ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΪΑΤΡΙΚΗΣ

# Εξόρυξη κειμένου στην ανασκόπηση βιβλιογραφίας και την τεκμηριωμένη ιατρική

**Γρηγοροπούλου Μαρίνα**
**Αριθμός Μητρώου: 48014016**

**Επιβλέπων Καθηγητής**
**Δρ. Ζουμπουλάκης Παναγιώτης, Αναπληρωτής Καθηγητής**

**Αθήνα 29/09/2021**

Η Τριμελής Εξεταστική Επιτροπή

| Ο Επιβλέπων Καθηγητής | | |
|---|---|---|
| Παναγιώτης Ζουμπουλάκης | Θεοδώρα Κατσίλα | Διονύσιος Κάβουρας |
| Αναπληρωτής καθηγητής | Εντεταλμένη Ερευνήτρια | Ομότιμος Καθηγητής |
| [ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ] | | [ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ] |
| Panagiotis Zoumpoulakis Digitally signed by Panagiotis Zoumpoulakis Date: 2021.11.04 17:13:02 +02'00' | | Dionysios Kavouras Digitally signed by Dionysios Kavouras Date: 2021.11.04 13:45:03 +02'00' |

**ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ**

Η υπογράφουσα Μαρίνα Γρηγοροπούλου του Χαρισίου, με αριθμό μητρώου 48014016 φοιτήτρια του Τμήματος Μηχανικών Βιοϊατρικής Τεχνολογίας της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου».

Ημερομηνία

03/11/2021

Ο/Η Δηλών/ούσα

## Περίληψη

Η εφαρμογή της τεκμηριωμένης ιατρικής απαιτεί την συνεχή ενημέρωση του ιατρικού προσωπικού σχετικά με τις βέλτιστες πρακτικές, πράγμα ολοένα και πιο δύσκολο λόγω του αυξανόμενου όγκου νέων επιστημονικών δημοσιεύσεων. Αυτή η εργασία ερευνά την εφαρμογή τεχνικών εξόρυξης κειμένου σε προβλήματα ανασκόπησης βιβλιογραφίας. Για τον σκοπό αυτό υλοποιήθηκαν προγραμματιστικές τεχνικές πρόσβασης σε αποθετήρια επιστημονικών δημοσιεύσεων, όπου αποκτήθηκε μεγάλος όγκος δεδομένων κειμένου. Στη συνέχεια υλοποιήθηκαν δυο διαφορετικές τεχνικές αναζήτησης για την ανάκτηση δεδομένων σχετικών με την εξόρυξη κειμένου στην τεκμηριωμένη ιατρική. Η πρώτη τεχνική βασίστηκε στο full text search της PostgreSQL για την αναζήτηση λέξεων και φράσεων κλειδιών. Η δεύτερη τεχνική που υλοποιήθηκε ήταν η Latent Semantic Analysis – LSA μια μαθηματική τεχνική που βασίζεται στην ανάλυση σε ιδιάζουσες τιμές (Singular Value Decomposition - SVD) για την αναπαράσταση των δεδομένων σε έναν σημασιολογικό χώρο μειωμένων διαστάσεων. Τα αποτελέσματα αξιολογήθηκαν μετά από χειροκίνητο χαρακτηρισμό των εγγράφων ως σχετικά και μη, με ενθαρρυντικά αποτελέσματα.

***Λέξεις Κλειδιά: Τεκμηριωμένη Ιατρική, Εξόρυξη Κειμένου, Βάσεις Δεδομένων, R, SQL, Latent Semantic Analysis, Ανάλυση σε Ιδιάζουσες Τιμές***

## Abstract

Evidence-based medicine in practice requires the continuous education of medical personnel according to the best up to date medical practices, an increasingly difficult task due to the growing volume of new published research. This thesis explores the implementation of text mining practices under the scope of literature reviews. For this purpose, a large corpus was acquired through programmatical access to scientific publication repositories. Subsequently, two different searching techniques were implemented to retrieve texts concerning text mining and evidence-based medicine. The first method utilized PostgreSQL full-text search to carry out keyword and phrase searches. The second method implemented was Latent Semantic Analysis – LSA, a mathematical technique that relies on Singular Value Decomposition – SVD to project the data in a reduced dimension semantic space. Results were evaluated after manual labelling of records as relevant or non-relevant, with promising results.

***Keywords: Evidence-based Medicine, Text Mining, Databases, R, SQL, Latent Semantic Analysis, Singular Value Decomposition***

**Ευχαριστίες:**

Θα ήθελα να ευχαριστήσω τον Δρ. Παναγιώτη Ζουμπουλάκη για την επίβλεψη της εργασίας και τον καθηγητή Δρ. Διονύσιο Κάβουρα που μου έδωσε την ευκαιρία να λάβω μέρος στο σεμινάριο Μηχανικής Μάθησης που οργάνωσε το καλοκαίρι του 2020, βοηθώντας με να έρθω σε στενότερη επαφή με αυτή την εξαιρετικά ενδιαφέρουσα επιστήμη.

Επίσης, θα ήθελα να ευχαριστήσω την Δρ. Θεοδώρα Κατσίλα για την πολύτιμη στήριξη και καθοδήγηση καθ' όλη την διάρκεια της εργασίας. Ευχαριστώ επίσης τον συνάδελφο μου υποψήφιο διδάκτωρ Σωτήρη Ουζούνη για την βοήθεια και τις συμβουλές του.

# Περιεχόμενα

# 1. Εισαγωγή

Σε αυτό το κεφάλαιο, αρχικά γίνεται μια γενική εισαγωγή στη θεματολογία της εργασίας. Στη συνέχεια παρουσιάζεται μια σύντομη βιβλιογραφική ανασκόπηση σχετικά με τεχνικές εξόρυξης κειμένου στην βιοϊατρική επιστήμη καθώς και την τεκμηριωμένη ιατρική. Έπειτα περιγράφονται ο σκοπός και οι στόχοι της εργασίας καθώς και ο τρόπος που προσεγγίστηκε το θέμα, με την τελευταία ενότητα να παρουσιάζει τη δομή της εργασίας που θα ακολουθήσει.

## 1.1. Γενική εισαγωγή στο θέμα

Η τεκμηριωμένη ιατρική αναφέρεται στην εκούσια και συνετή εφαρμογή των τρεχόντων στοιχείων και αποδείξεων στην διαδικασία λήψης αποφάσεων για κάθε ασθενή. Υπό αυτόν τον ορισμό, η τεκμηριωμένη ιατρική συνδυάζει την κλινική εμπειρία του ιατρού με μία αξιολόγηση της τρέχουσας καλύτερης εξωτερικής γνώσης από συστηματική έρευνα [1]. Ως κλινική εμπειρία αναφέρεται η κρίση και η ευχέρεια του ιατρού που αποκτάται από την έμπρακτη νοσοκομειακή πρακτική. Η εμπειρία αυτή εκφράζεται με πολλούς τρόπους, όπως η αποτελεσματική αναγνώριση των δυσκολιών, προτιμήσεων και δικαιωμάτων του κάθε ασθενή κατά την λήψη κλινικών αποφάσεων για την φροντίδα τους. Με εξωτερική γνώση αναφέρονται τα δεδομένα που προέρχονται από την κλινική έρευνα και αφορούν τα προγνωστικά σημάδια, την ακρίβεια των διαγνωστικών πειραμάτων και την αποτελεσματικότητα των θεραπευτικών, αναρρωτικών και προληπτικών θεραπειών. Τέτοια δεδομένα μπορούν είτε να ακυρώσουν προηγουμένως αποδεκτές θεραπείες ή τεστ και να τα αντικαταστήσουν με καινούργια, είτε να επιβεβαιώσουν την αξία των τρεχόντων πρακτικών.

Ο συνδυασμός της εμπειρίας του ιατρικού προσωπικού με την εξωτερική έρευνα είναι ιδιαίτερα σημαντικός και κανένα από τα δύο κομμάτια δεν επαρκεί μόνο του για την βέλτιστη αποτελεσματικότητα της πρακτικής. Χωρίς πρακτική κλινική εμπειρία είναι εύκολο να ακολουθούνται τυφλά δεδομένα και έρευνες οι οποίες δεν είναι εφαρμοστέες στην εκάστοτε περίπτωση. Αντίστοιχα, χωρίς τακτική ενσωμάτωση των αποτελεσμάτων της τωρινής έρευνας στην πρακτική, ο κίνδυνος να εφαρμοστεί μία απαρχαιωμένη θεραπεία αυξάνεται, εις βάρος των ασθενών.

Ενώ η εφαρμογή της τεκμηριωμένης ιατρικής, μίας ιδέας που αναπτύχθηκε στα μέσα του 19$^{ου}$ αιώνα, υπόσχεται αποτελεσματικότερη πρακτική, η εφαρμογή της έχει πρακτικά εμπόδια. Δεδομένα και έρευνες δημοσιεύονται σε τεράστιο όγκο με ταχύτατους ρυθμούς και ο αριθμός των δημοσιεύσεων που αφορούν κάθε πεδίο της επιστήμης, συμπεριλαμβανομένων των ιατρικών επιστημών, όλο και αυξάνεται. Σύμφωνα με τους Hunter και Cohen [2], το 2005 δημοσιεύθηκαν στο MEDLINE (η βάση δεδομένων του National Library of Medicine [3]), 666,029 άρθρα· περισσότερα από 1800 ημερησίως. Η συστηματική ενημέρωση για τις τελευταίες εξελίξεις σε κάθε κλάδο καταλήγει εξαιρετικά δύσκολη καθώς οι απαιτήσεις σε χρόνο αυξάνονται εκθετικά με τον όγκο της πληροφορίας. Επιπροσθέτως, γίνεται όλο και ευκολότερο να χαθεί στον θόρυβο μία σχετική με κάποιο σενάριο έρευνα, δυσχεραίνοντας την βιβλιογραφική ανασκόπηση που περιγράφεται στο πλαίσιο της τεκμηριωμένης ιατρικής. Μία λύση που προτείνεται σε αυτά τα αδιέξοδα είναι η εφαρμογή τεχνικών εξόρυξης κειμένου (text mining).

Οι τεχνικές εξόρυξης κειμένων, αντλώντας από την επεξεργασία φυσικής γλώσσας (natural language processing), της μηχανικής μάθησης (machine learning) και της θεωρίας ανάκτησης πληροφορίας (information retrieval), είναι σε θέση να εξάγουν πληροφορίες ακόμα και από μεγάλες συλλογές κειμένου. Ενώ δεν είναι σε θέση να αντικαταστήσουν ανθρώπους σε πολύπλοκες διαδικασίες, η εφαρμογές τέτοιων αυτοματοποιημένων τεχνικών μπορεί να βοηθήσουν στην ανασκόπηση βιβλιογραφίας και στην ανακάλυψη πληροφοριών που χάνονται μέσα στον όγκο των διαθέσιμων πληροφοριών. Κατά κάποιον τρόπο, η εξόρυξη κειμένων είναι μία επέκταση των μηχανών αναζήτησης που καθιστούν ένα κοινό εργαλείο εδώ και πολλά χρόνια.

## 1.2. Βιβλιογραφική ανασκόπηση

Η εκθετική αύξηση του διαθέσιμου ερευνητικού υλικού και των δημοσιεύσεων έχουν κάνει την ανασκόπηση βιβλιογραφίας ιδιαίτερα δύσκολη, δυσχεραίνοντας το έργο των επιστημόνων που επιθυμούν να αφομοιώσουν τις εξελίξεις σχετικές με την έρευνα τους. Το ίδιο εμπόδιο συναντάται στην τεκμηριωμένη ιατρική, όπου για κάθε ασθενή εκτελείται εκτεταμένη βιβλιογραφική ανασκόπηση ώστε να εντοπιστούν οι αρμόζουσες κλινικές μελέτες. Οι Allen και Harkins [4] περιέγραψαν το 2005 ένα σενάριο μίας 24-ώρης βάρδιας με 18 ασθενείς και συνολικά 44 διαγνώσεις. Τα βιβλία οδηγιών του National Institute of Clinical Excellence (NICE) του Ηνωμένου Βασιλείου που άρμοζαν στις διαγνώσεις αυτές και ο εφημερεύοντας ιατρός έπρεπε να είχε διαβάσει και ακολουθήσει, είχαν 3679 σελίδες. Είναι προφανές πως ο όγκος της πληροφορίας είναι πολύ μεγάλος για να είναι πρακτική η αφομοίωση της από το ιατρικό προσωπικό χωρίς βοήθεια, ιδιαίτερα αν ληφθεί υπ' όψει ότι τα προαναφερθέντα βιβλία οδηγιών περιέχουν την πληροφορία σε συμπυκνωμένη μορφή. Στην συνέχεια θα παρουσιαστούν διάφορες εφαρμογές εξόρυξης κειμένου στην βιοϊατρική επιστήμη, συμπεριλαμβανομένης της τεκμηριωμένης ιατρικής.

Οι Cheng κ.ά.[5] παρουσίασαν το 2020 επισκόπηση της βιβλιογραφίας πάνω στις COVID-19, MERS και SARS, εφαρμόζοντας Latent Dirichlet Allocation για την ομαδοποίηση των δημοσιεύσεων και εξαγωγή τυχών σχέσεων ανάμεσα στην βιβλιογραφία για κάθε ασθένεια. Στην έρευνα τους χρησιμοποιήθηκε η LDA για ανακάλυψη θεματικών ενοτήτων στην βιβλιογραφία και τον προσδιορισμό των «αποστάσεων» ανάμεσα τους. Δεν γίνεται περεταίρω εμβάθυνση αλλά φαίνεται πως τεχνικές εξαγωγής πληροφορίας μπορούν να είναι χρήσιμες στην βιοϊατρική επιστήμη, βοηθώντας να δαμαστεί ο όγκος της βιβλιογραφίας.

Οι Choi κ.ά. [6], σε συνεργασία με ερευνητές της IBM, δημοσίευσαν το 2018 μία έρευνα πάνω στην αυτόματη ανακάλυψη σχέσεων μέσα από βιβλιογραφική ανασκόπηση. Από ένα σύνολο 21 εκατομμυρίων δημοσιεύσεων, επιλέχθηκαν περίπου 130,000 που εμπεριείχαν αναφορές σε κινάσες και μέσω αλγορίθμων παραγωγής υποθέσεων (hypothesis generation), εντοπίστηκαν κινάσες που ήταν πιθανό να προκαλέσουν φωσφορυλίωση της ογκοπρωτεΐνης p53. Για έξι από αυτές τις κινάσες αποδείχθηκε πειραματικά η σχέση. Η παραγωγή των υποθέσεων έγινε μέσω συνδέσεων Swanson [7], υποθέτοντας ότι αν υπάρχει μία σχέση ανάμεσα στο Α με το Β, και στο Β με το Γ, τότε είναι πιθανών να υπάρχει σχέση ανάμεσα στο Α και το Γ. Η εξαγωγή των αρχικών σχέσεων από τις δημοσιεύσεις έγινε μέσω εξόρυξης κειμένου.

Το 2021 οι Henry κ.ά. [8] παρουσίασαν εφαρμογή Literature Based Discovery σε μία έρευνα βασισμένη σε εξόρυξη κειμένου για την αναγνώριση μεταβολιτών που σχετίζονται με την καρδιακή ανακοπή. Με παρόμοιο τρόπο με αυτόν που παρουσιάστηκε στην προηγούμενη παράγραφο, αναπτύχθηκαν σχέσεις Swanson ανάμεσα σε βιβλιογραφία σχετική με μεταβολίτες ως ότου να αποκαλυφθούν πιθανές, προηγουμένως άγνωστες, σχέσεις. Μερικές από αυτές επιβεβαιώθηκαν με in vivo έρευνες.

Εκτός από χρήση εξόρυξης κειμένου για την τροφοδότηση νέων ανακαλύψεων, τέτοιες τεχνικές μπορούν να βοηθήσουν τον ερευνητή στην «απλή» αναζήτηση σχετικής βιβλιογραφίας. Ενώ οι μηχανές αναζήτησης έχουν γίνει κάτι ευρέως διαδεδομένο στις ζωές όλων εδώ και πολλά χρόνια, στην πραγματικότητα είναι ένα επιστημονικό πεδίο με συνεχής ανάπτυξη. Το βιβλίο «Understanding Search Engines» των M. Berry και M. Browne [9] προσφέρεται ως μία εισαγωγή στον κόσμο των μηχανών αναζήτησης, αναλύοντας τις βασικές τεχνικές αναζήτησης και εξαγωγής πληροφορίας από κείμενα. Η S. Dumais περιγράφει με λεπτομέρεια, σε άρθρο της το 2005[10], την μέθοδο Latent Semantic Analysis, η οποία θα χρησιμοποιηθεί σε αυτή την εργασία. Αξίζει να σημειωθεί πως μαζί με τον W. Furnas, τον T. Landauer και άλλους συνεργάτες τους, είχαν δημοσιεύσει για την LSA από το 1988 [11], όσο εργάζονταν για την Bell Communications Research (γνωστή σήμερα ως iconectiv, θυγατρική της Ericsson). Αρχικά η LSA συχνά αναφερόταν Latent Semantic Indexing, καθώς κυρίως εφαρμοζόταν σε προβλήματα αναζήτησης αλλά στην συνέχεια το πεδίο εφαρμογών επεκτάθηκε και η μέθοδος άρχισε να εφαρμόζεται σε προβλήματα ομαδοποίησης, ανάλυσης σχέσεων, και άλλα.

## 1.3. Αντικείμενο διπλωματικής εργασίας – Σκοπός και στόχοι

Σκοπός αυτής της διπλωματικής εργασίας είναι η εξερεύνηση της εξόρυξης κειμένου ως προσέγγιση για συστηματική ανασκόπηση της βιβλιογραφίας και για την αποσαφήνιση του όρου τεκμηριωμένη ιατρική. Όπως συζητήθηκε στα προηγούμενα κεφάλαια, η εφαρμογή της τεκμηριωμένης ιατρικής απαιτεί την κατανάλωση μεγάλου όγκου επιστημονικών δημοσιεύσεων και δεδομένων, ένα σχεδόν αδύνατο σενάριο χωρίς αυτοματοποιημένη βοήθεια. Με βάση αυτούς τους προβληματισμούς σε αυτή την εργασία ερευνήθηκαν δύο προσεγγίσεις εξόρυξης σχετικής πληροφορίας από μεγάλο όγκο άγνωστων δεδομένων.

Η αξιολόγηση της εξόρυξης κειμένου έγινε μέσω της υλοποίησης δύο μεθόδων εξαγωγής πληροφορίας από κείμενα. Συγκεκριμένα, μετά από συλλογή μεγάλου όγκου από τίτλους και περιλήψεις επιστημονικών δημοσιεύσεων, χρησιμοποιήθηκαν τεχνικές εξόρυξης κειμένου για τον χαρακτηρισμό των κειμένων ως σχετικά με την εξόρυξη κειμένου, την τεκμηριωμένη ιατρική, ή και τα δύο. Η πρώτη μέθοδος βασίζεται στην αναζήτηση όρων μέσα σε απλοποιημένες αναπαραστάσεις των κειμένων, εκμεταλλευόμενη μόνο την ύπαρξη των όρων και τους γειτονικούς τους. Η δεύτερη μέθοδος, η οποία είναι γνωστή ως Latent Semantic Analysis (LSA) ή Latent Semantic Indexing (LSI), βασίζεται ανάλυση των σχέσεων των κειμένων και των όρων που τα συνθέτουν, υπό την υπόθεση πως κείμενα παρόμοιας σημασιολογίας θα περιέχουν κάποιο κοινό λεξιλόγιο. Η LSA μπορεί να ανακαλύψει σχέσεις ανάμεσα σε όρους ή/και κείμενα και έχει την δυνατότητα να ανακαλύπτει έγγραφα που είναι σημασιολογικά κοντά στους όρους μίας αναζήτησης, χωρίς αναγκαστικά να περιέχει τους όρους καθ' αυτούς.

Ως προς την επίτευξη του παραπάνω στόχου απαιτήθηκε η απόκτηση και η διαχείριση μεγάλου όγκου δεδομένων κειμένου. Η πραγματικότητα αυτή οδήγησε ένα μεγάλο κομμάτι της εργασίας στην ενασχόληση με τεχνικές διαχείρισης δεδομένων και προγραμματισμού δικτυακών εφαρμογών για να υποστηρίξουν τις τεχνικές εξόρυξης κειμένου. Ιδιαίτερη προσοχή δόθηκε ώστε τα προγράμματα που υλοποιήθηκαν να είναι σε θέση να διαχειριστούν τους μεγάλους όγκους δεδομένων σε έναν κοινό ηλεκτρονικό υπολογιστή.

## 1.4. Δομή εργασίας

Η διπλωματική αυτή εργασία είναι οργανωμένη όπως περιγράφεται σε αυτήν την ενότητα. Το Κεφάλαιο 2 περιέχει το θεωρητικό υπόβαθρο που απαιτείται για την υλοποίηση του σκοπού, όπως αυτός περιεγράφηκε στην προηγούμενη ενότητα. Συγκεκριμένα, καλύπτονται τα πρωτόκολλα που χρησιμοποιούνται για πρόσβαση σε αποθετήρια επιστημονικών (και μη) δημοσιεύσεων, τα οποία θα χρησιμοποιηθούν για την απόκτηση των απαιτούμενων δεδομένων. Όσον αφορά την οργάνωση τους, γίνεται αναφορά στο Dublin Core Metadata Set, μία συλλογή από πεδία που χρησιμοποιούνται στην βιβλιοθηκονομία για την περιγραφή πόρων. Επιπλέον, παρατίθενται πληροφορίες για τα συστήματα διαχείρισης βάσεων δεδομένων (database management systems, DBMS) και την PostgreSQL, που θα χρησιμοποιηθεί για την αποθήκευση, πρόσβαση αλλά και αναζήτηση στα δεδομένα. Το κεφάλαιο κλείνει με περιγραφή της μεθόδου Latent Semantic Analysis (LSA), μία μαθηματική τεχνική για την σημασιολογική αναζήτηση σε κείμενα.

Στο κεφάλαιο 3 περιγράφεται με λεπτομέρεια η υλοποίηση των προγραμμάτων και των αλγόριθμων που αναπτύχθηκαν κατά την διάρκεια συγγραφής αυτής της εργασίας. Συγκεκριμένα, περιγράφονται τα αποθετήρια επιστημονικών δημοσιεύσεων που χρησιμοποιήθηκαν καθώς και τα προγράμματα που αναπτύχθηκαν με σκοπό την απόκτηση δεδομένων. Επιπροσθέτως, αναλύονται οι τεχνικές που εφαρμόστηκαν για τον καθαρισμό και την οργάνωση των δεδομένων. Τέλος, το κεφάλαιο περιέχει πληροφορίες για το πως υλοποιήθηκαν και εφαρμόστηκαν οι τεχνικές εξόρυξης κειμένου.

Τα αποτελέσματα της εργασίας παρατίθενται στο κεφάλαιο 4 ενώ ο σχολιασμός τους γίνεται στο κεφάλαιο 5. Τέλος στο κεφάλαιο 6 παρατίθεται η βιβλιογραφία ενώ στο 7 ακολουθεί παράρτημα με  τον κώδικα που αναπτύχθηκε για όλη την εργασία.

# 2. Θεωρητικό υπόβαθρο

Σε αυτό το κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο των διάφορων τεχνολογιών και εννοιών που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

## 2.1. Πρωτόκολλα πρόσβασης σε αποθετήρια

Για την υλοποίηση απαιτήθηκε μεγάλος όγκος επιστημονικών δημοσιεύσεων και των σχετικών τους μεταδεδομένων (metadata) τα οποία αποκτήθηκαν μέσω προγραμματιστικής πρόσβασης σε αποθετήρια επιστημονικών δημοσιεύσεων.

Τα επιστημονικά αποθετήρια είναι βάσεις δεδομένων που διαχειρίζονται ακαδημαϊκό υλικό. Αυτά μπορεί να είναι ιδρυματικά, όπως από πανεπιστημιακές βιβλιοθήκες ή ερευνητικούς οργανισμούς, θεματικά τα οποία έχουν αν κάνουν με κάποιο επιστημονικό πεδίο ή γενικά αποθετήρια τα οποία περιέχουν δημοσιεύσεις αλλά και άλλο υλικό από πληθώρα επιστημονικών κλάδων. Μέσω αυτών μπορούμε να έχουμε πρόσβαση σε άρθρα και λοιπές δημοσιεύσεις ως απλοί χρήστες «χειροκίνητα» ή και προγραμματιστικά μέσω διεπαφών λογισμικού (APIs) που είναι διαθέσιμες για αυτόν τον σκοπό. Ενώ έχουν γίνει προσπάθειες τυποποίησης αυτών των APIs, στην πράξη όπως θα δούμε δεν είναι διαδεδομένες και συχνά απαιτείται ειδική μέριμνα για την πρόσβαση σε κάθε ξεχωριστό αποθετήριο.

Σε αυτή την ενότητα ερευνώνται οι μέθοδοι και τα πρωτόκολλα που χρησιμοποιούνται για την προγραμματιστική πρόσβαση σε αποθετήρια επιστημονικών δημοσιεύσεων.

### 2.1.1. REST APIs και HTTP requests

Το Πρωτόκολλο Μεταφοράς Υπερ-Κειμένου (Hyper Text Transfer Protocol – HTTP) είναι το πρωτόκολλο στο οποίο βασίζεται ο παγκόσμιος ιστός για την μεταφορά των πληροφοριών μεταξύ των διακομιστών και πελατών. Ακολουθεί μοντέλο πελάτη–διακομιστή (client-server) και είναι ένα πρωτόκολλο αίτησης–απάντησης (request-response), όπου οι πελάτες εκκινούν την επικοινωνία στέλνοντας μηνύματα-αιτήσεις και περιμένουν μέχρι ο διακομιστής να στείλει το μήνυμα απάντησης [12]. Είναι ένα πρωτόκολλο στο επίπεδο εφαρμογών (application layer) και συνήθως εκτελείται μέσω Πρωτοκόλλου Ελέγχου Μετάδοσης (Transmission Control Protocol – TCP) αλλά μπορεί να λειτουργήσει μέσω οποιουδήποτε αξιόπιστου πρωτοκόλλου μεταφοράς. Στο πλαίσιο αυτής της διπλωματικής εργασίας, το πρωτόκολλο HTTP εμφανίστηκε ως η βάση του REST API του αποθετηρίου Scopus και του OAI-PMH, το οποίο θα αναλυθεί περεταίρω στην συνέχεια.

**Αίτηση**

```
GET /api/ HTTP/1.1
Host: www.zenodo.org
```

**Απάντηση**

```
HTTP/1.1 200 OK
Server: nginx
Date: Sat, 07 Aug 2021 17:13:53 GMT
Content-Type: application/json
Content-Length: 362

{"links": ...}
```

*Κώδικας 1: Μία HTTP αίτηση GET στην ιστοσελίδα https://www.zenodo.org/api/ και απάντηση του εξυπηρετητή. Το πλήρης περιεχόμενο της απάντησης έχει αποκοπεί.*

Οι αιτήσεις αποτελούνται από τη μέθοδο (HTTP method/verb), τη διεύθυνση Ενιαίου Εντοπιστή Πόρων (Uniform Resource Locator – URL), την έκδοση του πρωτοκόλλου, προαιρετικά κάποιες κεφαλίδες αίτησης (request headers) ή/και κάποιο σώμα (περιεχόμενο) και μία κενή γραμμή.

