ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Συστήματος Πρότασης Περιοδικού

Νούλης Αριστείδης
ΑΕΜ:2390

Επιβλέπων καθηγητής
Γρηγόριος Τσουμάκας, Επίκουρος Καθηγητής

2 Ιουλίου 2018

Περίληψη

Η δημοσίευση μιας επιστημονικής εργασίας δε μπορεί να θεωρηθεί απλή και εύκολη υπόθεση. Ο ερευνητής εκτός από τις δυσκολίες που μπορεί να συναντήσει κατά τη συγγραφή, ενδέχεται να αντιμετωπίσει δυσκολίες και κατά την επιλογή του περιοδικού δημοσίευσης. Αν για κάποιο λόγο η εργασία του δε γίνει δεκτή από τα περιοδικά της αρχικής επιλογής ή είναι νέος ερευνητής σε ένα γνωστικό επίπεδο ή ακόμη δεν έχει κατατοπιστική εικόνα των διαθέσιμων και κατάλληλων περιοδικών για το είδος της εργασίας του, τότε θα χρειαστεί να μελετήσει αρκετά τους σκοπούς κάθε περιοδικού και να αποκτήσει μια κατατοπιστική εικόνα του είδους των άρθρων τα οποία δημοσιεύει το κάθε περιοδικό, για να πραγματοποιήσει την καλύτερη επιλογή. Η ανάγκη για την ανάπτυξη ενός συστήματος που θα ελαχιστοποιήσει στο μέγιστο τον χρόνο αυτό και θα αποτελέσει ένα χρήσιμο εργαλείο για τους ερευνητές, αποτελεί τον κινητήριο μοχλό της παρούσας πτυχιακής. Ειδικότερα, θα αναπτύξουμε μοντέλο το οποίο χρησιμοποιώντας μετρικές ομοιότητας μεταξύ κειμένων και συλλογών κειμένων και δοθέντος ενός τίτλου και μιας περίληψης θα εμφανίζει τα πιο σχετικά περιοδικά.

Abstract

Publication of a scientific paper can not be considered a simple and easy task. The researcher, in addition to the difficulties he may encounter during the writing, may also have difficulty in choosing a journal for the publication of his scientific paper. If his scientific work is not accepted by the journals of his first choice or is a young researcher in a specific scientific field or has no strong knowledge of the available and suitable journals for his paper, then he will need to study the purposes of each journal and get sufficient information about the articles that each journal publishes, so as to make the best choice. The need of developing a system that will minimize this search time and will be a useful tool for researchers, is the main reason that we are writing this thesis. In particular, we will develop a model which using similarities between texts and text collections, and given a title and an abstract, will provide to user the most relevant journals for his paper.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή προβλήματος

Ένα συχνό πρόβλημα το οποίο αντιμετωπίζουν οι ερευνητές, αφού έχουν ολοκληρώσει την έρευνα ή εργασία τους, είναι η επιλογή κατάλληλου επιστημονικού περιοδικού ή συνεδρίου για να την δημοσιεύσουν.

Αυτή η διαδικασία μπορεί να χαρακτηριστεί δύσκολη και χρονοβόρα κυρίως σε περιπτώσει όπου έχουμε έναν νέο ερευνητή ή συγγραφή εργασίας σε καινούργια επιστημονική περιοχή, καθώς στις μέρες μας οι επιλογές είναι πολλές.

Αν εξαιρέσουμε την περίπτωση όπου ο φορέας δημοσίευσης έχει ήδη προαποφασιστεί λόγω προσωπικών προτιμήσεων ή συνθηκών (παραδείγματος χάριν αν η εργασία έχει συγγραφεί στο πλαίσιο πρόσκλησης ειδικού τεύχους (special issue) συγκεκριμένου περιοδικού), ο ερευνητής χρειάζεται να υπολογίσει πολλούς παράγοντες ώστε να πάρει την βέλτιστη απόφαση με γνώμονα πάντα την μέγιστη δυνατή απήχηση της δημοσίευσης της εργασίας του.

