.88	0.258823529
200	0.74	0.435294118	0.765	0.45
450	0.537777778	0.711764706	0.542222222	0.717647059
1000	0.289	0.85	0.288	0.847058824
	Bucket 3		Bucket 4	
	Precision	Recall	Precision	Recall
340	0.6	0.6	0.6	0.6
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.98	0.144117647	0.96	0.141176471
100	0.9	0.264705882	0.88	0.258823529
200	0.735	0.432352941	0.745	0.438235294
450	0.517777778	0.685294118	0.513333333	0.679411765
1000	0.29	0.852941176	0.282	0.829411765
	Bucket 5		Bucket 6	
	Precision	Recall	Precision	Recall
340	0.614705882	0.614705882	0.594117647	0.594117647
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.96	0.141176471
100	0.92	0.270588235	0.92	0.270588235
200	0.745	0.438235294	0.745	0.438235294
450	0.517777778	0.685294118	0.515555556	0.682352941
1000	0.291	0.855882353	0.285	0.838235294

Table 8.17: Αποτελέσματα με 6 δείγματα και μέγεθος παραθύρου ίσο με 1

mod 6 w/ bonus = 5	Bucket 1		Bucket 2	
Λέξεις	Precision	Recall	Precision	Recall
340	0.611764706	0.611764706	0.635294118	0.635294118
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.96	0.141176471	0.94	0.138235294
100	0.9	0.264705882	0.89	0.261764706
200	0.72	0.423529412	0.73	0.429411765
450	0.526666667	0.697058824	0.542222222	0.717647059
1000	0.287	0.844117647	0.287	0.844117647
	Bucket 3		Bucket 4	
	Precision	Recall	Precision	Recall
340	0.579411765	0.579411765	0.597058824	0.597058824
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.94	0.138235294	0.96	0.141176471
100	0.87	0.255882353	0.87	0.255882353
200	0.71	0.417647059	0.725	0.426470588
450	0.508888889	0.673529412	0.508888889	0.673529412
1000	0.289	0.85	0.282	0.829411765
	Bucket 5		Bucket 6	
	Precision	Recall	Precision	Recall
340	0.605882353	0.605882353	0.576470588	0.576470588
10	0.9	0.026470588	0.9	0.026470588
25	0.96	0.070588235	0.96	0.070588235
50	0.92	0.135294118	0.96	0.141176471
100	0.87	0.255882353	0.89	0.261764706
200	0.735	0.432352941	0.735	0.432352941
450	0.506666667	0.670588235	0.506666667	0.670588235
1000	0.287	0.844117647	0.28	0.823529412

Table 8.18: Αποτελέσματα με 6 δείγματα και αναλογία σκελών βαθμού αξιολόγησης 1:5

8.0.3 Συνάρτηση απεικόνησης γραφήματος και διαγράμματος βαθμού ακμών

Τέλος, ακολουθεί μία συνάρτηση που παράγει διαγράμματα βαθμού ακμών ενός γραφήματος, καθώς και μία απλοποιημένη απεικόνηση του γραφήματος.

```
# ----------------------Graph visualization----------------------
def getGraphStats(graph,filename,graphPng, degreePng):
        if nx.is_connected(graph):
                print("IT IS CONNECTED")
        name = filename[10:]
        if graphPng:
                graphToPng(graph = graph, filename = str(name))
        if degreePng:
                plot_degree_dist(graph = graph, filename = str(name))

def graphToPng(graph, *args, **kwargs):
    options = {
        'node_color': 'yellow',
        'node_size': 50,
        'linewidths': 0,
        'width': 0.1,
        'font_size': 8,
    }
    filename = kwargs.get('filename', None)
    if not filename:
        filename = 'Union graph'
    plt.figure(filename, figsize=(17, 8))
    plt.suptitle(filename)
    pos_nodes = nx.circular_layout(graph)
    nx.draw(graph, pos_nodes, with_labels=True, **options)
    pos_atrs = {}
    for node, coords in pos_nodes.items():
        pos_atrs[node] = (coords[0], coords[1] + 0.01)
    node_attrs = nx.get_node_attributes(graph, 'term')
    cus_node_att = {}
    for node, attr in node_attrs.items():
        cus_node_att[node] = attr
    nx.draw_networkx_labels(graph, pos_atrs, labels=cus_node_att,
        font_color='red', font_size=8)
    labels = nx.get_edge_attributes(graph, 'weight')
    nx.draw_networkx_edge_labels(graph, pos_nodes, edge_labels=labels)
    #plt.show()
    plt.savefig('figures/allq/'+str(filename)+'.png',format="PNG", dpi
        =600)

def plot_degree_dist(graph, *args, **kwargs):
    filename = kwargs.get('filename', None)
    degree_sequence = sorted([d for n, d in graph.degree()], reverse=
        True)  # degree sequence
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    fig, ax = plt.subplots()
    plt.bar(deg, cnt, width=0.80, color="b")
    plt.title("Degree Histogram")
    plt.ylabel("Count")
    plt.xlabel("Degree")
    ax.set_xticks([d + 0.4 for d in deg])
    plt.setp(ax.get_xticklabels(), rotation=90, horizontalalignment='
        right', fontsize=3)
    ax.set_xticklabels(deg)
```

```
    # draw graph in inset
    plt.axes([0.4, 0.4, 0.5, 0.5])
    Gcc = graph.subgraph(sorted(nx.connected_components(graph), key=len
        , reverse=True)[0])
    pos = nx.spring_layout(graph)
    plt.axis("off")
    nx.draw_networkx_nodes(graph, pos, node_size=20)
    nx.draw_networkx_edges(graph, pos, alpha=0.4)

    plt.savefig('figures/allq/'+str(filename)+'_degree.png',format="PNG
        ", dpi=600)
```

Αλγόριθμος 8.1: Graph Visualization