Το πρωτόκολλο ορίζει ένα σύνολο από μεθόδους για τις αιτήσεις, οι οποίες ποια είναι η επιθυμητή ενέργεια να εκτελεστεί για τον πόρο. Ενώ δεν επιβάλλεται η χρήση των μεθόδων για την αρμόζουσα εφαρμογή τους, συχνά τα λογισμικά που υλοποιούν το πρωτόκολλο συμπεριφέρονται διαφορετικά ανάλογα την αίτηση. Στον Πίνακας 1 περιγράφονται τα διάφορα είδη αιτήσεων [13]. Οι αιτήσεις σχηματίζονται από τα παραπάνω γραμμένα με λατινικούς χαρακτήρες και κωδικοποιημένα σε ASCII[1]. Στον Κώδικας 1 φαίνεται μία αίτηση και η αντίστοιχη απάντηση σε πρωτόκολλο HTTP. Οι κεφαλίδες επιτρέπουν τη μεταφορά πληροφοριών που αφορούν την αίτηση ή τον πελάτη, όπως το είδος του περιεχομένου, τους αποδεκτούς τρόπους κωδικοποίησης ή την έκδοση του λογισμικού. Στον Πίνακας 2 παρατίθενται μερικές συχνές κεφαλίδες [15].

| HTTP Method | Περιγραφή |
|---|---|
| **GET** | Η μέθοδος GET αιτείται μία αναπαράσταση του συγκεκριμένου πόρου. Η αναπαράσταση αυτή μπορεί να είναι διαφορετική ανάλογα τον πελάτη (βλ. Κεφαλίδα Accept). Οι αιτήσεις GET δεν πρέπει να προκαλούν κάποια αλλαγή του πόρου. Ως επί το πλείστων, οι αιτήσεις GET δεν έχουν σώμα. Συνηθίζεται οι αιτήσεις GET να παίρνουν παραμέτρους στο τέλος του URL, μετά από ένα «?». |
| **HEAD** | Η μέθοδος HEAD ζητά μία πανομοιότυπη απάντηση με την GET αλλά χωρίς το σώμα της απάντησης. |
| **POST** | Η αίτηση POST χρησιμοποιείται για την υποβολή πληροφορίας σε έναν συγκεκριμένο πόρο. Αυτή η αίτηση συχνά προκαλεί αλλαγές στην μεριά του εξυπηρετητή. |
| **PUT** | Παρόμοια με την αίτηση POST, ζητά την αντικατάσταση του πόρου με το περιεχόμενο της αίτησης. |
| **DELETE** | Η αίτηση DELETE ζητά την διαγραφή ενός πόρου. |
| **PATCH** | Η αίτηση PATCH χρησιμοποιείται για την αλλαγή μέρους του πόρου. |

*Πίνακας 1: Μέθοδοι του πρωτοκόλλου HTTP. Στον πίνακα δεν περιλαμβάνονται οι μέθοδοι CONNECT, OPTIONS και TRACE που σπανίως χρησιμοποιούνται σε APIs*

| Κεφαλίδα | Request | Response | Περιγραφή |
|---|---|---|---|
| **Authorization** | ✔ | | Περιέχει διαπιστευτήρια για την αυθεντικοποίηση του πελάτη. |
| **Cache-Control** | | ✔ | Ρυθμίζει εάν η απάντηση μπορεί να αποθηκευτεί και να επαναχρησιμοποιηθεί χωρίς επικοινωνία με την εξυπηρετητή (caching). Χρησιμοποιείται για την διατήρηση των πόρων του δικτύου. |
| **Accept** | ✔ | | Ενημερώνει τον εξυπηρετητή τι είδους περιεχόμενο είναι αποδεκτό για την απάντηση. Εάν υποστηρίζεται και από τον εξυπηρετητή, θα επιστραφεί διαφορετική αναπαράσταση του πόρου ανάλογα με τις επιθυμίες του πελάτη. |

[1] Ιστορικά, επιτρεπόταν η χρήση και άλλων μορφών κωδικοποίησης αλλά αυτό δεν συστήνεται πια. [14]

| | | | |
|---|---|---|---|
| **Content-Length** | ✔ | ✔ | Το μήκος του σώματος σε bytes. |
| **Content-Type** | ✔ | ✔ | Το είδος του περιεχόμενου (media type). |
| **Location** | | ✔ | Η νέα τοποθεσία του πόρου, σε περίπτωση ανακατεύθυνσης. |
| **User-Agent** | ✔ | | Περιέχει πληροφορίες για το λογισμικό του πελάτη, πχ. την έκδοση του. |
| **Host** | ✔ | | Περιέχει το όνομα του εξυπηρετητή στον οποίον κατευθύνεται η αίτηση. |

*Πίνακας 2: Κεφαλίδες (headers) του πρωτοκόλλου HTTP και εάν αυτές αφορούν αιτήσεις, απαντήσεις ή και τα δύο.*

Οι απαντήσεις των διακομιστών στην πρώτη γραμμή περιέχουν την έκδοση του HTTP που ακολουθούν, έναν κωδικό κατάστασης (status code) και ένα μήνυμα κατάστασης (status message) που συμπληρώνει με μία σύντομη περιγραφή των κωδικό κατάστασης. Οι κωδικοί κατάστασης περιγράφουν την κατάσταση της αίτησης, δηλαδή κατά πόσο η απαιτούμενη διαδικασία ολοκληρώθηκε και αν όχι για ποιο λόγο απέτυχε η ολοκλήρωσή της. Ποιο συγκεκριμένα κωδικοί 1XX χρησιμοποιούνται σπάνια και είναι πληροφοριακοί, οι 2XX δηλώνουν επιτυχή εκτέλεση της αίτησης, οι 3XX δηλώνουν ανακατεύθυνση, οι 4XX σφάλμα πελάτη ενώ οι 5XX δηλώνουν σφάλμα διακομιστή. Στη συνέχεια η απάντηση μπορεί να περιέχει κεφαλίδες όπως αυτές των αιτήσεων και σώμα στο οποίο περιέχεται το περιεχόμενο που ζητήθηκε στην αίτηση.

Λόγω της γενικότητας και επεκτασιμότητάς του, το HTTP χρησιμοποιείται και για εφαρμογές πέρα από αυτές των ιστοσελίδων, όπως Κλήση Απομακρυσμένων Εφαρμογών (Remote Procedure Calls - RPC) και στο Απλό Πρωτόκολλο Προσπέλασης Αντικειμένων (Simple Object Access Protocol – SOAP) ενώ διάφορες Διεπαφές Προγραμματισμού Εφαρμογών (Application Programming Interfaces – APIs) είναι βασισμένες σε αυτό. Τα APIs είναι σύνολα κανόνων και πρωτοκόλλων που καθορίζουν την αλληλεπίδραση εφαρμογών και υπηρεσιών επιτρέποντας τους να επικοινωνούν χωρίς να γνωρίζουν το ένα την υλοποίηση του άλλου.

Οι διεπαφές τύπου REST (Representational state transfer) χρησιμοποιούν τις μεθόδους του HTTP πρωτοκόλλου με συγκριμένο τρόπο ώστε να οργανώσουν την πρόσβαση σε πόρους (resources). Με άλλα λόγια, το REST δεν είναι ένα πρωτόκολλο αλλά ένα σύνολο από αρχιτεκτονικές αποφάσεις ώστε να είναι απλούστερος ο σχεδιασμός και η χρήση τέτοιων διεπαφών. Κάθε πρόγραμμα ή υπηρεσία υλοποιεί διαφορετικά την διεπαφή της και όταν ακολουθεί τους κανόνες του REST αποκαλείται «RESTful», «REST API» ή/και «REST Service» [15]. Σύμφωνα με την αρχιτεκτονική REST, οι εξυπηρετητές προσφέρουν πρόσβαση σε πόρους μετά από συγκεκριμένες αιτήσεις. Οι πόροι μπορεί να είναι οποιοδήποτε κομμάτι πληροφορίας, όπως ένα έγγραφο, μία εικόνα ή κατάσταση μίας συσκευής. Με αυτή την αφαίρεση, οι πελάτες έχουν πρόσβαση στους πόρους χωρίς να χρειάζεται να γνωρίζουν για τις λεπτομέρειες της υλοποίησης της υπηρεσίας.

### 2.1.2. OAI-PMH

Το Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) είναι ένα πρωτόκολλο για την συμβατότητα των αποθετηρίων [16]. Το πρωτόκολλο περιγράφει έναν απλό μηχανισμό για την πρόσβαση σε μεταδεδομένα που αφορούν το περιεχόμενο που φιλοξενείται σε ένα αποθετήριο. Το OAI-PMH από τον σχεδιασμό του δεν προβλέπει πολλές λειτουργίες, όπως πολύπλοκη αναζήτηση ή πρόσβαση στο περιεχόμενο των δημοσιεύσεων [17]. Αυτή η απόφαση αυτή έγινε ευελπιστώντας ότι το απλό αυτό πρωτόκολλο θα είναι εύκολο στην υλοποίηση και πως αυτό θα οδηγούσε στην ευρεία χρήση του. Η τελευταία έκδοση του πρωτοκόλλου είναι η 2.0, η οποία

δημοσιεύθηκε τον Μάιο του 2002. Σε αυτή την ενότητα περιγράφεται ένα υποσύνολο του πρωτοκόλλου που είναι αρκετό για την εξαγωγή πληροφοριών από αποθετήρια και όχι όλες οι λειτουργίες.

Το πρωτόκολλο περιγράφει την επικοινωνία ανάμεσα στα Repositories (αποθετήρια) και στους Harvesters, προγράμματα που συλλέγουν μεταδεδομένα από αποθετήρια. Τα αποθετήρια πρέπει να είναι προσβάσιμα μέσω δικτύου και να είναι σε θέση να επεξεργαστούν τους 6 τύπους αιτημάτων που περιγράφονται από το πρωτόκολλο. Σύμφωνα με το OAI-PMH, τα αποθετήρια φιλοξενούν πόρους (resources) στα οποία αντιστοιχούν αντικείμενα (items). Κάθε οντότητα που φιλοξενείται από το αποθετήριο είναι ένας πόρος, χωρίς το πρωτόκολλο να κάνει λόγο για την φύση του (φυσικό ή ψηφιακό, είδος δημοσίευσης). Τα αντικείμενα είναι το σύνολο των μεταδεδομένων που αντιστοιχούν σε κάθε πόρο. Τέλος, σύμφωνα με το πρωτόκολλο οι πληροφορίες των αντικειμένων κωδικοποιούνται σε ένα record, δηλαδή ένα XML-encoded byte stream, πριν αποσταλθούν στον harvester.

Κάθε αντικείμενο πρέπει να ταυτοποιείται μοναδικά μέσω κάποιου μοναδικού αναγνωριστικού (Unique Identifier). Καθώς σε ένα αντικείμενο είναι πιθανών να αντιστοιχούν περισσότερα από ένα records μεταδεδομένων, το Unique Identifier είναι ένα τρόπος να συσχετίζονται αυτά τα δεδομένα μεταξύ τους. Οι Unique Identifiers πρέπει να είναι Unique Resource Identifiers (URIs) όπως URL ή DOI.

Τα αποθετήρια μπορούν επίσης να κατηγοριοποιούν τα αντικείμενα σε μία ή παραπάνω κατηγορίες, οι οποίες ονομάζονται σύνολα (sets). Τα σύνολα μπορούν να είναι οργανωμένα είτε σαν απλή λίστα είτε ιεραρχικά. Κάθε set πρέπει να έχει το `setSpec`, δηλαδή το «κωδικό-όνομα» του. Για παράδειγμα, στην περίπτωση ιεραρχικής οργάνωσης, θα μπορούσε να υπάρχει το set `subjects` που περιέχει τα sets `subjects:biology` και `subjects:physics`. Η χρήση του «:» δεν επιτρέπεται σε όνομα καθώς είναι δεσμευμένη για την ένδειξη της ιεραρχίας. Τα sets πρέπει επίσης να έχουν ένα επίσημο όνομα (`setName`) και, προαιρετικά, κάποια περιγραφή σε μορφή XML (`setDescription`). Η υποστήριξη των sets δεν απαιτείται από το πρωτόκολλο και τα αποθετήρια δικαιούνται να απαντούν με σφάλμα `noSetHierarchy` εάν δεν τα υποστηρίζουν.

Τα records, όπως προαναφέρθηκε, είναι metadata εκφρασμένα σε ένα συγκεκριμένο format. Μετά από αίτηση ενός harvester, τα records επιστρέφονται επισημασμένα σε XML και πρέπει να περιέχουν τις εξής δομές:

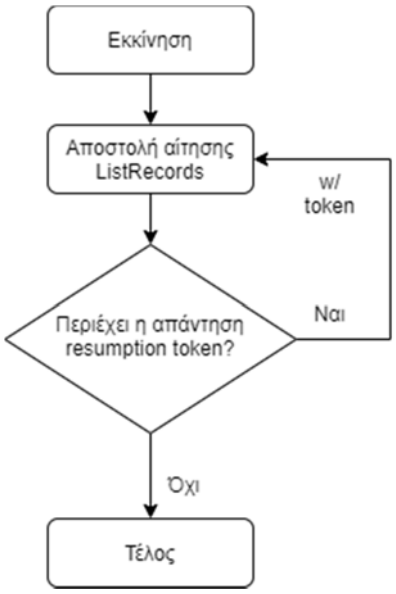
- Header – Περιέχει το Unique Identifier του αντικειμένου που αντιστοιχεί στην εγγραφή (record), μία ημερομηνία (συνήθως δημιουργίας) και τα sets στα οποία ανήκει αυτό το αντικείμενο.
- Metadata – Περιέχει τα μεταδεδομένα υπό συγκεκριμένη κωδικοποίηση. Πρέπει να περιέχει μόνο ένα XML element, το οποίο να διαθέτει τα απαραίτητα namespace declarations. Για παράδειγμα, συχνά χρησιμοποιείται το `oai_dc:dc` tag που παραπέμπει στο Dublin Core.
- About – Περιέχει πληροφορίες για το είδος πρόσβασης που επιτρέπεται στο resource (πχ. open access) και την προέλευση των μεταδεδομένων, για παράδειγμα σε περίπτωση που το resource προέρχεται από άλλο αποθετήριο.

Η επικοινωνία με τα αποθετήρια πραγματοποιείται χρησιμοποιώντας HTTP requests. Τα αιτήματα γίνονται όλα στο ίδιο URL και πρέπει να υποστηρίζεται η αποστολή αιτημάτων GET και POST. Μεταξύ των δύο HTTP verbs δεν υπάρχει διαφορά όσων αφορά το OAI-PMH. Τα αιτήματα έχουν παραμέτρους οι οποίες πρέπει να αποστέλλονται είτε ως Query Parameters με GET request είτε ως URL-encoded περιεχόμενο σε περίπτωση POST request. Η πρώτη παράμετρος ονομάζεται `verb` και είναι το είδος της αίτησης που πραγματοποιείται. Ανάλογα το είδος, μπορεί να απαιτούνται και άλλες παράμετροι. Για την κατάσταση των αιτημάτων χρησιμοποιούνται τα HTTP status codes, όπως 200 για επιτυχές αίτημα.

Το πρωτόκολλο περιγράφει 6 είδη αιτήσεων:

- `GetRecord` – Ανάκτηση ενός συγκεκριμένου record.
- `Identify` – Πληροφορίες για το αποθετήριο

- `ListIdentifiers` – Ανάκτηση αντικειμένων, επιστρέφοντας μόνο το Header.
- `ListMetadataFormats` – Πληροφορίες για τα είδη κωδικοποίησης που υποστηρίζονται από το αποθετήριο
- `ListRecords` – Ανάκτηση αντικειμένων, επιστρέφοντας και το Header και τα Metadata.
- `ListSets` – Ανάκτηση λίστας με τα διαθέσιμα Sets




*Σχήμα 1: Αλγόριθμος λήψης οντοτήτων με χρήση του resumption token στο πρωτόκολλο OAI-PMH.*

Για την μαζική ανάκτηση πληροφορίας, όπως απαιτείται για την ολοκλήρωση αυτής της εργασίας, η πιο σημαντική αίτηση είναι η `ListRecords`. Καθώς η αίτηση `ListRecords` (και οι `ListSets` , `ListIdentifiers`) επιστρέφει εξ' ορισμού μία λίστα από διακριτές οντότητες, είναι πιθανών το μέγεθος της πληροφορίας να είναι πολύ μεγάλο για να μπορεί να μεταδοθεί αξιόπιστα σε μόνο μία απάντηση. Για πρακτικούς λοιπόν λόγους, τα αποθετήρια επιτρέπεται από το πρωτόκολλο να επιστρέψουν ένα κομμάτι της λίστας μαζί με έναν κωδικό συνέχισης (resumption token). Το πρόγραμμα harvester μπορεί να επαναλάβει το request, περιλαμβάνοντας τώρα το resumption token, ώστε να λάβει το επόμενο κομμάτι της λίστας. Μετά από αρκετές επαναλήψεις, είναι δυνατόν να ληφθεί το σύνολο των οντοτήτων. Ο αλγόριθμος φαίνεται στο Σχήμα 1. Στην περίπτωση που δεν είναι επιθυμητή η λήψη του συνόλου των οντοτήτων, ο harvester μπορεί να σταματήσει την επανάληψη των αιτημάτων όταν έχει αρκετά δεδομένα. Το resumption token ενδέχεται να συνοδεύεται από ημερομηνία λήξης, πέρας της οποίας δεν θα είναι αποδεκτό από το αποθετήριο.

Η αίτηση `ListRecords` μπορεί να δεχτεί ως παραμέτρους:

- `from/until`: Προαιρετικό πάνω/κάτω χρονικό όριο για επιλεκτική ανάκτηση οντοτήτων. Πρέπει να είναι σε μορφή ISO8601 και σε ζώνη ώρας UTC.
- `set`: επιλεκτική ανάκτηση οντοτήτων που ανήκουν σε κάποιο σύνολο (set).
- `resumptionToken`: Χρησιμοποιείται για την συνέχεια λήψης μίας λίστας οντοτήτων.
- `metadataPrefix`: Προαιρετική επιλογή της κωδικοποίησης των μεταδεδομένων. Οι υποστηριζόμενες μορφές μπορούν να ληφθούν με μία αίτηση `ListMetadataFormats`.

Ένα παράδειγμα αίτησης στο φανταστικό αποθετήριο `https://archive.example.com` με σκοπό την λήψη πόρων με ημερομηνία έκδοσης το έτος 2012 και που ανήκουν στο σύνολο «bme» θα ήταν:

```
https://archive.example.com/
                    ?verb=ListRecords&from=2012-01-01&until=2012-12-31&set=bme
```

Οι απαντήσεις του αποθετηρίου σε όλα τα αιτήματα περιγράφονται από το XML σχήμα που είναι διαθέσιμο στην ιστοσελίδα του πρωτοκόλλου[2]. Κοινά στοιχεία για όλα τα είδη αιτήσεων είναι η ημερομηνία της απάντησης (`responseDate`), μία επανάληψη των παραμέτρων της αίτησης (`request`)