Αυτοί μπορεί να είναι είναι πολλοί, όπως η σχετικότητα μεταξύ της συγκεκριμένης εργασίας και του πεδίου στο οποίο ειδικεύεται το κάθε περιοδικό ή συνέδριο, αν το εκάστοτε κοινό είναι αυτό στο οποίο στοχεύει και απευθύνεται η εργασία, και αν τα κίνητρα και οι σκοποί συγγραφής του ερευνητή συμφωνούν με αυτούς του χώρου δημοσίευσης. Ακόμη αρκετά σημαντικά είναι η φήμη του περιοδικού, καθώς ο συντελεστής επιρροής (Impact Factor) ενός περιοδικού που αποτελεί ένα μέτρο της φήμης του, και το κύρος των συγγραφέων που δημοσιεύουν στο περιοδικό και το επίπεδο των ερευνών τους, καθώς σε συνδυασμό αποτελούν χρήσιμα μέτρα αξιολόγησης και των ίδιων των ερευνητών.

Για να μπορέσει ο ερευνητής να εξετάσει αυτούς τους παράγοντες θα πρέπει να μελετήσει το ιστορικό, τις προθέσεις και τους στόχους κάθε περιοδικού ή συνεδρίου, να μελετήσει ήδη υπάρχουσες δημοσιεύσεις για να αξιολογήσει την ομοιότητα και τον προσανατολισμό τους σε σχέση με την δική του και επίσης να καταλάβει αν η εργασία του θα βρει την ανταπόκριση που επιθυμεί { [1] [2] [3] }.

Έτσι γεννιέται και η ανάγκη της ύπαρξης ενός συστήματος πρότασης κατάλληλου περιοδικού ή συνεδρίου, η σωστή και αποτελεσματική λειτουργία του οποίου

θα εξοικονομήσει χρόνο στους ερευνητές και θα οδηγήσει σε περισσότερο στοχευμένες δημοσιεύσεις με αποτέλεσμα μεγαλύτερο αναγνωστικό κοινό και αναγνώρισης του έργου του ερευνητή.

1.2 Υπάρχουσες λύσεις

Μέχρι σήμερα υπάρχουν συστήματα που κυρίως προτείνουν περιοδικά για δημοσίευση και ελάχιστα προτείνουν και συνέδρια. Ένας σεβαστός αριθμός εξ αυτών λειτουργεί υπό την σκέπη συγκεκριμένων εκδοτικών οίκων, κάτι το οποίο προκαθορίζει το πεδίο αναζήτησης και επιβάλει κάποιος περιορισμούς. Επίσης, η μικρή κάλυψη συνεδρίων εντείνει το πρόβλημα των ερευνητών που στοχεύουν σε αυτό το είδος της δημοσίευσης για λόγους όπως η ανάδραση με το κοινό, επικαιρότητας του θέματος και διαφόρων άλλων.

Η παρούσα εργασία αποσκοπεί στο να επιτύχει την δημιουργία ενός αποτελεσματικού συστήματος συστάσεων σχετικών περιοδικών, με μελλοντική δυνατότητα κάλυψης και του κενό των συνεδρίων και να προσφέρει μια λύση ανεξάρτητη από εκδοτικούς οίκους. Η βιωσιμότητα και εμπορική αξιοποίηση του συστήματος μας μπορεί να επιτευχθεί είτε με την ενσωμάτωση σχετικών διαφημίσεων είτε με παροχή ειδικών εκδόσεων σε εκδοτικούς οίκους απευθείας ή μέσω ενδιάμεσων φορέων, διατηρώντας πάντα τον καθολικό και ανεξάρτητο χαρακτήρα του.