[2] http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
         http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2021-06-02T12:20:30Z</responseDate>
  <request verb="ListRecords" from="2012-01-01" until="2012-12-31"
          set="bme">
          http://an.oa.org/OAI-script</request>
  <ListRecords>
    <record>
      <header>
        <identifier>oai:example.com:bme/1</identifier>
        <datestamp>2012-03-24</datestamp>
        <setSpec>bme</setSpec>
        <setSpec>engineering</setSpec>
      </header>
      <metadata>
        <!-- ... -->
      </metadata>
      <about>
        <oai_dc:dc
            xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oa
i_dc/
            http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
          <dc:publisher>Example Publisher</dc:publisher>
        </oai_dc:dc>
      </about>
    </record>
  </ListRecords>
</OAI-PMH>
```

*Κώδικας 2: Απάντηση αποθετηρίου σε αίτηση ListRecords σύμφωνα με το πρωτόκολλο OAI-PMH.*

και τέλος, την πληροφορία που αντιστοιχεί στην αίτηση. Συγκεκριμένα για αίτηση `ListRecords`, εμφανίζεται το XML node `ListRecords` που περιέχει ένα ή περισσότερα record nodes, όπως αυτά αναλύθηκαν παραπάνω (header, metadata, about). Η απάντηση για την αίτηση-παράδειγμα φαίνεται στον Κώδικας 2.

## 2.2. Οργάνωση δεδομένων

Η οργάνωση των αρχείων που αποκτήθηκαν και των σχετικών τους μεταδεδομένων έγινε με τη βοήθεια μιας σχεσιακής βάσης δεδομένων. Παρακάτω περιγράφεται ένας τρόπος σήμανσης (markup) δεδομένων ψηφιακών αρχείων που συναντήθηκε και στη συνέχεια περιγράφονται τα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων που χρησιμοποιήθηκαν.

### 2.2.1. Dublin Core Metadata Set

Ένας τρόπος σήμανσης των μεταδεδομένων ενός αντικειμένου (πχ. επιστημονική δημοσίευση) είναι το «Dublin Core Metadata Set», ή πιο συχνά «Dublin Core». Το Dublin αναφέρεται στο Δουβλίνο του Οχάιο, όπου το 1995 σε ένα συνέδριο βιβλιοθηκονομίας (NCSA Metadata Workshop[18]) ξεκίνησε η εργασία πάνω σε ένα σύνολο πεδίων που θα ήταν αρκετό για να περιγράψει διαφόρων μορφών αντικείμενα με έναν ομοιόμορφο τρόπο. Από τότε, το σύνολο αυτό έχει τυποποιηθεί (ISO 15836:2003 [19]) και το Dublin Core έχει βρει εφαρμογή σε διάφορα πεδία που σχετίζονται με την αρχειοθέτηση

πληροφοριών, όπως και επιστημονικών δημοσιεύσεων. Ενώ τα αποθετήρια συχνά διαχειρίζονται τις δημοσιεύσεις με μεγαλύτερη λεπτομέρεια από ότι προβλέπει το Dublin Core, τα βασικότερα πεδία είναι συνήθως διαθέσιμα και με αυτή την μορφοποίηση, γεγονός που διευκολύνει την αυτόματη επεξεργασία τους.

Το Dublin Core δεν προβλέπει την χρήση συγκεκριμένης κωδικοποίησης (encoding), αντί αυτού είναι αποδεκτή η χρήση οποιουδήποτε format είναι αναγνώσιμο και από ανθρώπους και μηχανές. Για παράδειγμα, στα αποθετήρια που χρησιμοποιήθηκαν για την συλλογή δεδομένων, το Dublin Core εμφανίστηκε και σε μορφή XML (eXtensible Markup Language) και σε μορφή JSON (JavaScript Object Notation).

Το βασικό επίπεδο του Dublin Core προβλέπει 15 κύρια πεδία για την περιγραφή των αντικειμένων [20]. Τα πεδία και μία σύντομη περιγραφή για το κάθε ένα υπάρχουν στον Πίνακας 3. Η περιγραφή του κάθε πεδίου δεν είναι περιοριστική όσον αφορά το περιεχόμενο με αποτέλεσμα διαφορετικές πηγές να αναπαριστούν την ίδια πληροφορία με άλλο τρόπο. Αντιμέτωποι με αυτό το πρόβλημα θα βρεθούμε στο επόμενο κεφάλαιο.

| **Πεδίο** | **Σύντομη Περιγραφή** |
|---|---|
| **Contributor/Συνεισφέρων** | Οντότητα που πραγματοποίησε συνεισφορές στον πόρο |
| **Coverage/Κάλυψη** | Το χωρικό ή χρονικό θέμα του πόρου, η χωρική εφαρμογή του, ή η δικαιοδοσία υπό την οποία ο πόρος είναι σχετικός |
| **Creator/Δημιουργός** | Οντότητα υπεύθυνη για τη δημιουργία του πόρου |
| **Date/Ημερομηνία** | Μια χρονική στιγμή ή χρονική περίοδος σχετική με τον πόρο |
| **Description/Περιγραφή** | Περιγραφή του πόρου |
| **Format/Μορφή** | Ο τύπος αρχείου, το φυσικό μέσο ή οι διαστάσεις του πόρου |
| **Identifier/Αναγνωριστικό** | Μια μοναδική αναφορά στον πόρο μέσα σε συγκεκριμένο πλαίσιο |
| **Language/Γλώσσα** | Η γλώσσα του πόρου |
| **Publisher/Εκδότης** | Οντότητα υπεύθυνη για την έκδοση του πόρου |
| **Relation/Συγγένεια** | Συγγενής οντότητα |
| **Rights/Δικαιώματα** | Πληροφορίες σχετικά με τα δικαιώματα που διατηρούνται πάνω στον πόρο. |
| **Source/Πηγή** | Πηγή του περιγραφόμενου πόρου |
| **Subject/Αντικείμενο** | Το αντικείμενο του πόρου |
| **Title/Τίτλος** | Ο τίτλος του πόρου |
| **Type/Τύπος** | Η φύση ή το είδος του πόρου. |

*Πίνακας 3: Σύνολο πεδίων του Dublin Core Metadata Element Set.*

### 2.2.2. Relational Database Management Systems

Τα συστήματα διαχείρισης βάσης δεδομένων (database management system - DBMS) είναι συστήματα για την διαχείριση πληροφορίας και την πρόσβαση σε αυτήν [21]. Περιλαμβάνει τη βάση δεδομένων, δηλαδή μια συλλογή από αλληλένδετα δεδομένα και προγράμματα για την εύκολη και γρήγορη αποθήκευση και ανάκτηση τους. Τα DBMS πρέπει να εκτελούν αυτές τις διαδικασίες με αποτελεσματικότητα και να εξασφαλίζουν την ασφάλεια των δεδομένων. Ανάλογα τις ανάγκες, συχνά τα συστήματα αυτά πρέπει να είναι σε θέση να προσφέρουν πρόσβαση σε πολλαπλούς χρήστες/προγράμματα, διατηρώντας πάντα την αρτιότητα των δεδομένων. Για την διαχείριση των δεδομένων της εργασίας χρησιμοποιήθηκε το σύστημα PostgreSQL.

Τα DBMS βασίζονται ιδιαίτερα στην έννοια της αφαίρεσης (abstraction) για να είναι σε θέση να καλύψουν τις ανάγκες που προαναφέρθηκαν. Στην βάση της στοίβας υπάρχει το Φυσικό επίπεδο, όπου γίνεται η αποθήκευση της πληροφορίας σε μη-πτητική μνήμη, χρησιμοποιώντας κατάλληλες τεχνικές κωδικοποίησης. Σε αυτό το επίπεδο εργάζονται κατά κύριο λόγο οι δημιουργοί των συστημάτων. Το αμέσως επόμενο επίπεδο είναι το Λογικό, το οποίο περιγράφει τι είδους πληροφορίας αποθηκεύεται στο σύστημα και πως τα δεδομένα αυτά συσχετίζονται μεταξύ τους. Ενώ η υλοποίηση των δομών που περιγράφονται στο λογικό επίπεδο πιθανών να απαιτούν πιο πολύπλοκες δομές στο φυσικό επίπεδο, ο χρήστης της βάσης δεδομένων δεν είναι εκτεθειμένος σε αυτή την πολυπλοκότητα. Τέλος, είναι λογικό ότι ο κάθε χρήστης (ή εφαρμογή) του συστήματος να χρειάζεται πρόσβαση σε ένα τμήμα των δεδομένων. Στο τελευταίο επίπεδο της λίστας, το επίπεδο Προβολής, προσφέρει πρόσβαση ακριβώς στο τμήμα αυτό. Το σύστημα μπορεί να προσφέρει πολλές διαφορετικές προβολές για την ίδια βάση δεδομένων ή ακόμα και για τα ίδια δεδομένα. Τα DBMS έχουν την δυνατότητα να διαχειρίζονται βάσεις δεδομένων πολλές φορές μεγαλύτερες από την κύρια μνήμη του συστήματος και έχουν υπό-συστήματα (όπως οι Buffer Managers [22]) με σκοπό την βέλτιστη διαχείριση της διαθέσιμης κύριας μνήμης για καλύτερες επιδόσεις.

Τα σχεσιακά συστήματα διαχείρισης βάσης δεδομένων (relational database management systems - RDBMS) βασίζονται στο σχεσιακό μοντέλο για την αναπαράσταση των δεδομένων [23]. Σύμφωνα με το σχεσιακό μοντέλο, τα δεδομένα οργανώνονται σε πίνακες, όπου κάθε δείγμα δεδομένων αντιστοιχεί σε μία γραμμή (πλειάδα). Ο κάθε πίνακας περιέχει εγγραφές ενός συγκεκριμένου τύπου, δηλαδή κάθε εγγραφή έχει συγκεκριμένο αριθμό πεδίων και κάθε πεδίο δέχεται μόνο έναν τύπο δεδομένων (για παράδειγμα, μόνο αριθμούς). Αυτό το μοντέλο αναπαράστασης είναι το πιο διαδεδομένο και η πλειοψηφία των συστημάτων βάσεων δεδομένων βασίζονται σε αυτό. Η αλληλεπίδραση των χρηστών με το σύστημα γίνεται χρησιμοποιώντας την Structured Query Language (SQL). Μέσω της SQL, οι χρήστες ορίζουν και το σχήμα της βάσης αλλά μπορούν να κάνουν και queries που δημιουργούν κατά παραγγελία προβολές ή τροποποιούν τα δεδομένα.

Καθώς η διαχείριση δεδομένων είναι εξαιρετικά σημαντική για σχεδόν κάθε είδους επιχείρημα, έχει γίνει εκτεταμένη δουλειά στον κλάδο και πλέον υπάρχουν διαθέσιμα πολλά συστήματα για να καλύψουν τις ποίκιλες ανάγκες που προκύπτουν. Διάσημα συστήματα ανοιχτού κώδικα είναι το MariaDB[24] και το PostgreSQL [25], το οποίο χρησιμοποιήθηκε και για την διαχείριση των δεδομένων αυτής της εργασίας. Η PostgreSQL είναι ένα διαδεδομένο RDBMS με περισσότερα από 30 χρόνια ανάπτυξης. Υποστηρίζει τα κοινά λειτουργικά συστήματα (Windows, Linux, BSD, macOS και Solaris) και έχει εκτεταμένες ικανότητες αναζήτησης κειμένου. Η αλληλεπίδραση με την βάση δεδομένων γίνεται χρησιμοποιώντας SQL αλλά υπάρχει η δυνατότητα για εκτέλεση άλλων γλωσσών προγραμματισμού (όπως R) για την επεξεργασία των αποθηκευμένων δεδομένων.

Η γλώσσα SQL δεν είναι διαδικαστική (non-procedural) σε αντίθεση με τις κοινές γλώσσες προγραμματισμού όπως η R [21]. Αυτό μεταφράζεται πως σε αντίθεση με τις διαδικαστικές γλώσσες προγραμματισμού που εκτελούν εντολές από πάνω προς τα κάτω, σε μία εντολή SQL περιγράφεται κατευθείαν το επιθυμητό αποτέλεσμα. Στην συνέχεια, είναι δουλειά του συστήματος να μεταφράσει αυτή την εντολή σε διακριτά βήματα προς εκτέλεση ώστε να παραχθεί το ζητούμενο (συγκεκριμένα, το υποσύστημα που αναλαμβάνει αυτή την δουλειά λέγετε query planner, καθώς «σχεδιάζει» το πως

θα εκτελεστεί η εντολή). Εξαίρεση σε αυτό αποτελούν τα User-Defined Functions, τα οποία θα αναλυθούν παρακάτω. Η SQL προσφέρει εντολές για τον ορισμό και την τροποποίηση του σχήματος της βάσης δεδομένων (δηλαδή, είναι μία Data-Definition Language, DDL) αλλά και για την εισαγωγή, εξαγωγή και τροποποίηση των δεδομένων (Data-Manipulation Language, DML).

Οι περισσότερες εντολές SQL δέχονται ως είσοδο έναν ή παραπάνω πίνακες και πάντα επιστρέφουν έναν πίνακα. Ο πίνακας που επιστρέφει δεν αντιστοιχεί αναγκαστικά σε κάποιον πίνακα αποθηκευμένο στην βάση, αλλά είναι μία προβολή των δεδομένων ανάλογα με τις επιθυμίες του χρήστη. Η SQL προσφέρει συναρτήσεις για κοινούς υπολογισμούς, όπως στατιστικά μεγέθη, αλλά υπάρχει και η δυνατότητα συγγραφής ειδικών συναρτήσεων που ονομάζονται «User-Defined Functions» (UDFs). Οι συναρτήσεις αυτές, όπως αναφέρθηκε νωρίτερα, μπορούν να υλοποιηθούν και σε άλλες γλώσσες εκτός της SQL, όπως R, Python και C. Παραδοσιακά, για την επεξεργασία των δεδομένων, κάποιο πρόγραμμα επικοινωνεί με την βάση δεδομένων και μέσω κάποιου SQL script, μεταφέρει ένα κομμάτι των δεδομένων στην μνήμη του για επεξεργασία. Εάν ο όγκος των δεδομένων είναι μεγάλος, και ιδιαίτερα αν η βάση δεδομένων βρίσκεται σε διαφορετικό υπολογιστή από το πρόγραμμα, αυτή η μεταφορά μπορεί να είναι ιδιαίτερα απαιτητική σε πόρους. Υλοποιώντας ένα κομμάτι του αλγόριθμου σε UDF, αντί να μεταφερθούν τα δεδομένα στο πρόγραμμα, μεταφέρεται το πρόγραμμα στα δεδομένα, εξοικονομώντας χρόνο και πόρους δικτύου. Επιπλέον, ένας αλγόριθμος υλοποιημένος σε UDF εκμεταλλεύεται άμεσα τις δυνατότητες των DBMS να διαχειρίζονται μεγάλο όγκο δεδομένων, χωρίς ιδιαίτερη μέριμνα από τον προγραμματιστή.

## 2.3. Τεχνικές αναζήτησης στα δεδομένα

Ως προς τον σκοπό της εργασίας, χρησιμοποιήθηκαν δύο τεχνικές αναζήτησης σε κείμενο. Η πρώτη βασίζεται στις δυνατότητες αναζήτησης της Postgres (full-text searching) ενώ η δεύτερη βασίζεται στην τεχνική Latent Semantic Analysis (LSA). Σε αυτή την ενότητα θα αναλυθούν σε βάθος οι δύο τεχνικές.

### 2.3.1. Αναζήτηση κειμένου στην Postgres

Στο πλαίσιο της Postgres, αναζήτηση κειμένου (full text searching) ορίζεται η δυνατότητα να ταυτοποιούνται έγγραφα σε φυσική γλώσσα σε σχέση με ένα ερώτημα (query) [26]. Η πιο βασική μορφή αναζήτησης είναι η εμφάνιση όλων των εγγράφων που περιέχουν κάποιους όρους αναζήτησης αλλά αναλόγως την εφαρμογή, μπορούν να εφαρμοστούν περισσότεροι κανόνες ή να επιθυμούμε την ταξινόμηση των αποτελεσμάτων σε σχέση με τους όρους της αναζήτησης.

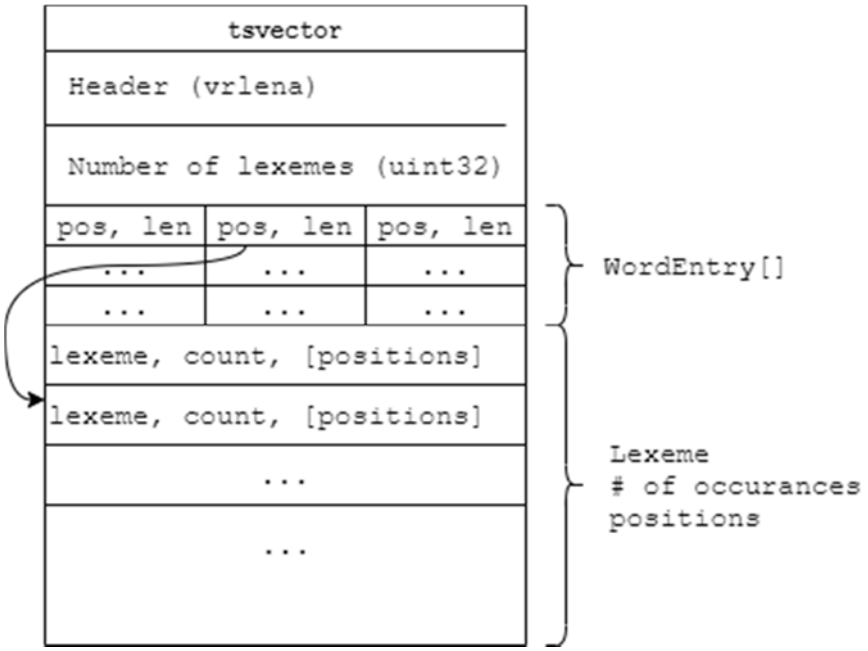
Τα συστήματα βάσεων δεδομένων υποστηρίζουν μία μορφή βασικής αναζήτησης μέσω των τελεστών αναζήτησης κειμένου. Στην SQL ορίζονται οι τελεστές `~`, `~*`, `LIKE` και `ILIKE` για αλφαριθμητικά δεδομένα (τύπου `CHAR`, `VARCHAR`, `TEXT`, ...) αλλά δεν αρκούν για να καλύψουν τις ανάγκες παρά για τις πιο απλές εφαρμογές. Χρησιμοποιώντας τον τελεστή `LIKE`, ο οποίος ελέγχει εάν μία καταχώρηση ταιριάζει σε ένα απλό μοτίβο[3], με query την λέξη "construction", είναι εφικτό να αναζητηθούν όλες οι καταχωρήσεις που περιέχουν την λέξη ως έχει αλλά όχι αυτές που περιέχουν "constructing" ή "constructed". Επιπροσθέτως, μία τέτοια προσέγγιση δεν ταξινομεί τα αποτελέσματα με κανέναν τρόπο, γεγονός που την καθιστά ανωφελή για αναζήτηση σε μεγάλες βάσεις δεδομένων. Τέλος, οι όροι ή το μοτίβο της αναζήτησης πρέπει να ελεγχθεί με κάθε καταχώρηση ξεχωριστά, απαγορεύοντας την χρήση ευρετηρίων (indices).

Στην Postgres, η διαδικασία "Full Text Indexing" προεργάζεται τις καταχωρήσεις και δημιουργεί ένα ευρετήριο για ταχεία αναζήτηση. Το πρώτο βήμα της προεργασίας είναι η μετατροπή των κειμένων σε λεξικές μονάδες (tokens). Σε αυτό το βήμα ταυτοποιούνται μονάδες οι οποίες πρέπει να επεξεργαστούν με κάποιον ιδιαίτερο τρόπο, όπως για παράδειγμα νούμερα, σύνθετες λέξεις ή

[3] Υποστηρίζεται το σύμβολο `%` στην θέση κανενός, ενός ή παραπάνω άγνωστων χαρακτήρων και το σύμβολο `_` στην θέση ενός άγνωστου χαρακτήρα. Για παράδειγμα, το `LIKE "Hello%"` θα αναζητούσε εγγραφές που ξεκινούν με "Hello".

διευθύνσεις email. Το δεύτερο βήμα μετατρέπει τις λεξικές μονάδες σε λεξήματα (lexemes). Τα λεξήματα είναι κανονικοποιημένες μορφές των λεκτικών μονάδων ώστε διάφορες μορφές της ίδιας λέξεις να γίνονται όμοιες. Η κανονικοποίηση περιλαμβάνει αντικατάσταση των κεφαλαίων χαρακτήρων με πεζούς και την λημματοποίηση (lemmatization), δηλαδή την αφαίρεση καταλήξεων και προθεμάτων από τις λέξεις ώστε να παραμείνει ο σκελετός τους. Για παράδειγμα, οι λέξεις "proccessed" και "processing" κανονικοποιούνται σαν "process". Λέξεις οι οποίες συναντιόνται πολύ συχνά στα κείμενα για να είναι χρήσιμες για τους σκοπούς μίας αναζήτησης (γνωστές και ως κενές λέξεις ή stop words), όπως για παράδειγμα άρθρα, αφαιρούνται. Ολοκληρώνοντας την διαδικασία, τα κείμενα πια αναπαρίστανται ως ταξινομημένες λίστες από λεξήματα. Μαζί με κάθε λέξημα αποθηκεύονται και πληροφορίες για την θέση και την συχνότητα του στο κείμενο.

Η Postgres προσφέρει τον τύπο δεδομένων `tsvector` για την αποθήκευση των επεξεργασμένων κειμένων. Παράλληλα, παρέχεται ο τύπος `tsquery` για την αναπαράσταση όρων και φράσεων αναζήτησης (queries), καθώς και διάφοροι τελεστές και ρουτίνες για την χρήση αυτών των λειτουργιών. Ένα `tsvector` αποτελείται από μία επικεφαλίδα (header), μία λίστα με `WordEntry` και τα δεδομένα των λεξημάτων [27]. Τα `WordEntry` είναι ένα ζεύγος αριθμών που αντιπροσωπεύουν τον αριθμό των bytes από το τέλος της λίστας των word entries μέχρι την αρχή του λεξήματος, καθώς και το μέγεθος του. Τα λεξήματα αποτελούνται από τους χαρακτήρες της λέξης, τον αριθμό των εμφανίσεων της στο αρχικό κείμενο, καθώς και μία λίστα με την θέση της κάθε εμφάνισης, ως αριθμός λέξεων από την αρχή του κειμένου. Στο Σχήμα 2 φαίνεται ένα `tsvector` όπως αυτό αποθηκεύεται στην κεντρική μνήμη του υπολογιστή. Το διπλό «indexing» (δηλαδή, από `tsvector` σε `wordentry` και μετέπειτα σε lexeme data) είναι απαραίτητο διότι δεν είναι γνωστό εξ' αρχής το μέγεθος των λεξημάτων, ούτε ο αριθμός των εμφανίσεων και το μέγεθος του αντίστοιχου array. Οι δύο αριθμοί του `WordEntry` αποθηκεύονται σαν ένα `uint32`, καθιστώντας το μέγιστο μέγεθος ενός λεξήματος στα 2Kb καθώς αποθηκεύεται σε 11 bits και το μέγεθος του αρχικού κειμένου καθώς αποθηκεύεται σε 20 bits.




*Σχήμα 2: Δομή ενός `tsvector` στην κεντρική μνήμη του υπολογιστή.*

Για την επιτάχυνση των queries και την ταχύτερη ανάκτηση των δεδομένων, η Postgres υποστηρίζει δύο τύπους ευρετηρίων σε στήλες `tsvector`. Ο πρώτος τύπος είναι τα Generalized Search Tree (GiST), ενώ ο δεύτερος είναι τα Generalized Inverted Index (GIN) [28]. Και οι δύο τύποι είναι επεκτάσιμοι ως προς το πως θα εφαρμοστούν σε κάθε τύπο δεδομένου, δίνοντας την δυνατότητα στους προγραμματιστές και στους χρήστες της Postgres να τα προσαρμόσουν για βέλτιστη απόδοση. Τα ευρετήρια GiST μπορούν να ανανεωθούν ταχύτερα σε περίπτωση προσθήκης καταχωρήσεων στον

πίνακα, αλλά είναι πιθανό να επιστρέψουν λάθος αποτελέσματα (false matches). Τα λάθη αυτά αναγκάζουν το σύστημα να κάνει τυχαίες αναγνώσεις από τον πίνακα (random access), κάτι ιδιαίτερα χρονοβόρο. Τα ευρετήρια GIN δεν επιστρέφουν λάθος αποτελέσματα αλλά η απόδοση τους εξαρτάται λογαριθμικά από τον αριθμό των μοναδικών λέξεων στον πίνακα. Για εφαρμογές μαζικής ανάλυσης δεδομένων όπου τα δεδομένα εισέρχονται στην βάση και δεν ανανεώνονται σε πραγματικό χρόνο, όπως θα πραγματοποιηθεί στην υλοποίηση αυτής της εργασίας, η χρήση ευρετηρίου GIN αποδίδει καλύτερα.

Για την χρήση των `tsvector` σε μία αναζήτηση προσφέρεται ο τελεστής `@@`, ο οποίος ελέγχει εάν ένα `tsquery` ταιριάζει σε ένα `tsvector` και επιστρέφει το αποτέλεσμα σαν λογική μεταβλητή. Κατά την κατασκευή ενός `tsquery` χρησιμοποιούνται τελεστές για να καθορίσουν πως πρέπει να βρίσκονται οι όροι της αναζήτησης σε ένα `tsvector`. Διαθέσιμοι είναι οι Boolean τελεστές `&` (λογικό και), `|` (λογικό ή) και `!` (λογικό όχι). Για παράδειγμα, το query «`Hello World`» θα επέστρεφε όλα τα κείμενα που περιέχουν την ολόκληρη την φράση, ενώ το query «`Hello AND world`» θα επέστεφε κείμενα που περιέχουν και τις δύο λέξεις, ανεξαρτήτως των θέσεων τους. Τέλος, ο τελεστής `<->` αναζητεί κείμενα όπου οι δύο όροι εμφανίζονται με μέγιστο ένα λέξημα ανάμεσα τους. Ως παράδειγμα, το query «`brown <-> fox`» θα επέστρεφε ένα κείμενο που περιέχει την φράση «brown lazy fox» αλλά όχι την φράση «`brown old and lazy fox`». Αντίστοιχα, υπάρχει και ο τελεστής `<N>` που αναζητεί τους όρους με μέγιστη απόσταση `N` λεξημάτων ανάμεσα τους.

### 2.3.2. Latent Semantic Analysis

Η Latent Semantic Analysis (LSA) είναι μία μαθηματική, στατιστική τεχνική για την εξαγωγή και ανάλυση σχέσεων ανάμεσα σε λέξεις και κείμενα μέσω της ανάλυσης μεγάλου όγκου δεδομένων κειμένου. Η LSA εξάγει πληροφορία τόσο για τις σχέσεις μεταξύ κειμένων με βάση τις λέξεις που τα απαρτίζουν καθώς και για τις σχέσεις μεταξύ λέξεων με βάση την ύπαρξή τους σε κείμενα. Είναι μια τεχνική μη-εποπτευόμενης μάθησης που βασίζεται στη μείωση των διαστάσεων αναπαράστασης των κειμένων (λιγότερες διαστάσεις από ότι μοναδικές λέξεις), αναδεικνύοντας ομοιότητες και σχέσεις μεταξύ όρων, ευνοώντας στην αντιμετώπιση προβλημάτων ανάκτησης πληροφορίας. Ξεκινώντας από μία μεγάλη συλλογή κειμένων ή εγγράφων, κατασκευάζεται ένας πίνακας όρων-εγγράφων (term-document matrix), ο οποίος προσεγγίζεται με έναν πίνακα χαμηλότερης βαθμίδας που δημιουργείται με ανάλυση του σε ιδιάζουσες τιμές (SVD). Σε εφαρμογές ανάκτησης δεδομένων η τεχνική συναντάται και με το όνομα Latent Semantic Indexing.

Η περιγραφή της τεχνικής που ακολουθεί βασίστηκε κυρίως στην δημοσίευση της Dumais S. [10] και στο βιβλίο Understanding Search Engines των Berry και Browne [9]. Για την τεχνική της ανάλυσης σε ιδιάζουσες τιμές αντλήθηκαν πληροφορίες από το βιβλίο του Theodoridis S, Machine Learning: A Bayesian and Optimization Perspective [29].

Το πρώτο βήμα της τεχνικής είναι η αναπαράσταση των κειμένων με τον πίνακα όρων-εγγράφων. Στην βασικότερη του μορφή είναι ένας πίνακας όπου κάθε κείμενο αντιστοιχεί σε μία στήλη και κάθε μοναδική λέξη σε μία γραμμή. Οι τιμές του πίνακα είναι ο αριθμός των φορών που εμφανίζεται η κάθε λέξη στο εκάστοτε κείμενο (συχνότητες όρων, term frequencies). Καθώς σε αυτή την αναπαράσταση δεν διατηρείται καμία πληροφορία όσων αφορά την θέση των λέξεων σε κάθε κείμενο, ονομάζεται και «bag of words». Τα κείμενα μπορεί να είναι ολόκληρα έγγραφα ή μικρότερα εδάφια, αναλόγως με την εφαρμογή. Για πρακτικούς λόγους, μερικές φορές χρησιμοποιείται ένα κάτω όριο για την συχνότητα των όρων, αποκόπτοντας λέξεις που εμφανίζονται ελάχιστες φορές στα έγγραφα. Ενώ είναι πιθανών πως τέτοιοι όροι δεν θα είχαν σημαντική αναπαράσταση στην προσέγγιση χαμηλότερης βαθμίδας, επιβαρύνουν το πρόβλημα υπολογιστικά.

Έστω $X$ ο πίνακας όρων-εγγράφων, διαστάσεων $t \times d$, όπου $t$ ο αριθμός των όρων και $d$ ο αριθμός των εδαφίων. Σύμφωνα με την ανάλυση σε ιδιάζουσες τιμές, υπάρχουν ορθογώνιοι πίνακες $U$ και $V$, με διαστάσεις $t \times t$ και $d \times d$ αντίστοιχα, ώστε να ισχύει:

$$X = U\Sigma V^T$$

όπου $\Sigma$ ένας διαγώνιος πίνακας διαστάσεων $r \times r$. Τα στοιχεία $\sigma_{ii}$ του πίνακα $\Sigma$ ονομάζονται ιδιάζουσες τιμές του $X$ και ισούνται με $\sigma_{ii} = \sqrt{\lambda_i}$, όπου $\lambda_i, i = 1,2, \dots, r$ οι μη-μηδενικές ιδιοτιμές του $XX^T$. Εάν χρησιμοποιηθούν μόνο οι $k$ μεγαλύτερες ιδιάζουσες τιμές και τα αντίστοιχα διανύσματα των $U$ και $V$ (όπου $k$ μικρότερο από την βαθμίδα του $X$), δημιουργείται μία χαμηλότερης βαθμίδας προσέγγιση του $X$:

$$X_k \approx U_k \Sigma_k V_k^T$$

Αυτή η προσέγγιση χρησιμοποιείται στην LSA για την αναπαράσταση του σημασιολογικού χώρου, γνωστός στην βιβλιογραφία ως semantic space. Οι γραμμές του $U_k$ και του $V_k$ είναι τα διανύσματα των όρων και των εδαφίων στον σημασιολογικό χώρο αντίστοιχα. Ενώ οι στήλες (ή γραμμές) του $X_k$ είναι γραμμικά ανεξάρτητες, τα αντίστοιχα διανύσματα των κειμένων (ή των όρων) δεν είναι. Συνεπάγεται λοιπόν ότι μία αναζήτηση μπορεί να ανακτήσει εδάφια τα όποια δεν περιέχουν τους όρους της αναζήτησης, εάν η αντίστοιχη διάσταση στον σημασιολογικό χώρο είναι γραμμικός συνδυασμός των όρων.

Η σύγκριση ανάμεσα στα διανύσματα των εδαφίων ή των όρων γίνεται τυπικά με τον υπολογισμό της γωνίας ανάμεσα στα διανύσματα (cosine similarity ή cosine distance), ή κάποιας άλλης ποσοτικοποίησης της απόστασης όπως η ευκλείδεια απόσταση. Οποιοδήποτε διάνυσμα κειμένου $d_j$ μπορεί να προβληθεί στον νέο χώρο μέσω:

$$d_{j(k)} = \Sigma_k^{-1} U_k^T d_j$$

Όπου στη συνέχεια μπορεί να συγκριθεί με άλλα κείμενα. Αυτή η διαδικασία ονομάζεται folding-in και πρέπει να γίνει με τον ίδιο τρόπου που έγινε και η δημιουργία του σημασιολογικού χώρο (δηλαδή, η κατασκευή του διανύσματος όρων-εγγράφου). Οι όροι μίας αναζήτησης μπορούν να προβληθούν σε αυτόν τον χώρο με σκοπό να παρατηρηθούν ποια άλλα έγγραφα βρίσκονται στην ίδια «γειτονιά». Ομοίως, υπάρχει η δυνατότητα να προβληθεί ένα ολόκληρο κείμενο με σκοπό τον εντοπισμό άλλων που βρίσκονται κοντά στον σημασιολογικό χώρο. Μία άλλη εφαρμογή της LSA είναι η αναπαράσταση μεγάλου αριθμού κειμένων στον χαμηλότερων-διαστάσεων χώρο, όπου θεωρητικά κείμενα που είναι κοντά σημασιολογικά θα βρίσκονται στην ίδια «γειτονιά» και θα μπορούν να εφαρμοστούν αλγόριθμοι συσταδοποίησης (clustering) ώστε να ταξινομηθούν τα κείμενα σε θεματικές ενότητες.

Ο αριθμός των διαστάσεων του σημασιολογικού χώρου ($k$) είναι μία παράμετρος που πρέπει να βελτιστοποιηθεί πειραματικά για κάθε εφαρμογή. Τυπικές τιμές βρίσκονται στο εύρος 50 με 1000, με το 300 να είναι μία καλή αρχή, χωρίς να υπάρχει κάποια θεωρία για την πρόβλεψη της βέλτιστης τιμής. Πολύ μικρές τιμές του $k$ μπορεί να μην παρέχουν αρκετή πληροφορία ώστε να ξεχωρίζουν σημασιολογικά τα κείμενα, ενώ πολύ μεγάλες τιμές μπορεί να μην συνδυάζουν αρκετά τους όρους για να εμφανίζονται σημασιολογικές γειτονιές. Επιπροσθέτως, κατά την δημιουργία του πίνακα όρων-εγγράφων δεν αναπαρίσταται η χωρική πληροφορία του κάθε όρου σε κάθε κείμενο. Αυτό μπορεί εν μέρη να διορθωθεί με την χρήση n-grams, αλλά πολλαπλασιάζεται το υπολογιστικό κόστος.

Πριν την δημιουργία του σημασιολογικού χώρου, με κατάλληλη στάθμιση (weighting) των συχνοτήτων στον πίνακα όρων-εγγράφων, δύναται να βελτιωθεί η απόδοση της τεχνικής. Η στάθμιση αυτή έχει σκοπό να αναδείξει «σημαντικούς» όρους και να μειώσει την επίδραση αυτών που είναι σημασιολογικά ουδέτεροι. Διαισθητικά, όροι που εμφανίζονται σπάνια είναι σημαντικότεροι σε σχέση με όρους που συναντώνται σχεδόν σε όλα τα έγγραφα. Η στάθμιση μπορεί να γίνει για κάθε συχνότητα ξεχωριστά (local weighting) είτε συνολικά για ολόκληρο τον πίνακα (global weighting). Ανάλογα με την εφαρμογή, χρησιμοποιούνται διαφορετικές τεχνικές στάθμισης με σκοπό να αναδείξουν διαφορετικές όψεις των όρων.

Έστω $f_{ij}$ η συχνότητα του όρου του $i$ στο κείμενο $j$. Για local weighting χρησιμοποιήθηκε η λογαριθμική συνάρτηση ως εξής [9]:

$$f'_{ij} = \log(1 + f_{ij})$$

Η στάθμιση αυτή μειώνει την αξία της κάθε περαιτέρω εμφάνισης του κάθε όρου. Για global weighting χρησιμοποιήθηκε ο ορισμός της εντροπίας (κατά Shannon) [9]:

$$1 + \sum_j \frac{p_{ij} \log(p_{ij})}{\log n}$$

όπου $p_{ij} = \frac{f_{ij}}{\sum_j f_{ij}}$ (η πιθανότητα ένας όρος $i$ να εμφανιστεί στο κείμενο $j$) και $n$ ο αριθμός των κειμένων με σκοπό να ενισχυθούν όροι που αντιπροσωπεύουν περισσότερη πληροφορία.

Ο σημασιολογικός χώρος μπορεί να αναπαρασταθεί γραφικά μετά από μία περαιτέρω μείωση των διαστάσεων σε 3 ή 2 (για παράδειγμα, χρησιμοποιώντας Principal Component Analysis – PCA). Με αυτή την αναπαράσταση μπορούν να παρατηρηθούν οι γειτονίες των εγγράφων ή των όρων με «το μάτι».

Αξίζει να σημειωθεί πως ο πίνακας όρων-κειμένων, είτε στην αρχική είτε την προσεγγιστική του μορφή, είναι ένας αραιός πίνακας (sparse matrix). Αυτό είναι φυσικό επόμενο του γεγονότος ότι κάθε τομέας έχει το δικό του, εν' μέρη μοναδικό, λεξιλόγιο και πολλές λέξεις εμφανίζονται μόνο σε μία μειοψηφία των κειμένων. Αυτή η ιδιότητα του πίνακα καθιστά την μέθοδο πρακτικά εφαρμόσιμη, καθώς δεν απαιτείται η δέσμευση ενός πίνακα $t \times d$ μεταβλητών, αλλά μόνο $3n \ll t \times d$, όπου $n$ ο αριθμός των μη μηδενικών στοιχείων. Μία υλοποίηση της τεχνικής η οποία δεν εκμεταλλεύεται την συγκεκριμένη ιδιότητα του πίνακα θα είναι δύσκολο έως αδύνατο να εφαρμοστεί σε οτιδήποτε εκτός από πολύ μικρά δείγματα δεδομένων.

### 2.3.3. Συνδυαστικός αλγόριθμος

Ο τελευταίος αλγόριθμος που εξετάστηκε και υλοποιήθηκε ταξινομεί τα κείμενα με βάση μίας τιμής «σχετικότητας», η οποία εξαρτάται εν μέρη από μία ή παραπάνω full-text αναζητήσεις της Postgres και εν μέρη από την χρήση LSA. Συγκεκριμένα, η σχετικότητα μίας αναζήτησης – query σε σχέση με ένα κείμενο υπολογίζεται με βάση:

$$\text{score} = c_0 x + \sum_{j=1} c_j y_j$$

όπου $x$ το cosine similarity που υπολογίστηκε χρησιμοποιώντας LSA και $y_j$ ίσο με 1 εάν το κείμενο εμφανίζεται στην $j$ αναζήτηση, διαφορετικά 0. Τα $c_0, c_j$ είναι σταθερές για την ρύθμιση του βάρους κάθε μεθόδου. Η συνδυαστική μέθοδος βασίζεται στην LSA για την σημασιολογική αναζήτηση κειμένων και στην Postgres για μία πιο «κυριολεκτική» αναζήτηση για το εάν εμφανίζονται συγκεκριμένοι όροι.

# 3. Μεθοδολογία

Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά η μεθοδολογία και τα βήματα που ακολουθήθηκαν κατά την υλοποίηση της εργασίας.

## 3.1. Συλλογή δεδομένων

Για την υλοποίηση του συστήματος εξόρυξης κειμένου, αρχικά αντλήθηκε μεγάλος όγκος από μεταδεδομένα που αφορούν επιστημονικές δημοσιεύσεις. Μετά την επιλογή των αποθετηρίων που κρίθηκαν κατάλληλα ως προς το περιεχόμενο τους, συγγράφτηκαν προγράμματα/scripts που υλοποιούσαν τα πρωτόκολλα επικοινωνίας κάθε αποθετηρίου όπως αυτά περιγράφονται στο κεφάλαιο 2.

### 3.1.1. Επιλογή αποθετηρίων

Ως πηγές για την άντληση των δεδομένων που χρησιμοποιήθηκαν σε αυτή την εργασία επιλέχθηκαν τρία αποθετήρια / ευρετήρια τα οποία περιγράφονται σύντομα σε αυτή την ενότητα.

Το πρώτο που χρησιμοποιήθηκε είναι το Scopus, μία βάση δεδομένων για περιλήψεις (abstracts) και παραπομπές. Το Scopus είναι διαθέσιμο από το 2004, αναπτύχθηκε από την Elsevier και περιλαμβάνει περιεχόμενο από τουλάχιστον 35.000 peer reviewed ακαδημαϊκούς τίτλους. Το Scopus καλύπτει ένα μεγάλο ποσοστό της έρευνας σε όλα τα πεδία, γεγονός που το καθιστά πολύ σημαντικό εργαλείο σε εφαρμογές ανασκόπησης (αυτόματης ή μη) βιβλιογραφίας. Το Scopus προσφέρει πρόσβαση στην βάση δεδομένων μέσω ενός REST API κατόπιν εγγραφής στην υπηρεσία. Το Scopus επιλέχθηκε ώστε να αποκτηθούν μέσω query σχετικά με το θέμα metadata και περιλήψεις από peer reviewed δημοσιεύσεις σε επιστημονικά περιοδικά.

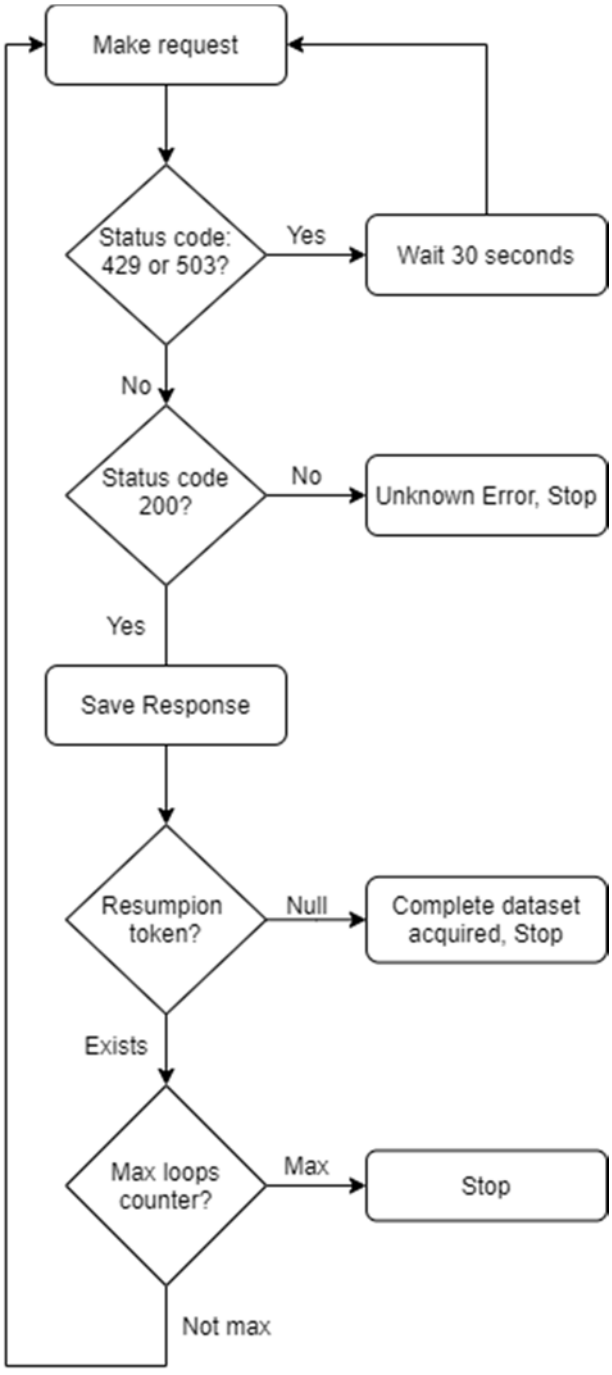
Το OpenGrey είναι μια Ευρωπαϊκή πολυεπιστημονική βάση βιβλιογραφικών αναφορών «γκρίζας βιβλιογραφίας» ανοικτής πρόσβασης [30]. Περιλαμβάνει διδακτορικές διατριβές, έγγραφα συνεδρίων, τεχνικές και ερευνητικές εκθέσεις αλλά και επίσημες δημοσιεύσεις ενώ καλύπτει θεματολογία πάνω στην τεχνολογία, οικονομία, κοινωνικές και ανθρωπιστικές επιστήμες και βιοϊατρική επιστήμη. Αυτή η βιβλιογραφική βάση πρωτοξεκίνησε το 1980 ως SIGLE (System for Information on Grey Literature) και αργότερα το 2006 το περιεχόμενό της μεταφέρθηκε στο OpenSIGLE, μια βάση ανοικτής πρόσβασης, ενώ το 2010 μετονομάστηκε σε OpenGrey Repository (αποθετήριο) καθώς ξεκίνησαν να συμπεριλαμβάνονται έγγραφα πλήρους κειμένου. Τέλος το 2020 ο πάροχος της υπηρεσίας, Inist-CNRS (Institut de l'information Scientifique et Technique) ανακοίνωσε την διακοπή της λειτουργίας το και η υπηρεσία αρχειοθετήθηκε ως βάση δεδομένων στο DANS EASY. Η προγραμματιστική πρόσβαση στο αποθετήριο έγινε με διεπαφή OAI-PMH. Το OpenGrey επιλέχθηκε ώστε να συμπεριληφθεί στην εργασία υλικό γκρίζας βιβλιογραφίας, σε μία προσπάθεια να μετριαστεί το «publication bias» [31]. Καθώς η πιθανότητα να δημοσιευτεί μια έρευνα επηρεάζεται από εάν αυτή κατέληξε σε στατιστικά σημαντικά αποτελέσματα, ευελπιστούμε πως η χρήση δημοσιεύσεων που δεν χρειάζεται να υπερβούν τις διαδικασίες για έκδοση σε peer reviewed journals θα μειώσει αυτή την επίδραση.

Το Zenodo είναι ένα αποθετήριο επιστημονικού περιεχομένου ανοιχτής πρόσβασης υπό την αιγίδα του OpenAIRE και του CERN [32], [33]. Στο Zenodo είναι αποδεκτό κάθε είδους περιεχόμενο, συμπεριλαμβανομένων επιστημονικών δημοσιεύσεων, δεδομένων και λογισμικού. Υπάρχει επίσης η δυνατότητα για πρόσβαση σε προηγούμενες εκδόσεις του περιεχομένου με σκοπό να διατηρείται το ιστορικό εξέλιξης του κάθε έργου. Η πρόσβαση στην υπηρεσία είναι δωρεάν κατόπιν εγγραφής. Για την προγραμματιστική πρόσβαση στο περιεχόμενο, είναι διαθέσιμες διεπαφές τύπου REST και OAI-PMH. Το Zenodo επιλέχθηκε για να συμπληρώσει το υλικό γκρίζας βιβλιογραφίας με πιο πρόσφατες δημοσιεύσεις καθώς το OpenGrey είχε μειωμένο αριθμό νέων μεταφορτώσεων τα τελευταία έτη.

### 3.1.2. Harvester scripts

Η συλλογή των δεδομένων έγινε με την συγγραφή προγραμμάτων/scripts για την εκμετάλλευση των διεπαφών που είναι διαθέσιμες σε κάθε αποθετήριο. Η υλοποίηση των

προγραμμάτων έγινε με την γλώσσα προγραμματισμού R. Στα αποθετήρια Zenodo και OpenGrey η διασύνδεση έγινε μέσω της διεπαφής OAI-PMH, ενώ για το Scopus που δεν υποστηρίζει αυτό το πρωτόκολλο χρησιμοποιήθηκε διεπαφή τύπου REST [34].




*Σχήμα 3: Αλγόριθμος OAI-PMH harvester*

Για την προγραμματιστική πρόσβαση στα αποθετήρια υλοποιήθηκε σε R ένας γενικός OAI-PMH harvester που δουλεύει με όλα τα αποθετήρια που υποστηρίζουν το πρωτόκολλο. Όπως περιεγράφηκε στην ενότητα 2.1.2 για την λήψη δεδομένων απαιτείται μία αίτηση HTTP GET ή POST με εντολή ListRecords. Ο harvester υλοποιήθηκε σε R χρησιμοποιώντας τις βιβλιοθήκες httr για την δημιουργία των αιτήσεων και xml2 για την επεξεργασία των απαντήσεων. Μετά από κάθε αίτηση το πρόγραμμα ελέγχει εάν ο κωδικός κατάστασης ενδεικνύει ότι οι αιτήσεις γίνονται πολύ γρήγορα (status codes: 429 ή 503) και εάν αυτό συμβαίνει μειώνει τη συχνότητα των αιτήσεων περιμένοντας 30 δευτερόλεπτα πριν αποστείλει την επόμενη. Εάν δεν υπάρχει πρόβλημα με τη συχνότητα των αιτήσεων το πρόγραμμα ελέγχει εάν ο κωδικός κατάστασης υποδηλώνει άγνωστο σφάλμα (status code ≠200). Εάν δεν υπάρχει κωδικός κατάστασης σφάλματος το πρόγραμμα μεταβαίνει στην αποθήκευση της απάντησης και την εξαγωγή του resumption token από αυτή. Σε περίπτωση που δεν υπάρχει resumption token σημαίνει ότι λήφθηκαν όλα τα αρχεία και το πρόγραμμα ολοκληρώνεται. Επίσης υπάρχει και ένας μετρητής του αριθμού επαναλήψεων του βρόχου του προγράμματος ώστε να μπορεί να σταματάει όταν πραγματοποιηθεί προκαθορισμένος αριθμός αιτημάτων.

Ο παραπάνω harvester στη συνέχεια καλείται από δεύτερο πρόγραμμα το οποίο παρέχει τις παραμέτρους που θα χρησιμοποιηθούν για τις αιτήσεις σε κάθε αποθετήριο, όπως το URL, το εύρος ημερομηνιών αναζήτησης, την μορφή στην οποία θέλουμε να αναπαρασταθούν τα δεδομένα (metadata prefix), τον αριθμό των αιτήσεων καθώς και το που θα αποθηκευτούν στον δίσκο (path). Χρησιμοποιώντας τη λειτουργία διεργασιών (jobs) του RStudio ήταν δυνατή η παράλληλη εκτέλεση του προγράμματος για κάθε αποθετήριο. Για κάθε ένα από τα δύο αυτά αποθετήρια πραγματοποιήθηκαν χίλιες αιτήσεις οι οποίες αντιστοιχούν σε 99900 εγγραφές. Από το OpenGrey λήφθηκαν εγγραφές από το 2010 μέχρι το 2020 ενώ από το Zenodo μόνο από το 2020 χωρίς κάποιο περεταίρω περιορισμό.

Για την προγραμματιστική πρόσβαση στο Scopus απαιτείται API Key το οποίο δίνει πρόσβαση σε διαφορετικούς πόρους, υπηρεσίες καθώς και ορίζει συγκεκριμένα ποσοστά χρήσης ανάλογα αν ο

χρήστης είναι συνδρομητής. Πιο συγκεκριμένα στην περίπτωση του Scopus Search API που χρησιμοποιήθηκε για τους σκοπούς της εργασίας, τα abstracts δεν περιέχονται στο «STANDARD view» που διατίθεται δωρεάν χωρίς συνδρομή, αλλά στο «COMPLETE view» που διατίθεται μόνο για ακαδημαϊκούς συνδρομητές.[35] Επιπλέον για τους μη-συνδρομητές κάθε αίτημα παρέχει μέχρι και 25 αποτελέσματα, ενώ για τους συνδρομητές μέχρι και 200. Συνολικά εβδομαδιαία επιτρέπεται να γίνουν μέχρι και 20,000 αιτήματα με ταχύτητα όχι μεγαλύτερη των 9 αιτημάτων ανά δευτερόλεπτο. Για τους σκοπούς της εργασίας η συνδρομή καλύφθηκε μέσω του HEAL-Link. Για την έκδοση API key δημιουργήθηκε λογαριασμός στο Elsevier developer portal μέσω των ιδρυματικών στοιχείων του ΠΑΔΑ. Η πρόσβαση για σκοπούς εξόρυξης κειμένου επιτρέπεται στους συνδρομητές για σκοπούς μη-εμπορικής έρευνας.

Για την άντληση δεδομένων από το Scopus κατασκευάστηκε ένα script που πραγματοποιεί HTTP requests, μέσω της R και της βιβλιοθήκης `httr`. Για την αναζήτηση και λήψη των abstracts και των σχετικών τους μεταδεδομένων από την βάση του Scopus, απαιτείται ένα GET request στον πόρο «`https://api.elsevier.com/content/search/scopus`» ορίζοντας τους όρους της αναζήτησής μας, όπως τον τύπο των δημοσιεύσεων που θέλουμε, τις ημερομηνίες, τον αριθμό των αρχείων που θέλουμε να επιστραφεί και τον τρόπο κατάταξης τους. Τέλος στις κεφαλίδες του αιτήματος πρέπει να περιέχεται το API key (X-ELS-APIKey) καθώς και η μορφή στην οποία θέλουμε να είναι κωδικοποιημένη η απάντηση. Η αίτηση επαναλήφθηκε 50 φορές ώστε να ληφθούν 500 εγγραφές. Ανάμεσα σε κάθε αίτηση υπήρξε ένα δευτερόλεπτο αναμονή ώστε να μην ξεπεραστεί το όριο αιτήσεων ανά δευτερόλεπτο. Τα αποτελέσματα αποθηκεύτηκαν προσωρινά στον δίσκο στη μορφή που λήφθηκαν. Στον Κώδικας 3 φαίνεται το αίτημα στην R ενώ στον Πίνακας 4 φαίνονται και εξηγούνται με λεπτομέρεια οι παράμετροι της αίτησης.

```
res <- GET(
  "https://api.elsevier.com/content/search/scopus",
  query = list(
    query = "text+mining AND evidence+based+medicine AND DOCTYPE(ar)",
    count = "10", sort = "relevancy", date = "2010-2020",
    start = x, view = "COMPLETE"
  ),
  add_headers(
    "X-ELS-APIKey" = "API key",
    Accept = "application/json"
  )
)
```

*Κώδικας 3: Αίτημα για αναζήτηση στο SCOPUS μέσω του API. Η αναζήτηση αφορά άρθρα σε επιστημονικά περιοδικά, δημοσιευμένα 2010-2020, σχετικά με την εξόρυξη κειμένου και την τεκμηριωμένη ιατρική.*

| **Παράμετρος** | **Τιμή** | **Περιγραφή** |
|---|---|---|
| **Query** | `text+mining AND evidence+based+medicine AND DOCTYPE(ar)` | Αναζήτηση λογικής Boole που θα πραγματοποιηθεί στη βάση του Scopus. Τα «+» πραγματοποιούν το ρόλο του κενού στη συμβολοσειρά. Με το DOCTYPE(ar) αναζητούμε μόνο για άρθρα. |
| **Count** | 10 | Μέγιστος αριθμός αποτελεσμάτων που πρέπει να επιστραφούν από την αναζήτηση. |
| **Sort** | Relevancy | Αντιπροσωπεύει τον τύπο και τη σειρά ταξινόμησης. Σε αυτήν την περίπτωση |

| | | |
|---|---|---|
| | | επιλέχθηκε ταξινόμηση ανάλογα με τη σχετικότητα, ενώ δεν ορίζεται η σειρά ταξινόμησης (Αύξουσα ή φθίνουσα) και επιλέγεται από το σύστημα η προκαθορισμένη φθίνουσα. |
| **Date** | 2010-2020 | Αντιπροσωπεύει το εύρος ημερομηνιών της αναζήτησης. |
| **Start** | x | Αριθμητική τιμή που αντιπροσωπεύει την θέση εκκίνησης των αποτελεσμάτων. Η τιμή x επιλέγεται από το script ώστε κάθε νέο αίτημα να λαμβάνει τα επόμενα 10 σχετικά αποτελέσματα. |
| **View** | `COMPLETE` | Αντιπροσωπεύει τη λίστα των στοιχείων που θα επιστραφούν στην απάντηση. Το COMPLETE view περιέχει επιπρόσθετα από το STANDARD πληροφορίες σχετικά με : affiliation, author list, abstract, author keywords, article number και funding. |
| **Headers/Κεφαλίδες** | | |
| **X-ELS-APIKey** | API key | Μοναδικό κλειδί που παρέχει πρόσβαση στο API. |
| **Accept** | `application/json` | Αποδεκτή μορφή απάντησης. |

*Πίνακας 4: Παράμετροι για την αίτηση GET προς το search API του Scopus.*

Τα δεδομένα που συλλέχθηκαν τελικά είναι 500 εγγραφές από το Scopus σχετικές με «εξόρυξη κειμένου» και «τεκμηριωμένη ιατρική», 99900 εγγραφές από το OpenGrey και 99900 από το Zenodo. Στην επόμενη ενότητα περιγράφεται πως οργανώθηκε αυτός ο όγκος πληροφορίας για τους σκοπούς της εργασίας.

## 3.2. Διαχείριση δεδομένων

Για την πιο αποδοτική διαχείριση των δεδομένων κρίθηκε σκόπιμη η χρήση μιας βάσης δεδομένων για την αποθήκευσή τους. Μερικοί από τους λόγους για τη χρήση βάσης δεδομένων είναι οι παρακάτω:

- Δυνατότητα διαχείρισης δεδομένων οποιουδήποτε μεγέθους, ακόμη και αυτών που ξεπερνούν τη χωρητικότητα της RAM του υπολογιστή.
- Το προκαθορισμένο σχήμα της βάσης εγγυάται ομοιογένεια μεταξύ των εγγραφών από όποιο αποθετήριο και αν προέρχονται.
- Πολύ γρηγορότερες δυνατότητες αναζήτησης κειμένου καθώς μέσω της R θα έπρεπε να γίνεται προσπέλαση του κειμένου όλων τον εγγραφών για να βρεθούν αυτές που αντιστοιχούν στην αναζήτηση.
- Εύκολη αποθήκευση ενδιάμεσων αποτελεσμάτων σε νέους πίνακες στη βάση χωρίς να επηρεάζονται τα αρχικά δεδομένα.

Το σύστημα διαχείρισης βάσεων δεδομένων που επιλέχθηκε για την συγκεκριμένη εργασία είναι το PostgreSQL καθώς είναι ανοικτού κώδικα και έχει ισχυρές δυνατότητες αναζήτησης κειμένου.

### 3.2.1. Μεταφόρτωση εγγραφών στη βάση δεδομένων

Για την οργάνωση των δεδομένων στη βάση αρχικά πρέπει να οριστεί το σχήμα της. Τα πεδία που επιλέχθηκαν για την αποθήκευση των εγγραφών περιγράφονται στον παρακάτω πίνακα.

| **Όνομα πεδίου** | **Τύπος δεδομένων** | **Περιγραφή** |
|---|---|---|
| **id** | serial | Σειριακή μοναδική ταυτότητα για απαρίθμηση και ταυτοποίηση των εγγραφών στη βάση |
| **publication_date** | date | Ημερομηνία δημοσίευσης |
| **abstract** | text | Περίληψη |
| **doi** | text | Ψηφιακό αναγνωριστικό αντικειμένου (digital object identifier – DOI) |
| **title** | text | Τίτλος της εγγραφής |
| **type** | text | Τύπος δημοσίευσης |
| **lang** | text | Γλώσσα |
| **repo** | text | Αποθετήριο από το οποίο προήλθε η εγγραφή |
| **raw** | xml | Ολόκληρη η εγγραφή όπως επιστράφηκε από το OAI-PMH, ώστε εάν χρειαστεί να μπορούν να αντληθούν περεταίρω πληροφορίες χωρίς να πρέπει να αναζητήσουμε το αρχικό xml στο δίσκο |
| **authors** | text[] | Πίνακας με τα ονόματα των συγγραφέων |
| **textcat_res** | text | Αποτέλεσμα αυτόματης κατηγοριοποίησης της γλώσσας για να κατηγοριοποιηθούν εγγραφές που δεν περιείχαν αυτήν την πληροφορία |
| **tsv** | tsvector | Ο τίτλος και η περίληψη σε διανυσματική μορφή, βελτιστοποιημένη για αναζήτηση κειμένου |

*Πίνακας 5: Πεδία των εγγραφών στη βάση δεδομένων.*

Για την ενσωμάτωση των δεδομένων στη βάση δημιουργήθηκαν προγράμματα, ένα για κάθε πηγή δεδομένων, που εξάγουν από την αρχική μορφή των δεδομένων τα παραπάνω πεδία και τα μεταφορτώνουν στη βάση. Για την επικοινωνία της R με την Postgres χρησιμοποιήθηκαν οι βιβλιοθήκες Rpostgres και DBI. Σε κάθε πρόγραμμα αρχικά δημιουργείται ένα αντικείμενο στην R το οποίο περιέχει της πληροφορίες που απαιτούνται για την σύνδεση της με τη βάση όπως την διεύθυνση IP της βάσης, το όνομά της καθώς και όνομα και κωδικό χρήστη. Σε κάθε εντολή επικοινωνίας με τη βάση, πέρα από το επιθυμητό SQL query συμπεριλαμβάνεται πάντα και αυτό το αντικείμενο σύνδεσης. Στη συνέχεια με ένα βρόχο για κάθε εγγραφή εντοπίζονται τα επιθυμητά πεδία και σώζονται σε μία λίστα, με τις απαραίτητες τροποποιήσεις όπως μαρκάρισμα κενών πεδίων, όπου χρειάζεται, ή τροποποιήσεις της μορφής τους και στο τέλος του βρόχου τα πεδία μεταφορτώνονται στη βάση μέσω ενός SQL query που δέχεται τη λίστα αυτή ως παράμετρο. Η διαδικασία επαναλαμβάνεται στον επόμενο βρόχο για την επόμενη εγγραφή έως ότου προσπελαστούν όλες.

Τα αρχεία που λήφθηκαν από το OpenGrey είναι αποθηκευμένα στο δίσκο ως αρχεία xml όπου το καθένα περιέχει εκατό εγγραφές. Για τη μεταφόρτωση τους στο δίσκο λοιπόν κατασκευάστηκαν δύο βρόχοι, ένας ώστε να διαβάζεται αρχικά ένα αρχείο xml από το δίσκο και στη συνέχεια με τον δεύτερο να διαβάζονται με τη σειρά οι εγγραφές μέσα σε αυτό. Για την εξαγωγή των πεδίων αρχικά διαβάζεται το στοιχείο xml `dc:date` (που παραπέμπει στο Dublin Core) και στη συνέχεια κανονικοποιείται σε μορφή ISO 8601 (`YYYY-MM-DD`) και για τις εγγραφές που δεν περιέχουν μέρα τοποθετούνται στην 1η του μήνα ενώ γι αυτές που δεν έχουν ούτε μήνα τοποθετούνται στην 1η Ιανουαρίου. Έπειτα διαβάζεται το πεδίο `dc:description` για την περίληψη. Στο αποθετήριο περιέχονταν πολλαπλά πεδία `dc:description` που δεν περιείχαν αναγκαστικά την περίληψη αλλά επιπλέον πληροφορίες σχετικά με το ίδρυμα από το οποίο προήλθε η δημοσίευση και την τοποθεσία του. Επομένως για να εντοπιστεί το πεδίο που περιείχε την περίληψη, όλα τα πεδία μεταφορτώθηκαν σε ένα `dataframe`, από τα οποίο αρχικά αφαιρέθηκαν τα πεδία που περιείχαν λιγότερες από 20 λέξεις. Έπειτα με τη βιβλιοθήκη `textcat` τα πεδία που απέμειναν επισημάνθηκαν με τη γλώσσα τους και τέλος εάν υπήρχε αγγλική περίληψη αυτή μεταφορτωνόταν στη βάση. Στην περίπτωση απουσίας μεταφορτωνόταν η τιμή `NA`. Για τον τίτλο διαβάστηκε το πεδίο `dc:title`. Για τον τύπο της εγγραφής στα δεδομένα παρατηρήθηκε πώς συχνά υπήρχαν δύο `dc:type` πεδία με το πρώτο να δηλώνει την κατηγορία ενώ το δεύτερο την υποκατηγορία (πχ. U-Thesis, New Ph.D. thesis) οπότε σε περίπτωση που υπήρχαν δύο πεδία στη βάση μεταφορτωνόταν το δεύτερο. Για την εξαγωγή του DOI προσπελάστηκαν σε βρόχο όλα τα πεδία `dc:relation` και ανιχνεύτηκε και μεταφορτώθηκε στη βάση το πεδίο που οι 4 πρώτοι χαρακτήρες του ήταν «`doi:`». Στο πεδίο raw μεταφορτώθηκε ολόκληρη η εγγραφή όπως επιστράφηκε από το αποθετήριο. Στο πεδίο language διαβάστηκε η τιμή του πεδίου `dc:language` και στο πεδίο repo μεταφορτώθηκε το όνομα του αποθετηρίου «OpenGrey». Τέλος για τους συγγραφείς διαβάστηκαν όλα τα πεδία `dc:creator` και αποθηκεύτηκαν σε ένα array.

Για τα δεδομένα από το Zenodo ακολουθήθηκε η ίδια διαδικασία μόνο που δεν απαιτήθηκε να γίνουν οι αλλαγές στο πεδίο της ημερομηνίας και να εντοπιστεί η περίληψη καθώς υπήρχε μόνο ένα πεδίο `dc:description`.

Για τη μεταφόρτωση των δεδομένων από το Scopus δημιουργήθηκε ένας μόνο βρόχος που διάβαζε τις εγγραφές από τη λίστα στο περιβάλλον της R. Για την ημερομηνία διαβάστηκε το πεδίο `prism:coverDate` και ακολουθήθηκε η ίδια διαδικασία για την κανονικοποίηση της μορφής της. Για την περίληψη διαβάζεται το πεδίο `dc:description`, για τον τίτλο το `dc:title`, τον τύπο το `subtypeDescription` και για το DOI το πεδίο `prism:doi`. Εάν οποιαδήποτε από αυτές τις τιμές είναι άδεια στη βάση μεταφορτώνεται η τιμή `NA`. Το αποθετήριο δεν δίνει πληροφορία για τη γλώσσα οπότε μεταφορτώνεται η τιμή `NA`. Για τους συγγραφείς διαβάζονται σε βρόχο όλα τα ονόματα από τα πεδία author και προστίθενται σε ένα array.

Μετά το τέλος της μεταφόρτωσης όλων των αρχείων στη βάση, δημιουργήθηκε ένα αντίγραφο του πίνακα ως αντίγραφο ασφαλείας σε περίπτωση που συνέβαινε κάποιο λάθος κατά την επεξεργασία των δεδομένων στον πίνακα. Αυτό θα διευκόλυνε την διόρθωση καθώς αντί να ξαναδιαβάζονται τα αρχεία από το δίσκο και να μεταφορτώνονται ξανά στη βάση μπορούν απλά να αντιγραφούν από τον αντίστοιχο πίνακα.

### 3.2.2. Ομογενοποίηση και φιλτράρισμα δεδομένων

Στη συνέχεια έγινε μια εξερεύνηση των δεδομένων που αποκτήθηκαν μέσω SQL queries στη βάση. Αρχικά έγινε μια αναζήτηση και μέτρηση των εγγραφών και των κατηγοριών που αποκτήθηκαν από το Zenodo μέσω μιας εντολής «`SELECT DISTINCT "type", COUNT(*) FROM records WHERE repo = 'zenodo' GROUP BY "type"`».

Τα δεδομένα που αποκτήθηκαν παρατίθενται στον παρακάτω Πίνακας 6. Από αυτές τις κατηγορίες απορρίφθηκαν οι παρακάτω και στη συνέχεια αφαιρέθηκαν από τη βάση : «`dataset, image-diagram, image-drawing, image-figure, image-other, image-photo, image-plot, lesson, other, poster, presentation, publication-annotationcollection,`

`publication-book`, `publication-datamanagementplan`, `publication-deliverable`, `publication-milestone`, `publication-other`, `publication-patent`, `publication-proposal`, `publication-report`, `publication-section`, `publication-softwaredocumentation`, `publication-taxonomictreatment`, `publication-technicalnote`, `software`, `video`». Τα ονόματα των μη επιθυμητών κατηγοριών συγκεντρώθηκαν σε μία λίστα και σβήστηκαν από τη βάση με μία εντολή SQL. Συνολικά απορρίφθηκαν 42,813 εγγραφές.

Στη συνέχεια επαναλήφθηκε η ίδια διαδικασία με τις εγγραφές του OpenGrey. Από αυτό το αποθετήριο τα δεδομένα ήταν κατηγοριοποιημένα σε 176 κατηγορίες όπου οι περισσότερες ήταν υποκατηγορίες πτυχιακών και διδακτορικών διατριβών (πχ. Thesis (D.Med.), Thesis (Psych.D) κλπ.) Έτσι για την ομογενοποίηση των δεδομένων αυτών χρησιμοποιήθηκε η εντολή «`UPDATE records SET "type"= 'thesis' WHERE "type" ILIKE '%thesis%' AND repo = 'OpenGrey'`» που μετονόμασε όλους τους διαφορετικούς τύπους από το συγκεκριμένο αποθετήριο που περιείχαν τη λέξη «thesis» σε thesis. Η μετονομασία αυτή επηρέασε 100,037 εγγραφές. Οι υπόλοιπες κατηγορίες απορρίφθηκαν και σβήστηκαν από τη βάση με μόνο 13 εγγραφές να επηρεάζονται. Από το Scopus καθώς είχαν ληφθεί μόνο άρθρα μέσω αναζήτησης, η βάση περιείχε μόνο άρθρα.

| **Τύπος εγγραφής** | **Ποσότητα** | **Τύπος εγγραφής** | **Ποσότητα** |
|---|---|---|---|
| dataset | 10379 | publication-datamanagementplan | 1 |
| image-diagram | 4 | publication-deliverable | 338 |
| image-drawing | 113 | publication-milestone | 18 |
| image-figure | 8384 | publication-other | 614 |
| image-other | 6 | publication-patent | 20 |
| image-photo | 1465 | publication-proposal | 19 |
| image-plot | 1 | publication-report | 848 |
| lesson | 94 | publication-section | 439 |
| other | 298 | publication-softwaredocumentation | 22 |
| poster | 636 | publication-taxonomictreatment | 11041 |
| presentation | 1981 | publication-technicalnote | 112 |
| publication-annotationcollection | 2 | publication-thesis | 228 |
| publication-article | 51442 | publication-workingpaper | 227 |
| publication-book | 338 | software | 5551 |
| publication-conferencepaper | 5190 | video | 89 |

*Πίνακας 6: Τύποι εγγραφών από το Zenodo*

Συνεχίζοντας την ομογενοποίηση των δεδομένων, μετονομάστηκαν οι τύποι των εγγραφών που αντιστοιχούσαν σε «conference preprints» και «publication-conferenceparper» σε «conference paper». Οι εγγραφές που επηρεάστηκαν ήταν 5,268. Οι 227 εγγραφές «publication-workingpaper» μετονομάστηκαν σε «working paper». Το ίδιο έγινε και για τα άρθρα όπου 51,442 «publication-article» και 500 «Article» μετονομάστηκαν σε «article». Στη συνέχεια όλες οι εγγραφές με πεδίο γλώσσας αγγλικά μετονομάστηκαν σε «eng» με την εντολή: «`UPDATE records SET lang = 'eng' WHERE lang ILIKE 'english' OR lang ILIKE 'en'`», όπου μετονομάστηκαν 87,718 πεδία. Στο επόμενο βήμα διαγράφηκαν 15,754 εγγραφές που δεν ήταν στα αγγλικά αλλά κρατήθηκαν αυτές που δεν είχαν ένδειξη γλώσσας δηλαδή είχαν τιμή NULL στο πεδίο, με την εντολή «`DELETE from records WHERE lang !='eng' AND lang IS NOT NULL`». Ομοίως διαγράφηκαν 63,574 εγγραφές από τη βάση που δεν είχαν abstract, είχαν δηλαδή ενδείξεις `NULL` ή `N/A`. Με αυτό το βήμα ολοκληρώθηκε ο καθαρισμός όσον δεδομένων είχαν κατηγοριοποιηθεί σωστά από το αποθετήριο. Στη βάση όμως είχαν παραμείνει εγγραφές που δεν είχαν ένδειξη γλώσσας καθώς και εγγραφές με πολύ μικρά, λανθασμένα abstracts.

Για την εκμετάλλευση των ικανοτήτων αναζήτησης της Postgres κατασκευάστηκε στη βάση η στήλη `tsv` με την εντολή «`ALTER TABLE records ADD COLUMN tsv tsvector`» και κατασκευάστηκε και το ευρετήριό της (index) με την εντολή «`CREATE INDEX tsv_idx ON records USING gin(tsv)`». Τέλος αυτή η στήλη συμπληρώθηκε με τον τίτλο και το abstract της κάθε εγγραφής σε μορφή `tsvector` με την εντολή «`UPDATE records SET tsv = to_tsvector(title) || to_tsvector(abstract)`». Μέσω αυτού του βήματος μπορεί πλέον να γίνει ταχεία και αποτελεσματική αναζήτηση κειμένου στη βάση. Για παράδειγμα η εντολή «`SELECT * FROM records WHERE tsv @@ to_tsquery('text & mining')`» επιστρέφει σε χρόνο όχι περισσότερο από δευτερόλεπτο όλα τα πεδία όλων των εγγραφών που περιέχουν στον τίτλο ή στην περίληψη τη λέξη «`text`» και τη λέξη «`mining`».

Το επόμενο βήμα ήταν ο καθαρισμός της βάσης από εγγραφές που δεν ήταν σωστά χαρακτηρισμένες, όπως για παράδειγμα εγγραφές που δεν ήταν στην Αγγλική και δεν είχαν ετικέτα γλώσσας. Για την αφαίρεση των εγγραφών δεν είχαν χρήσιμη περίληψη αλλά λανθασμένες καταχωρήσεις που περιείχαν μερικές λέξεις ή έναν μικρό τίτλο, χρησιμοποιήθηκε η εντολή «`DELETE FROM records WHERE length(tsv) < 15`» όπου αφαιρέθηκαν 3,086 εγγραφές που είχαν περίληψη μικρότερη των 15 λεξημάτων. Για τον χαρακτηρισμό των μη-αγγλικών εγγραφών που δεν είχαν ετικέτα, δημιουργήθηκε μια συνάρτηση (UDF) στην Postgres σε R όπου χρησιμοποιήθηκε η βιβλιοθήκη `textcat` για την κατηγοριοποίηση της γλώσσας των άγνωστων περιλήψεων. Η συνάρτηση αυτή δημιουργήθηκε όπως φαίνεται στον Κώδικας 4.