1.3 Επισκόπηση κεφαλαίων

Το υπόλοιπο μέρος της συγκεκριμένης εργασίας δομείται ως εξής:

- Το δεύτερο κεφάλαιο αφορά το θεωρητικό υπόβαθρο της συγκεκριμένης επιστημονικής περιοχής και την λεπτομερή ανάλυση των διαφόρων μετρικών ομοιότητας καθώς και της μεταξύ τους σχέσης.
- Το τρίτο κεφάλαιο αναφέρεται σε αντίστοιχα συστήματα που ήδη υπάρχουν, αναλύοντας την λειτουργία τους και τις δυνατότητες τους αλλά και στη σύντομη παρουσίαση παρόμοιων εργασιών.
- Το τέταρτο κεφάλαιο επικεντρώνεται στην ανάλυση της δικής μας υλοποίησης, καλύπτοντας τον τρόπο σχεδίασης και πειραματισμού, των προβλημάτων που προέκυψαν και τις μεθόδους αντιμετώπισης τους.
- Το πέμπτο και τελευταίο κεφάλαιο συνοψίζει τα συμπεράσματα της μέχρι πρότινος υλοποίησης και τις πιθανές μελλοντικές επεκτάσεις και βελτιώσεις.

Κεφάλαιο 2

Υπόβαθρο

2.1 Ευρετηριοποίηση συλλογής εγγράφων με ανεστραμμένο κατάλογο

Η ευρετηριοποίηση (indexing) είναι μια τεχνική η οποία χρησιμοποιείται σε περιπτώσεις όπου είναι αναγκαία η αποθήκευση του συνόλου των εγγράφων σε μορφή που θα επιτρέπει την άμεση αναζήτηση και ανάκτηση των σχετικών εγγράφων. Αυτοί οι κατάλογοι έχουν ουσιαστικά την δυνατότητα να απορρίπτουν δεδομένα τα οποία δεν έχουν καμιά χρησιμότητα στην εκάστοτε περίπτωση. Δυνατότητα που τους κάνει αποδοτικότερους σε σχέση με τεχνικές σειριακής αναζήτησης.

Ο ανεστραμμένος κατάλογος (inverted index) κάνει πράξη την τεχνική της ευρετηριοποίησης, αποθηκεύοντας κάθε λέξη ενός εγγράφου μαζί με μια λίστα με όλα τα έγγραφα της συλλογής στα οποία εμφανίζεται. Σαν αποτέλεσμα, έχουμε την άμεση πρόσβαση στον αριθμό των εγγράφων που εμφανίζεται η λέξη αλλά και ποια είναι αυτά.

Τα δυο βασικά τμήματα από τα οποία αποτελείται ο ανεστραμμένος κατάλογος είναι το λεξικό όρων, που απαρτίζεται από τους διαφορετικούς όρους της συλλογής και τη λίστα εμφανίσεων των όρων, που αναφέρουν για κάθε όρο σε ποια έγγραφα εμφανίζεται.

Όσον αφορά τη δημιουργία του καταλόγου μπορούμε να ξεχωρίσουμε τρεις βασικές μεθόδους. Η πρώτη είναι η αντιστροφή της κύριας μνήμης, όπου για κάθε όρο δημιουργούμε τη λίστα με τα έγγραφα στα οποία εμφανίζεται, αποθηκεύοντας το κατάλογο στη κύρια μνήμη. Το μεγάλο μειονέκτημα της παραπάνω περίπτωσης είναι ότι για μεγάλες συλλογές, η κύρια μνήμη έχει μεγάλες πιθανότητες να μη είναι επαρκής. Αυτό όμως το πρόβλημα έρχονται να επιλύσουν οι δύο άλλες μέθοδοι δημιουργίας καταλόγου, η αντιστροφή με ταξινόμηση και η αντιστροφή με συγχώνευση, καθώς διαβάζουν μια φορά τη συλλογή εγγράφων και αποθηκεύουν το κατάλογο που δημιουργούν σε ένα αρχείο.