```
res <- dbSendQuery(
  con,
  "CREATE OR REPLACE FUNCTION textcat(abstract character)
                  RETURNS character AS '
                  library(textcat)
                  return(textcat(abstract)) '
                  LANGUAGE 'plr' STRICT;"
)
```

*Κώδικας 4: User defined function για την αναγνώριση της γλώσσας ενός κειμένου με χρήση R και της βιβλιοθήκης textcat*

Στη συνέχεια η συνάρτηση κλήθηκε με την εντολή «`UPDATE records r SET textcat_res = textcat(r.abstract)`» που εφαρμόζει τη συνάρτηση `textcat` που δημιουργήσαμε στη στήλη abstract του πίνακα records. Στη συνέχεια οι εγγραφές με πεδίο «`textcat_res`» που δεν ήταν αγγλικά αφαιρέθηκαν με την εντολή «`DELETE FROM records WHERE textcat_res != 'english'`». Με αυτήν την εντολή αφαιρέθηκαν από τη βάση 1,860 εγγραφές.

## 3.3. Αναζήτηση στα δεδομένα

Σε αυτή την ενότητα περιγράφεται η υλοποίηση των μεθόδων αναζήτησης των σχετικών με το θέμα εγγραφών.

### 3.3.1. Αναζήτηση μέσω Postgres Full Text Search

Η πρώτη προσέγγιση που υλοποιήθηκε ήταν η χρήση του full text search της Postgres. Μέσω αυτού αρχικά πραγματοποιήθηκε αναζήτηση για τις λέξεις κλειδιά «`text`» και «`mining`». Με το σύμβολο «`&`» αναζητήθηκαν οι εγγραφές που περιείχαν στην περίληψη ή στον τίτλο τους ταυτόχρονα τις λέξεις «`text`» και «`mining`» σε οποιοδήποτε σημείο. Έπειτα αναζητήθηκαν οι εγγραφές που περιείχαν τη φράση «`text mining`».

```
text_and_mining <- dbGetQuery(
  con,
  "SELECT * FROM records WHERE tsv @@ to_tsquery('text & mining')"
)
text_near_mining <- dbGetQuery(
  con,
  "SELECT * FROM records WHERE tsv @@ to_tsquery('text <-> mining')"
)
```

Για αναφορά μετρήθηκαν πόσες εγγραφές που προήλθαν από την αναζήτηση «`text mining & evidence based medicine`» στο Scopus δεν εμφανίστηκαν στις παραπάνω αναζητήσεις μέσω του παρακάτω κώδικα.

```
missingand <- dbGetQuery(
  con,
  "SELECT * FROM records WHERE
    repo = 'Scopus' AND
    id NOT IN (
      SELECT id FROM records WHERE tsv @@ to_tsquery('text & mining')
    )"
)

missingnear <- dbGetQuery(
  con,
  "SELECT * FROM records WHERE
  repo = 'Scopus' AND
  id NOT IN (
    SELECT id FROM records WHERE tsv @@ to_tsquery('text <-> mining')
  )"
)
```

Στη συνέχεια δοκιμάστηκαν πιο περίπλοκες αναζητήσεις με περισσότερες λέξεις κλειδιά και λογική Boole.

Όπως φαίνεται από τον παρακάτω κώδικα, για να καλυφθούν περισσότερες περιπτώσεις συνώνυμων εννοιών ή διαφορετικής σειράς των λέξεων στο κείμενο και να μεγιστοποιηθούν τα επιστρεφόμενα αποτελέσματα, απαιτούνται όλο και πιο περίπλοκες αναζητήσεις. Ως αποτέλεσμα γίνονται σταδιακά πιο δυσανάγνωστες και πιο δύσκολο να κατασκευαστούν ορθά. Γι' αυτόν το λόγο θεωρήθηκε σκόπιμο να διερευνηθεί και άλλος τρόπος για την αναζήτηση της πληροφορίας στη βάση.

```
keywordsquery <- dbGetQuery(
  con,
  "SELECT * FROM records
  WHERE tsv @@ to_tsquery('(text | literature) & mining')"
)

res <- dbGetQuery(
  con,
  "SELECT * FROM records
  WHERE tsv @@ to_tsquery(
    '(
      ((text | literature) <-> mining) |
      (mining <-> (text | literature)) |
      ((knowledge | information) <-> extraction)) &
      (
        (evidence <-> based <-> medicine) |
        medicine |
        evidence <-> informed |
        evidence-based |
        evidence-info
      )
  ')"
)
```

### 3.3.2. Υλοποίηση Latent Semantic Analysis

Η υλοποίηση της LSA στην R βασίστηκε στην δουλειά των Fresenda κ.ά. [36], κάνοντας τις απαραίτητες αλλαγές για την εκμετάλλευση της βάσης δεδομένων και χρήση των τεχνικών στάθμισης που αναφέρθηκαν στο κεφάλαιο 2. Για την υλοποίηση της LSA αρχικά αντλήθηκαν από τη βάση δεδομένων σε ένα αντικείμενο στο περιβάλλον της R με μία εντολή «SELECT» οι περιλήψεις, οι τίτλοι, το αποθετήριο και οι ταυτότητες των εγγραφών.

```
res <- dbGetQuery(con, "SELECT id, abstract, title, repo FROM records")
```

Στη συνέχεια οι περιλήψεις ενσωματώθηκαν σε ένα «Volatile Corpus» δηλαδή μια συλλογή που διατηρείται πλήρως στη μνήμη και όλες οι αλλαγές πάνω σε αυτό επηρεάζουν μόνο το αντίστοιχο αντικείμενο στο περιβάλλον της R.

```
raw_corpus <- VCorpus(VectorSource(res$abstract),
  readerControl = list(language = "en")
)
```

Έπειτα ενσωματώθηκαν σε αυτό τα υπόλοιπα πεδία ως metadata. Τα «`Volatile Corpora`» καθώς και οι μετασχηματισμοί πάνω σε αυτά έγιναν μέσω της βιβλιοθήκης `tm`.

```
ids <- paste0("id", seq(73200))
i <- 0
raw_corpus <- tm_map(raw_corpus, function(x) {
  i <- i + 1
  meta(x, "id") <- res[i, 1]
  meta(x, "heading") <- res[i, 3]
  meta(x, "origin") <- res[i, 4]
  x
})
```

Στη συνέχεια έγινε μια επεξεργασία του corpus ώστε να αντικατασταθούν τα κεφαλαία γράμματα με μικρά, να αφαιρεθούν τα σύμβολα και να αντικατασταθούν με κενά, να αφαιρεθούν οι κενές λέξεις (stop words) και να αποκοπούν οι καταλήξεις των λέξεων (stemming). Το μη επεξεργασμένο corpus αφαιρέθηκε για να εξοικονομηθεί μνήμη.

```
remove_nonletter <- function(text) {
  # Regex matching anything that is not a lowercase character or space
  return(gsub("[^a-z\\s]+", " ", text))
}

p_corpus <- tm_map(raw_corpus, content_transformer(tolower))
p_corpus <- tm_map(
  p_corpus,
  content_transformer(removeWords), tm::stopwords("en")
)
p_corpus <- tm_map(p_corpus, content_transformer(removeWords), tm::stopword
s("en"))
p_corpus <- tm_map(p_corpus, content_transformer(remove_nonletter))
p_corpus <- tm_map(p_corpus, stemDocument)
remove_nonletter <- function(text) { return(gsub('[^a-z\\s]+', ' ', text))}

p_corpus <- tm_map(raw_corpus, content_transformer(tolower))
p_corpus     <-     tm_map(p_corpus,     content_transformer(removeWords),
tm::stopwords('en'))
p_corpus <- tm_map(p_corpus, content_transformer(remove_nonletter))
p_corpus <- tm_map(p_corpus, stemDocument)
rm(raw_corpus)
```

Αφού έγινε αυτή η επεξεργασία, το corpus χρησιμοποιήθηκε για τη δημιουργία του πίνακα όρων-εγγράφων (term document matrix – TDM). Για την μείωση των διαστάσεων του πίνακα για την κατασκευή του χρησιμοποιήθηκαν όροι που εμφανίζονται στο corpus το λιγότερο τρείς φορές.

```
frequency_range <- c(3, Inf)
tdm <- TermDocumentMatrix(
  p_corpus,
  control = list(bounds = list(global = frequency_range))
)
```

Έπειτα ο πίνακας μετατράπηκε σε «sparse matrix» μέσω του πακέτου matrix, ώστε να καταλαμβάνει λιγότερη μνήμη και να είναι δυνατή η αποδοτικότερη διαχείρισή του.

```
sparse_tdm <- Matrix::sparseMatrix(
  i = tdm$i, j = tdm$j, x = tdm$v,
  dims = c(tdm$nrow, tdm$ncol)
)
dimnames(sparse_tdm) <- dimnames(tdm)
```

Τέλος πριν την ανάλυση του πίνακα σε ιδιάζουσες τιμές πρέπει να αποδοθούν βάρη στο TDM. Επιλέχθηκε να χρησιμοποιηθεί log-entropy (local-global) weighting [9].

Για τον υπολογισμό της εντροπίας των λέξεων αρχικά υπολογίστηκε ο αριθμός των εγγράφων στο TDM και έπειτα ο λογάριθμος του με βάση το δύο.

```
doc_count <- dim(sparse_tdm)[[2]]
log_doc_count <- log2(doc_count)
```

Έπειτα εφαρμόστηκε στον πίνακα μια συνάρτηση που υπολόγισε το λογάριθμο του δύο συν ένα για όλες τις μη μηδενικές τιμές του sparse matrix. Το αντικείμενο sparseMatrix αποθηκεύει τις σειρές ως τριπλέτες, όπου η τιμή του κάθε μη μηδενικού κελιού αντιστοιχείται σε μία γραμμή και στήλη. Οι μηδενικές τιμές δεν αποθηκεύονται. Οι τιμές αποθηκεύονται στη μεταβλητή x που είναι διάνυσμα οπότε αντί η συνάρτηση να εφαρμοστεί σε όλο τον πίνακα, εφαρμόστηκε στη μεταβλητή x.

```
weighted_tdm <- sparse_tdm
weighted_tdm@x <- vapply(sparse_tdm@x, function(x) log2(x + 1), numeric(1))
```

Στη συνέχεια υπολογίστηκαν οι ολικές συχνότητες εμφάνισης των λέξεων στο corpus, και αποδόθηκαν στις διαστάσεις τα ίδια ονόματα ώστε να αντιστοιχούν στις λέξεις.

```
gf <- Matrix::rowSums(sparse_tdm)
names(gf) <- dimnames(sparse_tdm)$Terms
```

Για το global weighting, δημιουργήθηκε συνάρτηση υπολογισμού της εντροπίας, ονομάστηκε «partial_entropy» διότι η εντροπία είναι το άθροισμα αυτού του υπολογισμού συν ένα.

```
partial_entropy <- function(tf, gf) {
  p <- tf / gf
  return((p * log2(p)) / log_doc_count)
}
```

Έπειτα υπολογίστηκε η εντροπία ανά λέξη μέσω βρόχου που βρίσκει τις λέξεις που έχουν μη μηδενική συχνότητα και εφαρμόζει την συνάρτηση της «μερικής εντροπίας», τις αθροίζει και τις προσθέτει με τη μονάδα.

```
word_entropy <- numeric(dim(sparse_tdm)[[1]])
names(word_entropy) <- dimnames(sparse_tdm)$Terms
for (i in 1:dim(sparse_tdm)[[1]]) {
  word_row <- sparse_tdm[i, ]
  non_zero_frequencies <- word_row[which(word_row > 0)]
  word_entropy[i] <- 1.0 + sum(
    mapply(partial_entropy, non_zero_frequencies, gf = gf[i])
  )
}
```

Τέλος τα βάρη της εντροπίας ανά λέξη εφαρμόστηκαν στο locally weighted TDM, μέσω βοηθητικής συνάρτησης σάρωσης για sparse matrix, που υπολογίζει τον πολλαπλασιασμό μόνο στα μη μηδενικά στοιχεία του πίνακα.

```
sweep_sparse <- function(x, margin, stats, fun = "*") {
  f <- match.fun(fun)
  if (margin == 1) {
    idx <- x@i + 1
  } else {
    idx <- x@j + 1
  }
  x@x <- f(x@x, stats[idx])
  return(x)
}

weighted_tdm <- sweep_sparse(weighted_tdm, 1, word_entropy)
```

Στο επόμενο βήμα για την δημιουργία του semantic space πραγματοποιήθηκε η ανάλυση του πίνακα σε ιδιάζουσες τιμές μέσω του πακέτου `RSpectra`.

```
space <- svds(weighted_tdm, lsa_space_dim)
su_mat <- space$d * space$u
svt_mat <- space$d * Matrix::t(space$v)

dimnames(su_mat) <- list(dimnames(weighted_tdm)[[1]], 1:lsa_space_dim)
space <- svds(weighted_tdm, lsa_space_dim)
su_mat <- space$d * space$u
svt_mat <- space$d * Matrix::t(space$v)

dimnames(su_mat) <- list(dimnames(weighted_tdm)[[1]], 1:lsa_space_dim)
dimnames(svt_mat) <- list(1:lsa_space_dim, dimnames(weighted_tdm)[[2]])
```

Μετά την κατασκευή του νοηματικού χώρου, πραγματοποιήθηκαν αναζητήσεις για τους πλησιέστερους «νοηματικούς γείτονες» στις έννοιες που αφορούν το θέμα. Η αναζήτηση και απεικόνιση των πλησιέστερων γειτονικών λέξεων σε συγκεκριμένο λέξημα πραγματοποιήθηκε μέσω της συνάρτησης «`plot neighbors`» της βιβλιοθήκης `LSAFun`. Για παράδειγμα παρακάτω φαίνεται η αναζήτηση των είκοσι πλησιέστερων γειτονικών λέξεων στο λέξημα «medicin» (από medicine).

```
plot_neighbors(
  "medicin",
  connect.lines = "all", col = "rainbow", n = 20, tvectors = su_mat
)
```

Για την αναζήτηση των γειτόνων σε φράσεις, αρχικά η φράση πρέπει να υποστεί την ίδια επεξεργασία με το αρχικό corpus και έπειτα να προβληθεί στο νοηματικό χώρο. Για παράδειγμα για να βρεθούν οι νοηματικοί γείτονες της φράσης «evidence based medicine» δημιουργήθηκε ο παρακάτω κώδικας.

```
customquery <- "evidence based medicine"
customquery <- tolower(customquery)
customquery <- removeWords(customquery, tm::stopwords("en"))
customquery <- remove_nonletter(customquery)
customquery <- stemDocument(PlainTextDocument(customquery))
customquery <- termFreq(customquery)
customquery <- vapply(customquery, function(x) log2(x + 1), numeric(1))
customquery <- mapply(
  function(x, y) x * y, customquery, word_entropy[names(customquery)]
)
customquery <- colSums(customquery * su_mat[names(customquery), ])
plot_neighbors(customquery, 50, tvectors = su_mat)
```

Αντίστοιχα με τον παρακάτω κώδικα αναζητούνται τα 20 documents που είναι πλησιέστερα στο ίδιο query.

```
similardocs <- data.frame(
  neighbors(customquery, 20, tvectors = Matrix::t(svt_mat))
)
```

Για τη βελτιστοποίηση των αποτελεσμάτων δοκιμάστηκαν διαφορετικές διαστάσεις του σημασιολογικού χώρου. Για να μπορέσει να γίνει η ρύθμιση του συστήματος, αρχικά πραγματοποιήθηκε η προβολή του query «text mining evidence based medicine» σε διαφορετικών διαστάσεων σημασιολογικούς χώρους. Συγκεκριμένα η αναζήτηση ξεκίνησε από την τιμή 300 ως συνήθη τιμή στη βιβλιογραφία και ερευνήθηκαν στη συνέχεια οι τιμές 50, 100, 200, 600 έως και 900. Στη συνέχεια για κάθε τιμή, πραγματοποιήθηκε η αναζήτηση των πλησιέστερων γειτόνων λέξεων αλλά και εγγράφων. Για να γίνει η σύγκριση των αποτελεσμάτων, αποθηκεύτηκαν σε νέους πίνακες στη βάση οι ταυτότητες των πρώτων 2000 εγγράφων που ανασύρονταν για το query.

```
similardocs <- data.frame(neighbors(
  customquery, 2000,
  tvectors = Matrix::t(svt_mat)
))
similardocs <- similardocs %>% rename(
  cos_sim = neighbors.customquery..2000..tvectors...Matrix..t.svt_mat..
)
similardocs$id <- as.numeric(rownames(similardocs))

temp <- dbWriteTable(
  con, "temp", similardocs,
  row.names = FALSE, overwrite = TRUE
)
temp <- dbGetQuery(
  con, "SELECT abstract, cos_sim, title, repo, records.id FROM records
        JOIN temp ON records.id = temp.id"
)
```

Στη συνέχεια μέσω ενός `JOIN` μπορούσαν να εξαχθούν και τα υπόλοιπα πεδία των εγγραφών από τον αρχικό πίνακα «`records`». Για την επιλογή της βέλτιστης τιμής χρησιμοποιήθηκαν τα αποτελέσματα που αποκτήθηκαν από το Scopus ως μέτρο σύγκρισης για το πόσα σχετικά με το query έγγραφα εμφανίζονται στα πρώτα 100 και στη συνέχεια 1000 αποτελέσματα.

Για περεταίρω αξιολόγηση των αποτελεσμάτων τα πρώτα εκατό γειτονικά έγγραφα στο query χαρακτηρίστηκαν ως προς τη σχετικότητά τους με το χέρι. Χρησιμοποιήθηκαν δύο ετικέτες, μια για το εάν το κείμενο είναι σχετικό με την εξόρυξη κειμένου και μια για την τεκμηριωμένη ιατρική. Για να γίνει ο χαρακτηρισμός τα αποτελέσματα αποθηκευτήκαν σε πίνακα στην Postgres και χρησιμοποιήθηκαν δύο στήλες τύπου Boole, μια για κάθε ετικέτα. Στη συνέχεια με λογικό AND μπορεί να βρεθεί ποια έγγραφά είναι σχετικά και με τις δύο έννοιες. Ως σχετικά με εξόρυξη κειμένου χαρακτηρίστηκαν έγγραφα που πραγματεύονταν σε μεθόδους εξόρυξης κειμένου, ακόμη και εάν δεν περιέχονταν οι όροι «text mining» συγκεκριμένα στο κείμενο. Τέλος ως σχετικά με τον όρο «evidence based medicine» θεωρήθηκαν κείμενα που είχαν εφαρμογές εξόρυξης πληροφορίας στο πεδίο της βιοϊατρικής.

### 3.3.3. Συνδυασμός μεθόδων με αλγόριθμο ανακατάταξης αποτελεσμάτων

Η τελευταία τεχνική ανάκτησης σχετικών εγγράφων βασίστηκε στον συνδυασμό των δύο προηγούμενων, όπου κατασκευάστηκε ένας αλγόριθμος για την ανακατάταξη των αποτελεσμάτων. Αποφασίστηκε τα έγγραφα να ανακαταταχθούν με βάση ενός «σκορ» που θα υπολογίζεται εφαρμόζοντας βάρη στα κριτήρια της αναζήτησης των δύο μεθόδων, δηλαδή το cosine similarity και την ύπαρξη των λέξεων του query (αναζήτηση μέσω text search).

$$\text{Score} = \frac{1}{3}\text{cosine similarity} + \frac{1}{3}\text{text mining} + \frac{1}{3}\text{evidence based medicine}$$

Για την υλοποίησή του, επαναφέρονται στο περιβάλλον της R μέσω ενός SELECT query τα αποτελέσματα των αναζητήσεων με LSA και στη συνέχεια με JOIN φορτώνονται σε πίνακα οι ετικέτες τους. Τέλος εφαρμόζεται σε αυτά ο αλγόριθμος υπολογισμού του σκορ.

```
data <- dbGetQuery(con, "SELECT * FROM temp ORDER BY cos_sim desc")

ebm <- dbGetQuery(con, "SELECT temp.id FROM temp
                        LEFT OUTER JOIN records
                        ON temp.id = records.id
                      WHERE tsv @@ to_tsquery('evidence<->based<->medicine')
")
txt <- dbGetQuery(con, "SELECT temp.id  FROM temp
                        LEFT OUTER JOIN records
                        ON temp.id = records.id
                        WHERE tsv @@ to_tsquery('text<->mining')
")