Στην περίπτωση της ταξινόμησης, αφού πραγματοποιείται η ανάγνωση της συλλογής, δημιουργείται ένα ενδιάμεσο αρχείο που περιέχει εγγραφές της μορφής $(t, d, f_{t,d})$ όπου το t συμβολίζει τον όρο, το d το έγγραφο και το $(f_{t,d})$ τη συχνότητα

εμφάνισης του όρου στο έγγραφο. Στην αρχή το αρχείο είναι ταξινομημένο ως προς τα έγγραφα (λόγω της ανάγνωσης της συλλογής). Στη συνέχεια όμως πραγματοποιείται ταξινόμηση ως προς τους όρους και τελικά συγχώνευση των επιμέρους λιστών στη κύρια μνήμη, αποθηκεύοντας τελικά το αποτέλεσμα στο δίσκο.

Στην περίπτωση της αντιστροφής με συγχώνευση, όπου η διαδικασία ξεκινάει να λαμβάνει χώρα στη κύρια μνήμη και κάθε φορά που ο χώρος δεν είναι επαρκής, αποθηκεύεται το υπάρχον κομμάτι του καταλόγου στο δίσκο. Με την ολοκλήρωση της ανάγνωσης της συλλογής, τα τμήματα που έχουν αποθηκευτεί στο δίσκο, συγχωνεύονται ώστε να παραχθεί ο τελικός συνολικός ανεστραμμένος κατάλογος.

Εξίσου σημαντικού ενδιαφέροντος είναι και οι τεχνικές συντήρησης ενός ανεστραμμένου καταλόγου. Μία από αυτές είναι η αναδόμηση του καταλόγου, όπου έχοντας πια συλλέξει νέα έγγραφα για εισαγωγή, πραγματοποιείται μία εκ νέου δημιουργία του καταλόγου, διαγράφοντας το προηγούμενο.

Επίσης, υπάρχει η ενημέρωση με συγχώνευση, όπου δημιουργείται ένας προσωρινός κατάλογος για τα νέα έγγραφα. Αυτός διατηρείται στην κύρια μνήμη όσο υπάρχει χρόνος, και μετά εφαρμόζεται η τεχνική της συγχώνευσης μεταξύ του υπάρχοντος καταλόγου που βρίσκεται στο δίσκο με τον προσωρινό της κύριας μνήμης.

Τέλος έχουμε τη σταδιακή ενημέρωση καταλόγου. Στη συγκεκριμένη περίπτωση, για την ενημέρωση της λίστας εμφανίσεων ενός όρου της συλλογής, η λίστα διαβάζεται από το δίσκο στην κύρια μνήμη, ενημερώνεται και στη συνέχεια αποθηκεύεται πάλι στο δίσκο.

2.2 Μετρική ομοιότητας TF-IDF

Η μετρική tf-idf αποτελεί ακρωνύμιο του term frequency - inverse document frequency, δηλαδή συχνότητα όρου - αντίστροφη συχνότητα εγγράφων. Συγκεκριμένα, το βάρος tf-idf είναι ένα στατιστικό βάρος που χρησιμοποιείται συχνά στην ανάκτηση πληροφοριών και την εξόρυξη κειμένου. Χρησιμοποιείται για να αξιολογήσει πόσο σημαντική είναι μια λέξη σε ένα έγγραφο μιας συλλογής .

Η σημασία αυξάνεται ανάλογα με τον αριθμό των εμφανίσεων μιας λέξης σε ένα έγγραφο, αλλά αντισταθμίζεται από τη συχνότητα εμφάνισης της συγκεκριμένης λέξης στο σύνολο των εγγράφων. Διάφορες παραλλαγές της μετρικής ομοιότητας tf-idf χρησιμοποιούνται συχνά από τις μηχανές αναζήτησης ως κεντρικό εργαλείο βαθμολόγησης και ταξινόμησης της σχετικότητας ενός εγγράφου με βάση το ερώτημα του χρήστη.