for (i in 1:nrow(data)) {
  data$score[i] <- data$cos_sim[i] * 0.33
  if (any(data$id[i] == ebm$id)) {
    data$score[i] <- data$score[i] + 0.33
  }
  if (any(data$id[i] == txt$id)) {
    data$score[i] <- data$score[i] + 0.33
  }
}
```

Η τεχνική αυτή εφαρμόστηκε αρχικά για να ανακατατάξει τα πρώτα εκατό αποτελέσματα που είχαν χαρακτηριστεί με ετικέτες, προωθώντας στην αρχή της λίστας αποτελέσματα που περιείχαν

συγκεκριμένα τις λέξεις που αναζητήθηκαν. Έτσι για παράδειγμα εάν δυο έγγραφα είχαν παρόμοιο cosine similarity, θα εμφανιζόταν πρώτο στη λίστα αυτό που περιείχε και κάποιες από τις λέξεις κλειδιά που αναζητήθηκαν. Σαν δεύτερη προσέγγιση η τεχνική εφαρμόστηκε για να ανακατατάξει τα πρώτα χίλια αποτελέσματα. Με αυτόν τον τρόπο προωθήθηκαν στην κατάταξη έγγραφα που περιείχαν και τις δύο έννοιες (text mining & EBM) ενώ μπορεί να μην είχαν ιδιαίτερα υψηλό cosine similarity.

# 4. Αποτελέσματα

Σε αυτό το κεφάλαιο παραθέτονται τα αποτελέσματα του κάθε βήματος της εργασίας.

Μετά το τέλος του φιλτραρίσματος τα περιεχόμενα της βάσης δεδομένων ανά τύπο εγγραφής ήταν τα παρακάτω:

| | Type | Count | By repo: | Scopus | OpenGrey | Zenodo |
|---|---|---|---|---|---|---|
| **1** | Article | 22,254 | | 498 | - | 21,756 |
| **2** | Conference paper | 3,284 | | - | 73 | 3,211 |
| **3** | Thesis | 47,503 | | - | 47,392 | 111 |
| **4** | Working paper | 159 | | - | - | 159 |
| | Σύνολο: | 73,200 | | 498 | 47,465 | 25,237 |

*Πίνακας 7: Περιεχόμενα της βάσης δεδομένων ανά τύπο αρχείων και ανά αποθετήριο.*

## 4.1. Αποτελέσματα Postgres full-text search

Σε αυτή την ενότητα παραθέτονται αποτελέσματα για μερικές ενδεικτικές αναζητήσεις που πραγματοποιήθηκαν με Postgres full-text search και ο αριθμός των εγγραφών που επέστρεψαν συνολικά και ανά αποθετήριο.

| Query | Results | Scopus | OpenGrey | Zenodo |
|---|---|---|---|---|
| `Text & mining` | 278 | 219 | 26 | 33 |
| `Text <-> mining` | 200 | 173 | 13 | 14 |
| `Evidence & based & medicine` | 108 | 53 | 47 | 8 |
| `Evidence <-> based <-> medicine` | 52 | 38 | 10 | 4 |
| `(text \| literature) & mining` | 337 | 230 | 66 | 41 |
| `text & mining & evidence & based & medicine` | 11 | 11 | - | - |
| `(text <-> mining) & (evidence <-> based <-> medicine)` | 4 | 4 | - | - |
| `(((text \| literature) <-> mining) \|(mining <-> (text \| literature)) \|((knowledge \| information)<-> extraction))&((evidence<->(based \| informed)<-> medicine))` | 5 | 5 | - | - |

*Πίνακας 8: Full text search queries και αριθμός αποτελεσμάτων που επιστρέφονται ανά αποθετήριο.*

Ενδεικτικά οι τίτλοι των πέντε άρθρων που επιστρέφονται με το τελευταίο query είναι οι παρακάτω:

1. "A decision support system for evidence based medicine"
2. "Automating the process of critical appraisal and assessing the strength of evidence with information extraction technology"
3. "Evidence-based Chinese Medicine Clinical Practice Guideline for Stroke in Hong Kong"
4. "Automated confidence ranked classification of randomized controlled trial articles: An aid to evidence-based medicine"
5. "Corpus construction for named entities and entity relations on Chinese electronic medical records"

## 4.2. Αποτελέσματα LSA

Σε αυτή την ενότητα παραθέτονται τα αποτελέσματα τη αναζητήσεων μέσω LSA καθώς και τα αποτελέσματα της αναζήτησης που έγινε για τη βελτιστοποίηση της παραμέτρου k (διαστάσεων semantic space).

Για την αναζήτηση της βέλτιστης τιμής του k δημιουργήθηκαν σημασιολογικοί χώροι με k 50, 100, 200, 300, 600 και 900. Τα αποτελέσματα αξιολογήθηκαν με βάση το πόσα έγγραφα από το Scopus περιέχονταν στα πρώτα εκατό και χίλια αποτελέσματα που επιστράφηκαν για το query «text mining evidence based medicine». Τα αποτελέσματα αυτά φαίνονται στον παρακάτω πίνακα.

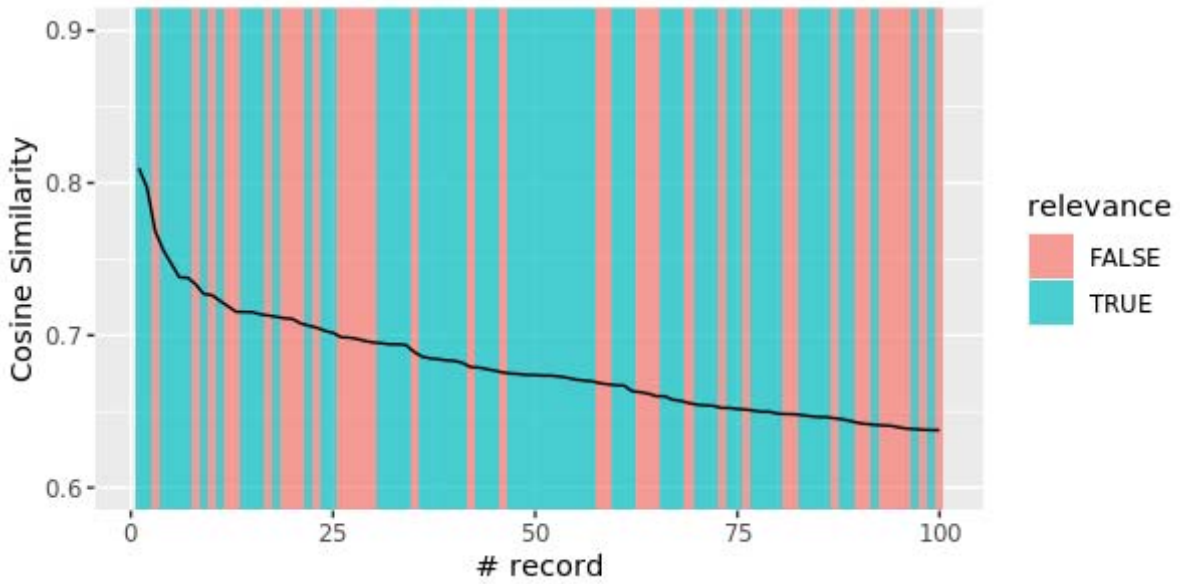
| k | Results in top 100 | Results in top 1000 |
|---|---|---|
| **50** | 64 | 355 |
| **100** | 75 | 398 |
| **200** | 75 | 380 |
| **300** | 74 | 376 |
| **600** | 78 | 364 |
| **900** | 79 | 371 |

*Πίνακας 9: Αποτελέσματα αναζήτησης βέλτιστου k.*

Ως βέλτιστο επιλέχθηκε το k 100 καθώς έχει συγκρίσιμα έως καλύτερα αποτελέσματα εάν ληφθούν υπόψη τα πρώτα 1000 έγγραφα ενώ λόγω της μικρότερης διάστασης απαιτεί πολύ λιγότερους υπολογιστικούς πόρους.

Στη συνέχεια τα αποτελέσματα για το επιλεγμένο βέλτιστο k αξιολογήθηκαν χειροκίνητα, ως προς τη σχετικότητά τους με τη θεματολογία εξόρυξης κειμένου, τεκμηριωμένης ιατρικής ή και με τις δύο.

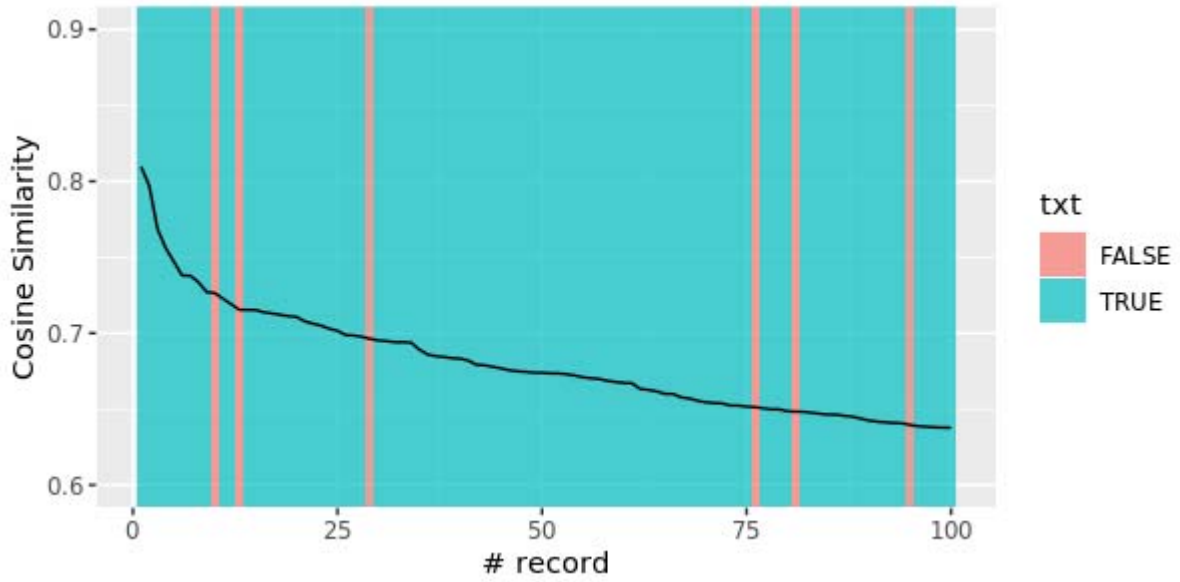
Στο παρακάτω γράφημα φαίνονται τα αποτελέσματα της χειροκίνητης επισήμανσης των πρώτων εκατό εγγράφων που επιστρέφονται από το query «text mining evidence based medicine». Συνολικά στα




*Εικόνα 1:Αποτελέσματα LSA σχετικά με εξόρυξη κειμένου και τεκμηριωμένη ιατρική.*

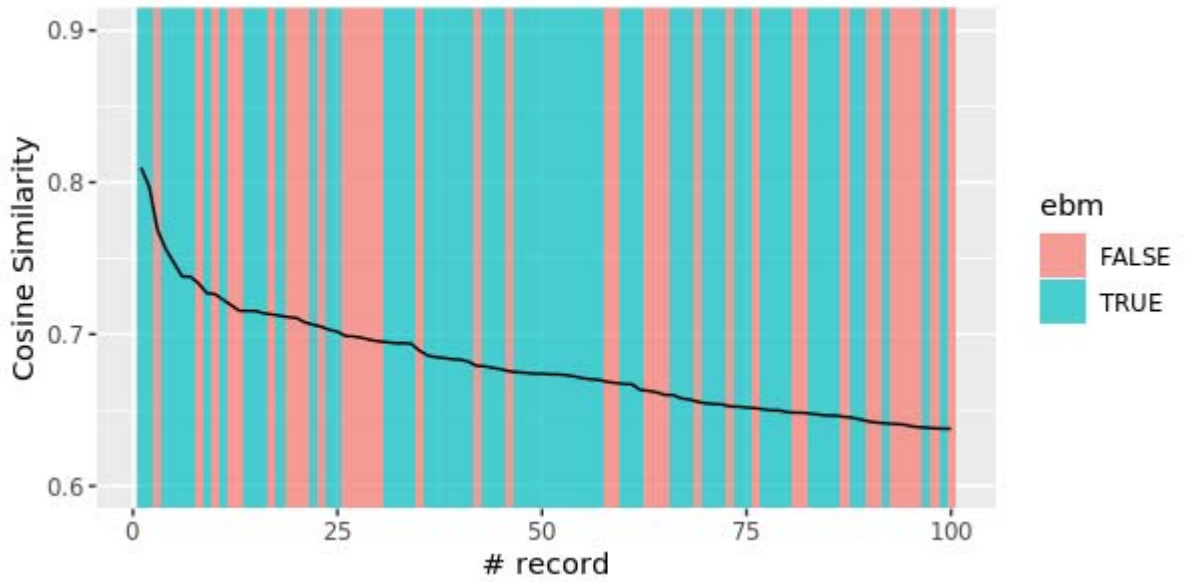
πρώτα 100 αποτελέσματα όπως φαίνεται και στο διάγραμμα τα 63 ήταν σχετικά (relevant) και με τις δυο ζητούμενες έννοιες ενώ 37 όχι.

Στο επόμενο γράφημα φαίνονται οι εγγραφές που ήταν σχετικές με εξόρυξη κειμένου. Συνολικά τα 94 κείμενα ήταν σχετικά ενώ 6 δεν ήταν.




*Εικόνα 2: Αποτελέσματα LSA σχετικά με εξόρυξη κειμένου.*

Τέλος στο τρίτο διάγραμμα απεικονίζονται τα αποτελέσματα που ήταν σχετικά με την τεκμηριωμένη ιατρική. Συνολικά τα 63 ήταν σχετικά ενώ 37 όχι.




*Εικόνα 3: Αποτελέσματα LSA σχετικά με τεκμηριωμένη ιατρική.*

Συγκεντρωτικά στον παρακάτω πίνακα φαίνεται ο αριθμός των εγγραφών που ήταν σχετικός με το κάθε θέμα.

| **Θεματολογία** | **Σχετικά αποτελέσματα** | **Μη σχετικά** |
|---|---|---|
| **Εξόρυξη κειμένου και τεκμηριωμένη ιατρική** | 63 | 37 |
| **Εξόρυξη κειμένου** | 94 | 6 |
| **Τεκμηριωμένη ιατρική** | 63 | 37 |

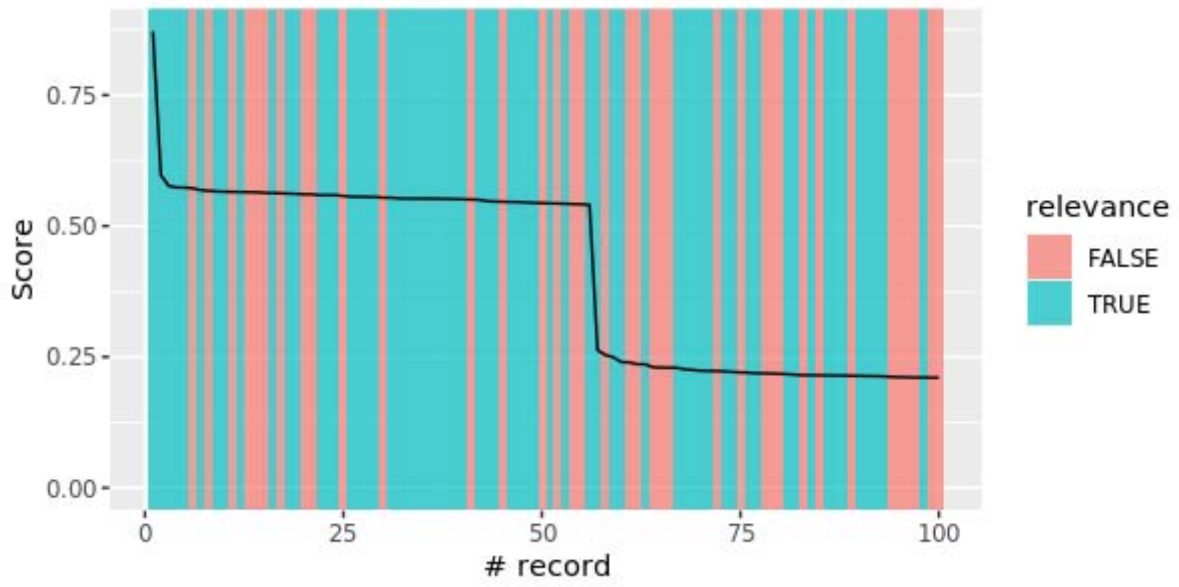
***Πίνακας 10:Συγκεντρωτικός πίνακας αποτελεσμάτων LSA.***

Παρατηρείται πως τα περισσότερα αποτελέσματα ήταν σχετικά με την εξόρυξη κειμένου και λιγότερα με την τεκμηριωμένη ιατρική.

## 4.3. Αποτελέσματα συνδυασμού μεθόδων

Σε αυτή την ενότητα αποδίδονται τα αποτελέσματα του συνδυαστικού αλγορίθμου που αναπτύχθηκε.

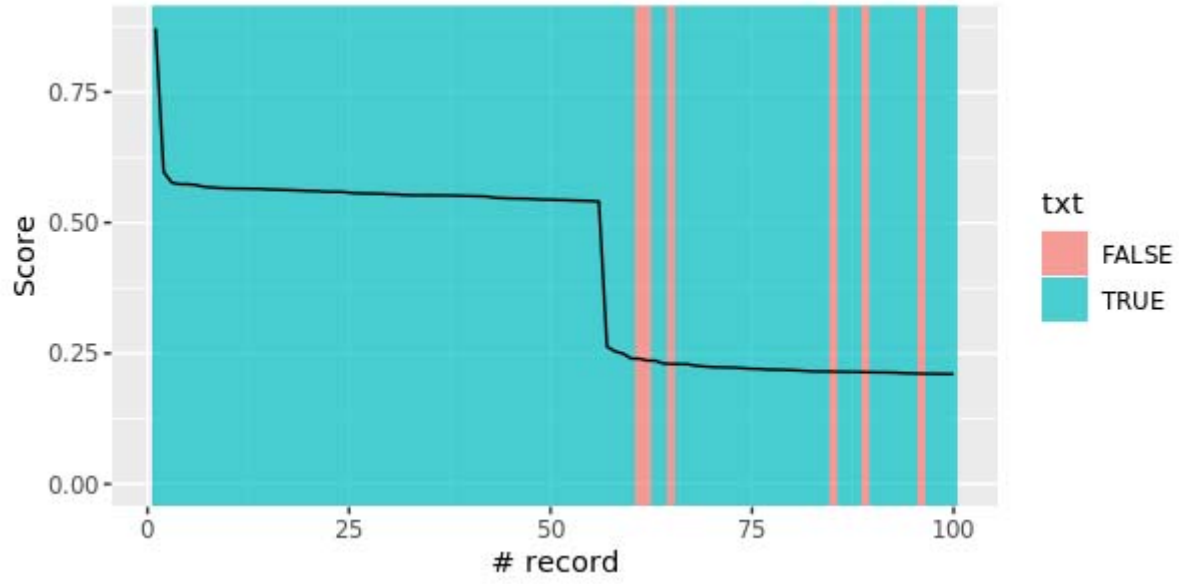
Αρχικά ο αλγόριθμος εφαρμόστηκε στα πρώτα εκατό άρθρα που επιστρέφονται από την LSA.




*Εικόνα 4: Αποτελέσματα μετά από ανακατάταξη μέσω αλγορίθμου.*

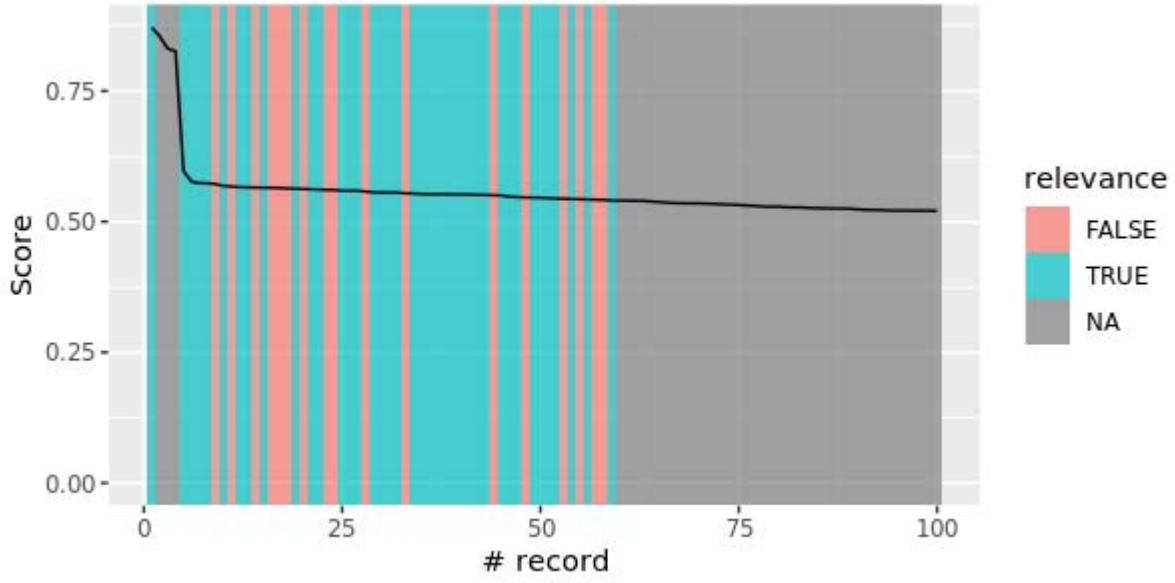
Όπως παρατηρείται μόνο ένα αποτέλεσμα περιέχει ταυτόχρονα τις φράσεις «text mining» και «evidence based medicine», ενώ 57 περιέχουν μόνο τη φράση «text mining».

Απεικονίζοντας τα ανακατανεμημένα άρθρα σχετικά με την εξόρυξη κειμένου μπορεί να παρατηρηθεί πως ο αλγόριθμος ενισχύει τα άρθρα που περιέχουν τις φράσεις αυτούσιες.



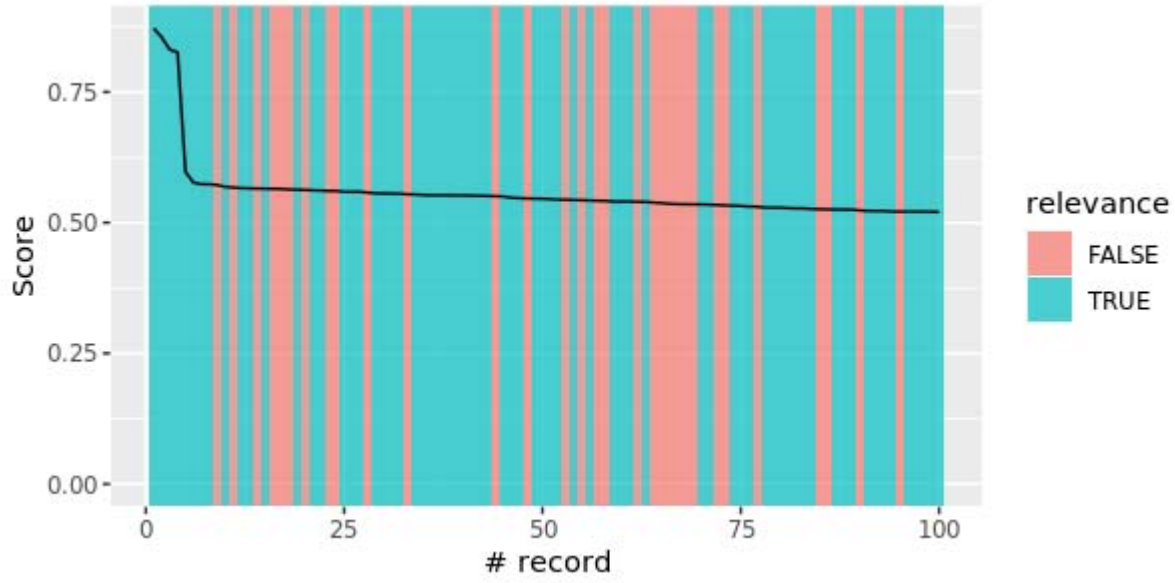

*Εικόνα 5:Αποτελέσματα σχετικά με εξόρυξη κειμένου μετά από ανακατάταξη μέσω αλγορίθμου.*

Στο επόμενο βήμα εφαρμόστηκε ο αλγόριθμος στα πρώτα 1000 αποτελέσματα της LSA. Με γκρίζο χρώμα απεικονίζονται τα έγγραφα που εμφανίστηκαν μετά την αναδιοργάνωση και δεν έχουν ακόμη χαρακτηριστεί ως προς την συνάφειά τους με τα θέματα.




*Εικόνα 6: Πρώτα 1000 αποτελέσματα LSA μετά από ανακατάταξη μέσω αλγορίθμου.*

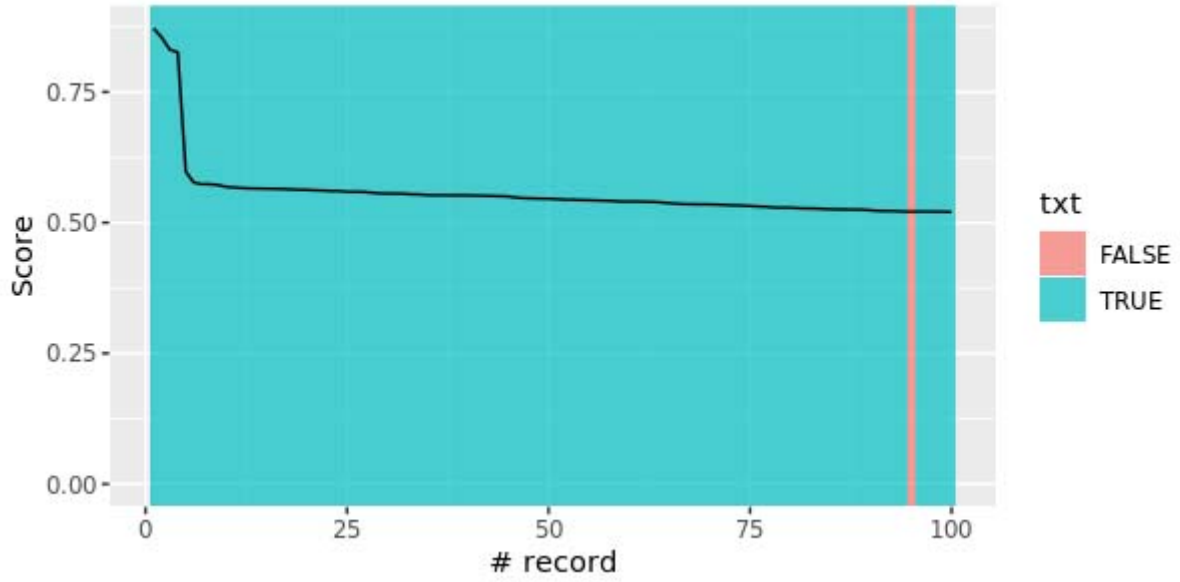
Έπειτα τα αποτελέσματα χαρακτηρίστηκαν ξανά χειροκίνητα.




*Εικόνα 7: Πρώτα 1000 αποτελέσματα LSA μετά από ανακατάταξη μέσω αλγορίθμου και χαρακτηρισμένα ως προς τη θεματολογία τους.*

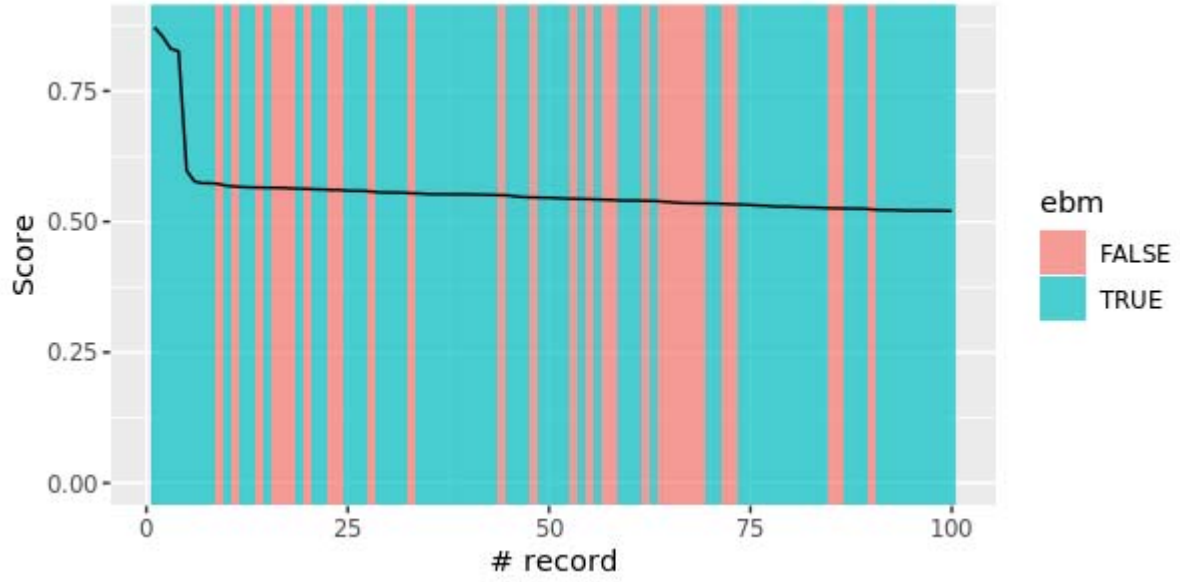
Παρατηρείται πως υπάρχει μικρή βελτίωση από τα πρώτα 100 καθώς εμφανίζονται στα αποτελέσματα 4 έγγραφα ακόμη που περιέχουν και τις δύο αναζητούμενες φράσεις. Επίσης τα συνολικά αποτελέσματα βελτιώνονται από 63 σχετικά με τη θεματολογία άρθρα σε 69.

Στο επόμενο γράφημα απεικονίζονται τα παραπάνω αποτελέσματα χαρακτηρισμένα ως προς τη σχετικότητά τους με την εξόρυξη κειμένου.




*Εικόνα 8: Αποτελέσματα σχετικά με εξόρυξη κειμένου, μετά την ανακατάταξη μέσω αλγορίθμου.*

Έπειτα απεικονίζονται τα ίδια αποτελέσματα χαρακτηρισμένα ως προς τη σχετικότητά τους με την τεκμηριωμένη ιατρική.




*Εικόνα 9: Αποτελέσματα σχετικά με τεκμηριωμένη ιατρική, μετά την ανακατάταξη μέσω αλγορίθμου.*

Τέλος στον παρακάτω πίνακα φαίνονται συγκεντρωτικά ο αριθμός των σχετικών με το κάθε θέμα αποτελεσμάτων.

| **Θεματολογία** | **Σχετικά αποτελέσματα** | **Μη σχετικά** |
|---|---|---|
| **Εξόρυξη κειμένου και τεκμηριωμένη ιατρική** | 69 | 31 |
| **Εξόρυξη κειμένου** | 99 | 1 |
| **Τεκμηριωμένη ιατρική** | 71 | 29 |

*Πίνακας 11: Συγκεντρωτικά αποτελέσματα συνδυαστικού αλγορίθμου όταν αυτός εφαρμόζεται στα πρώτα 1000 αποτελέσματα LSA.*

# 5. Συζήτηση και συμπεράσματα

Σε αυτή την εργασία επιχειρήθηκε να γίνει μια αποτίμηση της εξόρυξης κειμένου ως τεχνικής για την υποβοήθηση της ανασκόπησης βιβλιογραφίας και της τεκμηριωμένης ιατρικής. Για το σκοπό αυτό υλοποιήθηκαν δύο μέθοδοι αναζήτησης σχετικών κειμένων από μεγάλο όγκο άγνωστων δεδομένων.

Η πρώτη μέθοδος που υλοποιήθηκε ήταν η αναζήτηση κειμένου μέσω της δυνατότητας full-text search της Postgres. Αυτή η μέθοδος επιλέχθηκε καθώς είναι ιδιαίτερα ισχυρή και γρήγορη ακόμη και με τον μεγάλο όγκο κειμένων που χρησιμοποιήθηκε. Πράγματι αποδείχθηκε ιδιαίτερα αποδοτική επιστρέφοντας τα ζητούμενα queries σε χρόνο λιγότερο του δευτερολέπτου. Τα αποτελέσματα που επέφερε ήταν περιορισμένα κυρίως λόγω των φυσικών χαρακτηριστικών της γλώσσας όπως πολυσημία και συνωνυμία καθώς και το γεγονός ότι υπάρχουν πολλοί τρόποι να εκφραστεί η ίδια έννοια. Λόγω του τελευταίου, για να καλυφθεί πλήρως μια έννοια θα έπρεπε να συμπεριληφθούν στην αναζήτηση όλα τα συνώνυμά της.

Για την αντιμετώπιση του παραπάνω δοκιμάστηκαν μεγαλύτερες αναζητήσεις με περισσότερες λέξεις κλειδιά και λογική Boole. Το μειονέκτημα αυτής της μεθόδου επαύξησης των αποτελεσμάτων είναι πως περιορίζεται στον χρόνο του χειριστή να αναζητά και να προσθέτει συνώνυμες λέξεις και άλλες σχετικές έννοιες. Κάτι τέτοιο θα μπορούσε να αντιμετωπιστεί εν μέρη με χρήση αυτοματοποιημένης διαδικασίας με χρήση λεξικού, αλλά αυτό δεν διερευνήθηκε στα πλαίσια της εργασίας. Ακόμη και με αυτόν τον τρόπο όμως τα αποτελέσματα περιορίζονται στο βάθος, τη λεπτομέρεια και το μέγεθος του λεξικού. Τέλος τέτοια queries με λογική Boole γραμμένα σε κώδικα SQL στο περιβάλλον της R, γίνονται πολύπλοκα και δυσανάγνωστα αυξάνοντας την πιθανότητα λάθους στην αναζήτηση και μειώνοντας την αξιοπιστία των αποτελεσμάτων.

Η επόμενη μέθοδος που ερευνήθηκε, επιλέχθηκε για να αντισταθμίσει τις παραπάνω αδυναμίες της «κυριολεκτικής» αναζήτησης. Η LSA είναι μια στατιστική μέθοδος που εξάγει αυτόματα τις σχέσεις μεταξύ κειμένων και λέξεων, καθιστώντας δυνατή την αναζήτηση κειμένων που ανήκουν στην ίδια θεματική ενότητα χωρίς να περιέχουν αναγκαστικά τους όρους της αναζήτησης. Ένα μειονέκτημα της μεθόδου είναι πως απαιτεί μεγάλο, αντιπροσωπευτικό αριθμό κειμένων για να δώσει ορθά αποτελέσματα. Η εφαρμογή της μεθόδου αποδείχτηκε επιτυχής καθώς με ένα απλό query επιστράφηκαν επί το πλείστων σχετικά με τη θεματολογία κείμενα.

Συγκεκριμένα τα 63 από τα πρώτα εκατό κείμενα ήταν σχετικά και με τις δύο θεματολογίες ενώ τα 75 από αυτά είχαν επιστραφεί και από τη μηχανή αναζήτησης του Scopus. Επιπλέον τα 64 ήταν σχετικά με την εξόρυξη κειμένου ενώ τα περισσότερα από αυτά που δεν ήταν, είχαν να κάνουν με άλλου είδους ανάλυση κειμένων για παράδειγμα λογοτεχνικών. Τα δεδομένα ήταν λιγότερο σχετικά με την τεκμηριωμένη ιατρική, ίσως λόγω της μικρότερης εκπροσώπησης της έννοιας στο corpus. Αυτή η ίσως μη επαρκής εκπροσώπηση της ιδέας της τεκμηριωμένης ιατρικής στα δεδομένα πιθανώς να συνέβη διότι αναζητήθηκε μαζί με την εξόρυξη κειμένου.

Τα αποτελέσματα της συνδυαστικής μεθόδου βοήθησαν στην προώθηση αποτελεσμάτων που περιείχαν τις αναζητούμενες φράσεις ενώ διατηρούσαν επίσης και υψηλό cosine similarity. Αυτή η μέθοδος συνδυάζει το πλεονέκτημα της θεματολογικής αναζήτησης μέσω LSA αλλά προωθεί αποτελέσματα που περιέχουν και αυτολεξεί τα ζητούμενα. Έτσι στην κορυφή της λίστας εμφανίζονται άρθρα που επιτυγχάνουν υψηλό cosine similarity ενώ περιέχουν και τις φράσεις κλειδιά. Αυτά τα άρθρα είναι λοιπόν πιο πιθανό ότι είναι πραγματικά σχετικά με την αναζήτηση και μέσω της ανακατάταξης αυτής θα είναι τα πρώτα που θα διαβάσει κάποιος που πραγματοποιεί βιβλιογραφική ανασκόπηση. Οι επόμενες εγγραφές που εμφανίζονται μετά την ανακατάταξη είναι αυτές που περιέχουν τουλάχιστον ένα από τα δύο queries και υψηλό cosine similarity. Παρατηρήθηκε πως προωθήθηκαν κάποιες εγγραφές που περιείχαν τη φράση «evidence based medicine» και γι' αυτό βελτιώθηκαν τα αποτελέσματα σε σχέση με αυτά της απλής LSA (από τα 63 σχετικά άρθρα στα 69).

Συνολικά, από τα αποτελέσματα της συγκεκριμένης εργασίας φαίνεται ότι οι τεχνικές εξόρυξης κειμένου θα μπορούσαν να φανούν χρήσιμες στην ανασκόπηση βιβλιογραφίας και την τεκμηριωμένη ιατρική. Ο αλγόριθμος της LSA αποδείχτηκε ικανός στην ταυτοποίησή σχετικών εγγράφων χωρίς την ανάγκη για εκπαίδευσή του σε επισημασμένα δεδομένα. Η δυνατότητά εφαρμογής του σε μεγάλο όγκο δεδομένων τον κάνει κατάλληλο για εφαρμογές ανασκόπησης βιβλιογραφίας, ενώ η δυνατότητα συμπίεσης του σημασιολογικού χώρου σε μικρές διαστάσεις κάνουν την εφαρμογή υπολογιστικά φθηνότερη.

Στην γενικότερη μεθοδολογία που ακολουθήθηκε αναγνωρίζονται συγκεκριμένες αδυναμίες. Όσον αφορά την απόκτηση των δεδομένων, θα ήταν θεμιτό να δοκιμαστούν μεγαλύτερες συλλογές δημοσιεύσεων από μεγαλύτερο αριθμό πηγών ώστε να υπάρχει πιο αντιπροσωπευτικό δείγμα των θεμάτων υπό εξέταση. Για παράδειγμα, αντί να αντληθούν δεδομένα από μία αναζήτηση για εξόρυξη κειμένου και τεκμηριωμένη ιατρική (όπως έγινε με το αποθετήριο Scopus) και να βασιστούν τα αποτελέσματα σε ένα άλλο σύστημα αναζήτησης, θα μπορούσαν να αντληθούν μαζικά δημοσιεύσεις από σχετικά περιοδικά. Κατά την εφαρμογή της LSA, δεν διερευνήθηκε πως επηρεάζουν τα αποτελέσματα οι τεχνικές στάθμισης και εάν η χρήση διαφορετικών εξισώσεων θα μπορούσε να επιφέρει καλύτερα αποτελέσματα στην συγκεκριμένη εφαρμογή. Τέλος, θα είχε επίσης τεχνικό ενδιαφέρον η υλοποίηση μεγαλύτερου μέρους του αλγόριθμου σε SQL procedures ώστε να εκμεταλλευτούμε την παραλληλοποίηση που παρέχεται από το σύστημα.

# 6. Βιβλιογραφία


[1] D. L. Sackett, "Evidence-based medicine," *Semin. Perinatol.*, vol. 21, no. 1, pp. 3–5, Feb. 1997, doi: 10.1016/S0146-0005(97)80013-4.

[2] L. Hunter and K. B. Cohen, "Biomedical Language Processing: Perspective What's Beyond PubMed?," *Mol. Cell*, vol. 21, no. 5, pp. 589–594, Mar. 2006, doi: 10.1016/j.molcel.2006.02.012.

[3] "MEDLINE Overview." https://www.nlm.nih.gov/medline/medline_overview.html (accessed Sep. 23, 2021).

[4] D. Allen and K. J. Harkins, "Too much guidance?," *The Lancet*, vol. 365, no. 9473, p. 1768, May 2005, doi: 10.1016/S0140-6736(05)66578-6.

[5] X. Cheng, Q. Cao, and S. S. Liao, "An overview of literature on COVID-19, MERS and SARS: Using text mining and latent Dirichlet allocation," *J. Inf. Sci.*, p. 0165551520954674, Aug. 2020, doi: 10.1177/0165551520954674.

[6] B.-K. Choi *et al.*, "Literature-based automated discovery of tumor suppressor p53 phosphorylation and inhibition by NEK2," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 115, no. 42, pp. 10666–10671, Oct. 2018, doi: 10.1073/pnas.1806643115.

[7] D. R. Swanson and N. R. Smalheiser, "An interactive system for finding complementary literatures: a stimulus to scientific discovery," *Artif. Intell.*, vol. 91, no. 2, pp. 183–203, Apr. 1997, doi: 10.1016/S0004-3702(97)00008-8.

[8] S. Henry, D. S. Wijesinghe, A. Myers, and B. T. McInnes, "Using Literature Based Discovery to Gain Insights Into the Metabolomic Processes of Cardiac Arrest," *Front. Res. Metr. Anal.*, vol. 6, p. 644728, 2021, doi: 10.3389/frma.2021.644728.

[9] Michael W. Berry and Murray Browne, *Understanding Search Engines: Mathetical Modeling and Text Retreival*, 2nd ed. S.I.A.M., 2005.

[10] S. T. Dumais, "Latent semantic analysis," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, no. 1, pp. 188–230, Sep. 2005, doi: 10.1002/aris.1440380105.

[11] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, "Using latent semantic analysis to improve access to textual information," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, May 1988, pp. 281–285. doi: 10.1145/57167.57214.

[12] W. Tanenbaum, Andrew S. David J., *Δίκτυα Υπολογιστών*, 5th ed. Κλειδάριθμος, 2011.

[13] "HTTP request methods - HTTP | MDN." https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (accessed Sep. 23, 2021).

[14] "RFC7230: Hypertext Transfer Protocol (HTTP/1.1) - Message Syntax and Routing." [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7230#section-3.2.4

[15] "What is a REST API?" https://www.redhat.com/en/topics/api/what-is-a-rest-api (accessed Sep. 23, 2021).

[16] C. Lagoze, H. Van de Sompel, M. Nelson, and S. Warner, "Open Archives Initiative - Protocol for Metadata Harvesting - v.2.0," Jun. 14, 2002. https://www.openarchives.org/OAI/openarchivesprotocol.html (accessed Sep. 23, 2021).

[17] "Open Archives Initiative FAQ." http://www.openarchives.org/documents/FAQ.html#Is%20the%20Open%20Archives%20Initiative%20only%20concerned%20with%20metadata? (accessed Sep. 23, 2021).

[18] "The OCLC/NCSA Metadata Workshop: The Essential Elements of Network Object Description," Mar. 01, 1995. https://www.dublincore.org/news/1995/03-01-the-oclcncsa-metadata-workshop-the-essential-elements-of-network-object-description/ (accessed Sep. 23, 2021).

[19] "ISO 15836:2003 - Information and documentation - The Dublin Core metadata element set," *iTeh Standards Store*. https://standards.iteh.ai/catalog/standards/iso/f2137367-1407-4113-91bc-55406d4f5598/iso-15836-2003 (accessed Sep. 23, 2021).

[20] "DCMI Metadata Terms." https://www.dublincore.org/specifications/dublin-core/dcmi-terms/ (accessed Sep. 23, 2021).

[21] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. McGraw Hill Education, 2020.

[22] Chris Re, "Buffer Management Systems," presented at the CS346, Stanford University, Spring 2015.

[23] E. F. Codd, “A relational model of data for large shared data banks,” *Commun. ACM*, vol. 26, no. 1, pp. 64–69, Jan. 1983, doi: 10.1145/357980.358007.
[24] “MariaDB Foundation - MariaDB.org.” https://mariadb.org/ (accessed Sep. 23, 2021).
[25] P. G. D. Group, “PostgreSQL,” *PostgreSQL*, Sep. 23, 2021. https://www.postgresql.org/ (accessed Sep. 23, 2021).
[26] “PostgreSQL 13 Documentation - Chapter 12 Full Text Search,” *PostgreSQL Documentation*, Aug. 12, 2021. https://www.postgresql.org/docs/13/textsearch.html (accessed Sep. 23, 2021).
[27] *PostgreSQL Source Code (tag REL_13_4, e849f3f)*. Accessed: Sep. 15, 2021. [Online]. Available: https://github.com/postgres/postgres
[28] “PostgreSQL 13 Documentation - Section 12.9. GIN and GiST Index Types,” *PostgreSQL Documentation*, Aug. 12, 2021. https://www.postgresql.org/docs/13/textsearch-indexes.html (accessed Sep. 23, 2021).
[29] Sergios Theodoridis, *Machine Learning, A Bayesian and Optimization Perspective*, 2nd ed. Academic Press, 2020.
[30] Inist-CNRS, “System for Information on Grey Literature in EuropeOpenGrey.” Data Archiving and Networked Services (DANS), 2021. doi: 10.17026/DANS-XTF-47W5.
[31] J. D. Scargle, “Publication Bias (The ‘File-Drawer Problem’) in Scientific Inference,” *arXiv:physics/9909033*, Sep. 1999, Accessed: Sep. 11, 2021. [Online]. Available: http://arxiv.org/abs/physics/9909033
[32] M. Dillen, Q. Groom, D. Agosti, and L. H. Nielsen, “Zenodo, an Archive and Publishing Repository: A tale of two herbarium specimen pilot projects.,” *Biodivers. Inf. Sci. Stand.*, no. 2, Jun. 2019, Accessed: Sep. 23, 2021. [Online]. Available: https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=25350897&v=2.1&it=r&id=GALE%7CA646406315&sid=googleScholar&linkaccess=abs
[33] “Zenodo - Research. Shared.” https://about.zenodo.org/ (accessed Aug. 07, 2021).
[34] “Scopus Search API.” https://dev.elsevier.com/documentation/ScopusSearchAPI.wadl (accessed Sep. 12, 2021).
[35] “Elsevier Developer Portal.” https://dev.elsevier.com/api_key_settings.html (accessed Dec. 13, 2020).
[36] J. Fresneda, D. Gefen, J. Endicott, J. Miller, and K. Larsen, “A Guide to Text Analysis with Latent Semantic Analysis in R with Annotated Code: Studying Online Reviews and the Stack Exchange Community,” *Commun. Assoc. Inf. Syst.*, vol. 41, Nov. 2017, doi: 10.17705/1CAIS.04121.

# 7. Παράρτημα – Κώδικας

Σε αυτό το κεφάλαιο παρατίθενται ολόκληρα τα αρχεία κώδικα που αναπτύχθηκαν για την υλοποίηση της εργασίας.

## 7.1. dlrepo.r

Πρόγραμμα OAI-PMH harvester

```
library(httr)
library(xml2)
library(rlang)

resume.token = NULL
i = 1

repeat {
  query.params = list(verb = "ListRecords")

  if (!is.null(resume.token)) {
    query.params[["resumptionToken"]] = resume.token
  } else {
    query.params[["metadataPrefix"]] = metadata.prefix
    query.params[["from"]] = start.date
    query.params[["until"]] = end.date
  }

  r <-
    GET(
      url,
      query = query.params,
                add_headers(`Accept-Encoding`   =   "gzip,   deflate",   `User-
Agent` = "Thesis on Text Mining <bme14016@uniwa.gr>")
    )

  if (r$status_code == 503) {
    cat(" ! 503 waiting a bit")
    Sys.sleep(30)
    next
  }
  if (r$status_code == 429) {
    cat(" ! 429 waiting a bit")
    Sys.sleep(30)
    next
  }

  if (r$status_code != 200) {
    cat(" ! unknown error")
    cat(r)
    break
  }

  body <- content(r, "text")

  path = file.path(save.path, paste(i, ".xml", sep = ""))
  cat(body, file = path)
  cat("downloaded",path,"\n")
```

```
  xmlbody = read_xml(body)
  token_node = xml_find_all(xmlbody, "//d1:resumptionToken")
  resume.token = xml_text(token_node)


  if (is_empty(resume.token)) {
    cat(" ! No token, breaking.")
    break
  }

  i = i + 1
  if (i == maxloops) {
    break
  }
  Sys.sleep(1)
}
```

## 7.2. rundownloadjobs.r

Πρόγραμμα που καλεί τον παραπάνω harvester και δημιουργεί διεργασίες στο Rstudio που μεταφορτώνουν τα δεδομένα από τα αποθετήρια.