Πιο συγκεκριμένα το βάρος tf-idf αποτελείται από δύο όρους: το $tf_{t,d}$ και το idf_t .

Σαν $tf_{t,d}$ ορίζεται η συχνότητα του όρου t στο έγγραφο d. Δεδομένου ότι κάθε έγγραφο έχει διαφορετικό μήκος, είναι πιθανό ότι ένας όρος θα εμφανιζόταν περισσότερες φορές σε έγγραφα με μεγάλο αριθμό όρων από ότι σε αυτά με μικρότερο. Έτσι, ο όρος συχνότητα συχνά διαιρείται με το μήκος του εγγράφου (που είναι ο συνολικός αριθμός όρων στο έγγραφο) ως ένας τρόπος ομαλοποίησης. Λόγω αυτού

προκύπτει ο παρακάτω τύπος:

$$tf_{t,d} = \frac{f_{t,d}}{N_d} \tag{2.1}$$

όπου το f_t ισούται με τον αριθμό εμφάνισης του όρου στο έγγραφο και το N_d με το συνολικό αριθμό όρων στο έγγραφο.

Σαν idf_t ορίζεται η αντίστροφη συχνότητα εγγράφων, η οποία μετρά πόσο σημαντικός είναι ένας όρος. Κατά τον υπολογισμό $tf_{t,d}$, όλοι οι όροι θεωρούνται εξίσου σημαντικοί. Ωστόσο, είναι γνωστό ότι ορισμένοι όροι, όπως το ”είναι”, ”από” και ”ότι”, μπορεί να εμφανίζονται πολλές φορές αλλά έχουν μικρή σημασία. Επομένως, πρέπει να σταθμίσουμε τους συχνούς όρους, ενώ να αυξήσουμε τη σημασία των σπάνιων. Έτσι έχουμε το τύπο:

$$idf_t = 1 + log(\frac{N}{n_t}) \tag{2.2}$$

όπου N είναι ο συνολικός αριθμός εγγράφων και n_t ο αριθμός εγγράφων με τον όρο t σε αυτό.

Το συνολικό βάρος tf-idf προκύπτει από τον πολλαπλασιασμό των δύο επιμέρους όρων:

$$w_{t,d} = tf_{t,d} * idf_t \tag{2.3}$$

Ο παραπάνω τύπος μπορούμε να πούμε ότι δίνει τη γενική ερμηνεία της μετρικής tf-idf. Αναφέραμε προηγουμένως όμως ότι χρησιμοποιείται και με διάφορες παραλλαγές. Μια τέτοια είναι σε συνδυασμό με τον ανεστραμμένο κατάλογο τόσο για υπολογισμό ομοιότητας ανάμεσα στα έγγραφα της συλλογής αλλά και ανάμεσα σε ένα ερώτημα και τα έγγραφα της συλλογής.

2.3 BM25

Στην ανάκτηση πληροφορίας, το Okapi BM25 ή BM25 (BM από το Best Matching)είναι μια συνάρτηση βαθμολόγησης συνάφειας που χρησιμοποιείται από τις μηχανές αναζήτησης για την ταξινόμηση των συνυφασμένων εγγράφων ανάλογα με τη συνάφεια τους με ένα δεδομένο ερώτημα αναζήτησης.

Το όνομα της ουσιαστικά είναι BM25, αλλά επειδή το πρώτο σύστημα ανάκτησης πληροφορίας που το υλοποίησε ήταν το Okapi, συχνά αναφέρεται και Okapi BM25.