```
#---------------------------------OpenGrey-----------------------------------
----
url = "http://www.opengrey.eu/oai/"
start.date = "2010-01-01T00:00:00Z"
end.date = "2020-12-31T23:59:59Z"
metadata.prefix = "oai_dc"
save.path = "~/Text mining/Text mining/Repos/OpenGrey"

maxloops = 1000

rstudioapi::jobRunScript(path = "dlrepo.R", importEnv = TRUE)

#---------------------------------Zenodo-------------------------------------
----
url = "https://zenodo.org/oai2d"
start.date = "2020-01-01T00:00:00Z"
end.date = "2020-12-31T23:59:59Z"
metadata.prefix = "oai_dc"
save.path = "~/Text mining/Text mining/Repos/Zenodo"

maxloops = 1000

rstudioapi::jobRunScript(path = "dlrepo.R", importEnv = TRUE)
```

## 7.3. scopus_request.r

Πρόγραμμα που πραγματοποιεί την αίτηση λήψης αρχείων στο Scopus.

```
library(httr)

x = 0
entries = c()

for (i in 1:50) {
```

```
  res <- GET(
    "https://api.elsevier.com/content/search/scopus",
    query = list(query = "text+mining AND evidence+based+medicine AND DOCTYP
E(ar)", count = "10", sort = "relevancy",
                 date = "2010-2020", start = x , view = "COMPLETE"),
    add_headers("X-ELS-APIKey" = "API key",
                Accept = "application/json")
    )

  x = x + 10
  Sys.sleep(1)

  data = content(res, "parsed")
  entries = c(entries,data$`search-results`$entry)
}
```

## 7.4. scopus_to_db.r

Πρόγραμμα μεταφόρτωσης των δεδομένων που αποκτήθηκαν από το Scopus στη βάση.

```
library(xml2)
library(DBI)
library(RPostgres)

con <- dbConnect(
  Postgres(),
  host = "192.168.158.74",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

for (entry in entries) {

  date = entry[["prism:coverDate"]]
  if (grepl("\\d{4}$", date)) {
    date = paste0(date, "-01-01")
  } else if (grepl("\\d{4}-\\d{1,2}$", date)) {
    date = paste0(date, "-01")
  } else if (!grepl("\\d{4}-\\d{1,2}-\\d{1,2}$", date)) {
    date = NA
  }

  description = entry[["dc:description"]]
  if(is.null(description)){
    description = NA
  }
  title = entry[["dc:title"]]
  type = entry[["subtypeDescription"]]
  doi = entry[["prism:doi"]]
  if (is.null(doi)){
    doi = NA
  }
  raw = NA
  lang = NA
  repo = "Scopus"
  for (author in entry[["author"]]) {
```

```
  }
  creators = c()
  for (author in entry[["author"]]) {
    creators = c(creators,author[["authname"]])

  }
  authors = paste0("{", paste0("\"", creators, "\"", collapse = ","), "}")

  res <- dbSendQuery(
    con,
    "INSERT INTO records (publication_date,abstract,title,type,doi,raw,lang,
authors,repo) VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9);",
    params = list(date,
                  description,
                  title,
                  type,
                  doi,
                  raw,
                  lang,
                  authors,
                  repo)
  )
  dbClearResult(res)
  print(title)
}
```

## 7.5. Opengrey_to_db.r

Πρόγραμμα μεταφόρτωσης των δεδομένων που αποκτήθηκαν από το OpenGrey στη βάση.

```
library(xml2)
library(DBI)
library(RPostgres)
library(textcat)
library(stringr)

con <- dbConnect(
  Postgres(),
  host = "192.168.158.74",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

file.names <- dir("./Repos/OpenGrey/", pattern = ".xml")

for (i in 1:length(file.names)) {
  res <- read_xml(file.path("./Repos/OpenGrey/", file.names[i]))
  recs = xml_find_all(res, "//d1:record")

  for (rec in recs) {
    date = xml_text(xml_find_first(rec, ".//dc:date"))
    if(grepl("\\d{4}$", date)){
      date = paste0(date,"-01-01")
    }else if (grepl("\\d{4}-\\d{1,2}$",date)){
      date = paste0(date,"-01")
```

```
    }else if (!grepl("\\d{4}-\\d{1,2}-\\d{1,2}$",date)){
      date = NA
    }

    descriptions = data.frame(xml_text(xml_find_all(rec, ".//dc:description"
)))
    names(descriptions)[1] = "abs"
    descriptions$wordcount = lapply(descriptions$abs,FUN = function(x){retur
n(str_count(x,' '))})
    descriptions = descriptions[descriptions$wordcount > 20,]
    descriptions$lang = lapply(descriptions$abs,FUN = function(x){return(tex
tcat(x))})
    descriptions = descriptions[descriptions$lang == "english",]

    if (nrow(descriptions) == 0) {
      description = NA
    } else {
      description = descriptions[1, "abs"]
    }

    title = xml_text(xml_find_first(rec, ".//dc:title"))
    type_nodes = length(xml_find_all(rec, ".//dc:type"))
    if (type_nodes < 2){
      type = xml_text(xml_find_all(rec, ".//dc:type")[[1]])
    } else {
      type = xml_text(xml_find_all(rec, ".//dc:type")[[2]])
    }
    # doi = xml_text(xml_find_all(rec,".//dc:relation")[[1]])
    doi = NA
    for (thing in xml_text(xml_find_all(rec, ".//dc:relation"))) {
      if (substr(thing, 1, 4) == "doi:") {
        doi = thing
        break
      }
    }

    raw = toString(rec)
    lang = xml_text(xml_find_first(rec, ".//dc:language"))
    repo = "OpenGrey"
    creators = xml_text(xml_find_all(rec, ".//dc:creator"))
    authors = paste0("{", paste0("\"", creators, "\"", collapse = ","), "}")

    res <- dbSendQuery(
      con,
      "INSERT INTO records (publication_date,abstract,title,type,doi,raw,lan
g,authors,repo) VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9);",
      params = list(date,
                    description,
                    title,
                    type,
                    doi,
                    raw,
                    lang,
                    authors,
                    repo)
    )
```

```
    dbClearResult(res)
  }
  print(i)
}
```

## 7.6. Zenodo_to_db.r

Πρόγραμμα μεταφόρτωσης των δεδομένων που αποκτήθηκαν από το Zenodo στη βάση.

```
library(xml2)
library(DBI)
library(RPostgres)

con <- dbConnect(
  Postgres(),
  host = "192.168.158.74",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

file.names <- dir("./Repos/Zenodo/", pattern = ".xml")

for (i in 1:length(file.names)) {
  res <- read_xml(file.path("./Repos/Zenodo/", file.names[i]))
  recs = xml_find_all(res, "//d1:record")

  for (rec in recs) {
    date = xml_text(xml_find_first(rec, ".//dc:date"))
    description = xml_text(xml_find_first(rec, ".//dc:description"))
    title = xml_text(xml_find_first(rec, ".//dc:title"))
    type_nodes = length(xml_find_all(rec, ".//dc:type"))
    if (type_nodes < 2){
      type = xml_text(xml_find_all(rec, ".//dc:type")[[1]])
    } else {
      type = xml_text(xml_find_all(rec, ".//dc:type")[[2]])
    }
    doi = NA
    for (thing in xml_text(xml_find_all(rec, ".//dc:relation"))) {
      if (substr(thing, 1, 4) == "doi:") {
        doi = thing
        break
      }
    }
    raw = toString(rec)
    lang = xml_text(xml_find_first(rec, ".//dc:language"))
    repo = "zenodo"
    creators = xml_text(xml_find_all(rec, ".//dc:creator"))
   creators = lapply(creators, FUN = function(x){return((gsub("\"","'",x)))})
    authors = paste0("{", paste0("\"", creators, "\"", collapse = ","), "}")
    res <- dbSendQuery(
      con,
      "INSERT INTO records (publication_date,abstract,title,type,doi,raw,lan
g,authors,repo) VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9);",
      params = list(date,
                    description,
                    title,
                    type,
```

```
                    doi,
                    raw,
                    lang,
                    authors,
                    repo)
    )
    dbClearResult(res)
  }
  print(i)
}
```

## 7.7. Db_cleaning.r

Πρόγραμμα εκκαθάρισης ανεπιθύμητων αρχείων από τη βάση.

```
library(ggplot2)
library(DBI)
library(RPostgres)
library(dplyr)

con <- dbConnect(
  Postgres(),
  host = "192.168.158.74",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

# Get all types from zenodo
Zenodo_recs_per_type <- dbGetQuery(
  con,
  "SELECT DISTINCT \"type\", COUNT(*) FROM records WHERE repo = 'zenodo' GRO
UP BY \"type\" "
)

# Select which types to drop
zenodo_drops = c(
  "dataset",
  "image-other",
  "other",
  "publication-deliverable",
  "publication-proposal",
  "publication-taxonomictreatment",
  "software",
  "image-diagram",
  "image-photo",
  "publication-milestone",
  "publication-report",
  "publication-technicalnote",
  "video",
  "image-drawing",
  "image-plot",
  "presentation",
  "publication-section",
  "image-figure",
  "lesson",
```

```
  "publication-annotationcollection",
  "publication-softwaredocumentation",
  "publication-datamanagementplan",
  "publication-patent",
  "poster",
  "publication-other",
  "publication-book"
)

zendrop = paste0("'", zenodo_drops, "'", collapse = ",") # Create query string

# Drop irrelevant types
res <- dbSendQuery(con,
                   paste0(
                  "DELETE FROM records WHERE repo = 'zenodo' AND \"type\" IN (",
                       zendrop,
                       ")"
                     ))
zenodo_dropped_categories <- dbGetRowsAffected(res)
dbClearResult(res)

#Get all types from open grey
OG_recs_types <- dbGetQuery(
  con,
  "SELECT DISTINCT \"type\", COUNT(*) FROM records WHERE repo = 'OpenGrey' G
ROUP BY \"type\" "
)

# Rename all thesis like to thesis
res <- dbSendQuery(
  con,
  "UPDATE records SET \"type\"= 'thesis' WHERE \"type\" ILIKE '%thesis%' AND
 repo = 'OpenGrey' "
)
thesis_rename <- dbGetRowsAffected(res)
dbClearResult(res)

res <- dbSendQuery(
  con,
  "DELETE FROM records WHERE \"type\" NOT ILIKE '%thesis%' AND \"type\" NOT
ILIKE '%Conference%' AND repo = 'OpenGrey' "
)
dropped_types_OG <- dbGetRowsAffected(res)
dbClearResult(res)

# Get all types from Scopus
Scopus_recs_per_type <- dbGetQuery(
  con,
  "SELECT DISTINCT \"type\", COUNT(*) FROM records WHERE repo = 'Scopus' GRO
UP BY \"type\" "
)

#all articles to -> article
res <- dbSendQuery(con,
                   "UPDATE records SET \"type\"= 'article' WHERE \"type\" IL
IKE '%article%' ")
```

```
article_rename <- dbGetRowsAffected(res)
dbClearResult(res)

#all thesis to -> thesis
res <- dbSendQuery(con,
                   "UPDATE records SET \"type\"= 'thesis' WHERE \"type\" ILI
KE '%thesis%' ")
thesis_rename <- dbGetRowsAffected(res)
dbClearResult(res)

#all conferences to -> conference paper
res <- dbSendQuery(
  con,
  "UPDATE records SET \"type\"= 'conference paper' WHERE \"type\" ILIKE '%co
nference%' "
)
conference_rename <- dbGetRowsAffected(res)
dbClearResult(res)

#rename publication-workingpaper to workingpaper
res <- dbSendQuery(
  con,
  "UPDATE records SET \"type\"= 'working paper' WHERE \"type\" ILIKE '%worki
ngpaper%' "
)
workingpaper_rename <- dbGetRowsAffected(res)
dbClearResult(res)

#Get all types
all_types <- dbGetQuery(con,
                        "SELECT DISTINCT \"type\", COUNT(*) FROM records GRO
UP BY \"type\" ")

#all english to -> eng
res <- dbSendQuery(con,
                   "UPDATE records SET lang = 'eng' WHERE lang ILIKE 'englis
h' OR lang ILIKE 'en'")
english_to_eng <- dbGetRowsAffected(res)
dbClearResult(res)

#delete all non english
res <- dbSendQuery(con,
                 "DELETE from records WHERE lang !='eng' AND lang IS NOT NULL")
dropped_langs <- dbGetRowsAffected(res)
dbClearResult(res)

#Get all languages
langs <- dbGetQuery(con,
                  "SELECT DISTINCT lang, COUNT(*) FROM records GROUP BY lang ")

#drop all that have no abstract
res <- dbSendQuery(con,
                   "DELETE FROM records WHERE abstract = 'n/a' OR abstract I
S NULL ")
null_abs <- dbGetRowsAffected(res)
dbClearResult(res)
```

```
#create tsv column and index
res <- dbSendQuery(con,
                   "ALTER TABLE records ADD COLUMN tsv tsvector")
dbClearResult(res)

res <- dbSendQuery(con,
                   "CREATE INDEX tsv_idx ON records USING gin(tsv);")
dbClearResult(res)

#fill tsv column

res <- dbSendQuery(con,
                   "UPDATE records SET tsv = to_tsvector(title) || to_tsvect
or(abstract)")
dbClearResult(res)
```

## 7.8. Further_cleaning.r

Πρόγραμμα επιπλέον εκκαθάρισης με βάση κατηγοριοποίηση γλώσσας και χρήση tsv.

```
library(ggplot2)
library(DBI)
library(RPostgres)
library(dplyr)

con <- dbConnect(
  Postgres(),
  host = "192.168.158.74",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

non_abst <- dbGetQuery(
  con,
  "SELECT * FROM records WHERE length(tsv) < 15
  "
)

res <- dbSendQuery(con,
                   "DELETE FROM records WHERE length(tsv) < 15 ")
short_abstracts <- dbGetRowsAffected(res)
dbClearResult(res)

# textcat trial on few records hence <16
res <- dbGetQuery(con,"select make_document(r.title, r.abstract)
from records r WHERE length(tsv) < 16; ")

res <- dbSendQuery(con,
                   "CREATE OR REPLACE FUNCTION textcat(abstract character)
                   RETURNS character AS '
                   library(textcat)
                   return(textcat(abstract)) '
                   LANGUAGE 'plr' STRICT;")

res <- dbGetQuery(con,"select id, textcat(r.abstract)
```

```
from records r WHERE length(tsv) < 16; ")

res <- dbSendQuery(con,"ALTER TABLE records ADD COLUMN textcat_res text")

res <- dbSendQuery(con,"UPDATE records r
    SET textcat_res = textcat(r.abstract)")

non_eng <- dbGetQuery(con,"SELECT * FROM records WHERE textcat_res != 'engli
sh'")

del_non_eng <- dbSendQuery(con,"DELETE FROM records WHERE textcat_res != 'en
glish'")
del_non_eng  <- dbGetRowsAffected(del_non_eng )
```

## 7.9. Queries.r

Πρόγραμμα αναζήτησης στη βάση μέσω Postgres Full Text Search.

```
library(ggplot2)
library(DBI)
library(RPostgres)
library(dplyr)

con <- dbConnect(
  Postgres(),
  host = "192.168.134.145",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

text_and_mining <- dbGetQuery(con,
                              "SELECT * FROM records WHERE tsv @@ to_tsquery
('text & mining')")
text_near_mining <- dbGetQuery(con,
                               "SELECT * FROM records WHERE tsv @@ to_tsquer
y('text <-> mining')")
ebm_and <- dbGetQuery(con,"SELECT * FROM records WHERE tsv @@ to_tsquery('ev
idence & based & medicine')")

ebm_near <- dbGetQuery(con,"SELECT * FROM records WHERE tsv @@ to_tsquery('e
vidence <-> based <-> medicine')")

missingand <- dbGetQuery(con,
                         "SELECT * FROM records WHERE repo = 'Scopus' AND id
 NOT IN (SELECT id FROM records WHERE tsv @@ to_tsquery('text & mining'))")
missingnear <- dbGetQuery(con,
                          "SELECT * FROM records WHERE repo = 'Scopus' AND i
d NOT IN (SELECT id FROM records WHERE tsv @@ to_tsquery('text <-> mining'))")
missingandebm <- dbGetQuery(con,
                         "SELECT * FROM records WHERE repo = 'Scopus' AND id
 NOT IN (SELECT id FROM records WHERE tsv @@ to_tsquery('evidence & based &
medicine'))")
missingnearebm <- dbGetQuery(con,
                          "SELECT * FROM records WHERE repo = 'Scopus' AND i
d NOT IN (SELECT id FROM records WHERE tsv @@ to_tsquery('evidence <-> based <-
> medicine'))")
```

```
#count queried per repo
text_and_mining %>% count(repo)
missingand %>% count
text_near_mining %>% count(repo)
missingnear %>% count
ebm_and %>% count(repo)
missingandebm %>% count
ebm_near %>% count(repo)
missingnearebm %>% count

keywordsquery <- dbGetQuery(con,
                            "SELECT * FROM records WHERE tsv @@ to_tsquery('
(text | literature) & mining')")
keywordsquery %>% count(repo)

test <- dbGetQuery(con,
                   "SELECT * FROM records WHERE tsv @@ to_tsquery('(
                            ((text | literature) <-> mining) | (mining <-
> (text | literature)) | ((knowledge | information)<-> extraction)
                   ) & (
                   (evidence <-> ( based | informed  )<-> medicine))')")
test %>% count(repo)

keywordsquery2 <- dbGetQuery(con,
                            "SELECT * FROM records WHERE tsv @@ to_tsquery('
text & mining & evidence & based & medicine')")
keywordsquery2 %>% count(repo)
```

## 7.10. LSA.r

Πρόγραμμα υλοποίησης LSA.

```
library(tm)
library(LSAfun)
library(DBI)
library(RPostgres)
library(RSpectra)
library(dplyr)

con <- dbConnect(
  Postgres(),
  host = "192.168.134.145",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

#Set the parameters for minimum global frequency and LSA semantic space dime
ntions
global_freq = 3
lsa_space_dim = 25

#Load all records and relevant fields into environment
res <- dbGetQuery(con,"SELECT id, abstract,title,repo FROM records")

#Form corpus, load relevant fields as metadata
```

```
raw_corpus <- VCorpus(VectorSource(res$abstract), readerControl=list(languag
e='en'))

ids <- paste0('id',seq(73200))
i <- 0
raw_corpus = tm_map(raw_corpus, function(x) {
  i <<- i +1
  meta(x, "id") <- res[i,1]
  meta(x, "heading") <- res[i,3]
  meta(x, "origin") <- res[i,4]
  x
})

remove_nonletter <- function(text) { return(gsub('[^a-z\\s]+', ' ', text))}

p_corpus <- tm_map(raw_corpus, content_transformer(tolower))
p_corpus <- tm_map(p_corpus, content_transformer(removeWords), tm::stopwords
('en'))
p_corpus <- tm_map(p_corpus, content_transformer(remove_nonletter))
p_corpus <- tm_map(p_corpus, stemDocument)
rm(raw_corpus)
print('creating tdm')
tdm <- TermDocumentMatrix(p_corpus, control = list(bounds = list(global = c(
global_freq, Inf))))
sparse_tdm <- Matrix::sparseMatrix(i = tdm$i, j = tdm$j, x = tdm$v, dims = c
(tdm$nrow,
                                                                  tdm$ncol))
dimnames(sparse_tdm) <- dimnames(tdm)

doc_count <- dim(sparse_tdm)[[2]]
log_doc_count <- log2(doc_count)

weighted_tdm <- sparse_tdm
#apply local log weight
local_weight = function(x) log2(x + 1)
weighted_tdm@x <- vapply(sparse_tdm@x, local_weight, numeric(1))
#calculate global frequency per term
gf <- Matrix::rowSums(sparse_tdm)
names(gf) <- dimnames(sparse_tdm)$Terms

#Function to calculate entropy per term 1+(>this<)
partial_entropy <- function(tf, gf) {
  p <- tf / gf
  return((p * log2(p)) / log_doc_count)
}
#Calculate global entropy weight per term
word_entropy <- numeric(dim(sparse_tdm)[[1]])
names(word_entropy) <- dimnames(sparse_tdm)$Terms
for(i in 1:dim(sparse_tdm)[[1]]){
  word_row <- sparse_tdm[i,]
  non_zero_frequencies <- word_row[which(word_row>0)]
  word_entropy[i] <- 1.0 + sum(mapply(partial_entropy, non_zero_frequencies,
 gf=gf[i]))
}
#Apply calculated weight to TDM
sweep_sparse <- function(x, margin, stats, fun = "*") {
```

```
  f <- match.fun(fun)
  if (margin == 1) {
    idx <- x@i + 1
  } else {
    idx <- x@j + 1
  }
  x@x <- f(x@x, stats[idx])
  return(x)
}
print('applying word entropy')
weighted_tdm <- sweep_sparse(weighted_tdm, 1, word_entropy)

print('constructing semantic space')
space <- svds(weighted_tdm, lsa_space_dim)
su_mat <- space$d * space$u
svt_mat <- space$d * Matrix::t(space$v)

dimnames(su_mat) <- list(dimnames(weighted_tdm)[[1]], 1:lsa_space_dim)
dimnames(svt_mat) <- list(1:lsa_space_dim, dimnames(weighted_tdm)[[2]])
print('done!')
```

## 7.11. LSA_queries.r

Πρόγραμμα αναζήτησης μέσω LSA.

```
library(cluster)
library(tm)
library(LSAfun)
library(DBI)
library(RPostgres)
library(RSpectra)
library(dplyr)
library(tibble)
rgl.printRglwidget = TRUE

con <- dbConnect(
  Postgres(),
  host = "192.168.134.145",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

plot_neighbors("text mining",connect.lines="all",col="rainbow",n=20,tvectors
= su_mat)
# ------------------------ text mining query -----------------------------
text_mining_query <- 'Text mining'
text_mining_query <- tolower(text_mining_query)
text_mining_query <- removeWords(text_mining_query, tm::stopwords('en'))
text_mining_query <- remove_nonletter(text_mining_query)
text_mining_query <- stemDocument(PlainTextDocument(text_mining_query))
text_mining_query <- termFreq(text_mining_query)
text_mining_query <- vapply(text_mining_query, function(x) log2(x+.00001), n
umeric(1))
text_mining_query <- mapply(function(x, y) x*y, text_mining_query, word_entr
opy[names(text_mining_query)])
text_mining_query <- colSums(text_mining_query * su_mat[names(text_mining_qu
ery),])
```

```
neighbors(text_mining_query, 50, tvectors=su_mat)
plot_neighbors(text_mining_query,connect.lines="all",col="rainbow",n=20,tvec
tors= su_mat)
# ------------------------ ebm query -------------------------------------
ebm_query <- 'evidence based medicine doctor patient drug target biomedical
discovery'
ebm_query <- tolower(ebm_query)
ebm_query <- removeWords(ebm_query, tm::stopwords('en'))
ebm_query <- remove_nonletter(ebm_query)
ebm_query <- stemDocument(PlainTextDocument(ebm_query))
ebm_query <- termFreq(ebm_query)
ebm_query <- vapply(ebm_query, function(x) log2(x+.00001), numeric(1))
ebm_query <- mapply(function(x, y) x*y, ebm_query, word_entropy[names(ebm_qu
ery)])
ebm_query <- colSums(ebm_query * su_mat[names(ebm_query),])

neighbors(ebm_query, 50, tvectors=su_mat)

# ---------------------- custom query -----------------------------------
customquery <- 'text mining evidence based medicine'
customquery <- tolower(customquery)
customquery <- removeWords(customquery, tm::stopwords('en'))
customquery <- remove_nonletter(customquery)
customquery <- stemDocument(PlainTextDocument(customquery))
customquery <- termFreq(customquery)
customquery <- vapply(customquery, function(x) log2(x+1), numeric(1))
customquery <- mapply(function(x, y) x*y, customquery, word_entropy[names(cu
stomquery)])
# missingwords <- names(customquery[!(names(customquery) %in% rownames(su_ma
t))])
customquery <- colSums(customquery * su_mat[names(customquery),])
plot_neighbors(customquery, 50, tvectors=su_mat)

similardocs <- data.frame(neighbors(customquery, 2000, tvectors=Matrix::t(sv
t_mat)))
similardocs = similardocs %>% rename(cos_sim = neighbors.customquery..2000..
tvectors...Matrix..t.svt_mat.. )
similardocs$id = as.numeric(rownames(similardocs))

temp <- dbWriteTable(con, "temp", similardocs, row.names=FALSE, overwrite =
TRUE)
temp <- dbGetQuery(con,'SELECT abstract,cos_sim,title,repo, records.id FROM
records JOIN temp ON records.id = temp.id')
temp <- temp %>% arrange(desc(temp$cos_sim))

customquery <- temp %>% head(1000)

simplequery <- dbGetQuery(con,"SELECT abstract,title,id,repo FROM records WH
ERE tsv @@ to_tsquery('text & mining & evidence & based & medicine')")

matches_simpletext<- temp[temp$id %in% simplequery$id,]
scopus_in_query <- customquery %>% rownames_to_column('order') %>% filter(re
po == 'Scopus') %>% column_to_rownames('order')
res <- dbSendQuery(con,"DROP TABLE temp")
```

## 7.12. Reranking.r

Πρόγραμμα ανακατάταξης αποτελεσμάτων καθώς και δημιουργίας γραφημάτων.

```
library(ggplot2)
library(cluster)
library(tm)
library(LSAfun)
library(DBI)
library(RPostgres)
library(RSpectra)
library(dplyr)
library(tibble)
library(RColorBrewer)

con <- dbConnect(
  Postgres(),
  host = "192.168.134.145",
  dbname = "postgres",
  user = "postgres",
  password = "postgres"
)

data <- dbGetQuery(con,"SELECT * FROM \"100log\" ORDER BY cos_sim desc LIMIT
 1000")

ebm <- dbGetQuery(con,"SELECT l.id
FROM \"100log\" l LEFT OUTER JOIN records ON l.id = records.id WHERE tsv @@
to_tsquery('evidence<->based<->medicine')
")
txt <- dbGetQuery(con,"SELECT l.id
FROM \"100log\" l LEFT OUTER JOIN records ON l.id = records.id WHERE tsv @@
to_tsquery('text<->mining')
")

for (i in 1:nrow(data)) {
  data$score[i] = data$cos_sim[i] * 0.33
  if (any(data$id[i] == ebm$id)) {
    data$score[i] = data$score[i] + 0.33
  }
  if (any(data$id[i] == txt$id)) {
    data$score[i] = data$score[i] + 0.33
  }
}
data$x = as.numeric(rownames(data))

ggplot(data[1:100,], aes(x=x))+
  geom_tile(aes(fill=relevance, y=0), width=1, height=Inf, alpha=0.7) +
  geom_line(aes(y = cos_sim)) +
  coord_cartesian(ylim = c(0.6,0.9)) +
  labs(y = "Cosine Similarity", x = "# record")

data <- data%>%arrange(desc(score))
data$x = as.numeric(rownames(data))

ggplot(data[1:100,], aes(x=x))+
  geom_tile(aes(fill=ebm, y=0), width=1, height=Inf, alpha=0.7) +
```

```
  geom_line(aes(y = score)) +
  labs(y = "Score", x = "# record")
```