Βασίζεται στο πιθανοκρατικό μοντέλο ανάκτησης και υπάρχουν διάφορες παραλλαγές της. Μία από τις επικρατέστερες είναι η παρακάτω:

Δοθείσας μιας επερώτησης Q, που περιλαμβάνει τους όρους q_n,....,q_n, η βαθμολογία με βάση το BM25 για κάποιο κείμενο D είναι

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_{i,D}) + k_1(1-b+b\frac{|D|}{avgdl}} \tag{2.4}$$

όπου $f(q_i, D)$ είναι το tf (term frequency) του όρου q_i στο κείμενο D, $|D|$ είναι ο αριθμός των εμφανίσεων των λέξεων στο κείμενο D, και avgdl είναι το μέσο μήκος ενός κειμένου στη συλλογή μας.
Τα k_1 και b είναι ελεύθερες μεταβλητές (συνήθως $k_1 = [1.2, 2.0]$ και $b = 0.75$). $IDF(q_i)$ είναι το Inverse Document Frequensy του όρου q_i, το οποίο συνήθως υπολογίζεται ως:

$$IDF(q_i) = log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}, \tag{2.5}$$

όπου N είναι ο συνολικός αριθμός κειμένων της συλλογής και $n(q_i)$ είναι ο αριθμός των κειμένων που περιλαμβάνου τον όρο q_i. [4]

2.4 Word2Vec

Το Word2Vec είναι ένα μοντέλο που παράγει ενσωματώσεις λέξεων (word embeddings). Συγκεκριμένα, το Word2Vec δέχεται ως είσοδο ένα σύνολο εγγράφων και έχει σαν έξοδο ένα σύνολο διανυσμάτων. Αυτά είναι διανύσματα χαρακτηριστικών για τις λέξεις που ανήκουν στο σύνολο του λεξιλογίου που προκύπτει από το σώμα των εγγράφων της εισόδου.

Με τη χρήση του Word2Vec μπορεί είναι να διατηρηθεί η σημασιολογική πληροφορία των λέξεων. Γι αυτό το λόγο, λέξεις που συνδέονται σημασιολογικά και έχουν πιο κοινές έννοιες θα έχουν όμοια διανύσματα.

Το Word2Vec είναι ένα προγνωστικό μοντέλο με. Χρησιμοποιώντας το καταφέρνουμε την ομαδοποίηση των διανυσμάτων παρόμοιων λέξεων μαζί στο διανυσματικό χώρο. Δηλαδή ανιχνεύουμε μαθηματικές ομοιότητες. Το Word2vec δημιουργεί διανύσματα που είναι κατανεμημένες αριθμητικές αναπαραστάσεις των λεξικών χαρακτηριστικών, όπως είναι το περιεχόμενο και η σημασία κάθε μεμονωμένης λέξης. Αυτό επιτυγχάνεται χωρίς την ανθρώπινη παρέμβαση κατά τη λειτουργία του μοντέλου.

Το Word2Vec μπορεί να εφαρμοστεί με τα ακόλουθα 2 μοντέλα το Continuous Bag-Of-Words (CBOW) και το Skip-Gram.

Το CBOW είναι ένα νευρωνικό δίκτυο που εκπαιδεύεται για να προβλέψει ποια λέξη ταιριάζει σε ένα κενό σε μια πρόταση. Για παράδειγμα, δεδομένης της μερικής φράσης ”μια γάτα __ στο”,το νευρωνικό δίκτυο προβλέπει ότι το ”κάθεται” έχει μεγάλη πιθανότητα να γεμίσει το κενό. Η σειρά των λέξεων δεν είναι σημαντική: δίνοντας τέσσερις λέξεις, το νευρικό δίκτυο προβλέπει τη λέξη που βρίσκεται στη μέση. Το επίπεδο ενσωμάτωσης του νευρικού δικτύου χρησιμοποιείται για να αντιπροσωπεύει γενικά τις λέξεις.

Από την άλλη πλευρά, το μοντέλο skip-gram λειτουργεί αντίστροφα. Με δεδομένη μια μεσαία λέξη, προβλέπει ποιες λέξεις την περιβάλλουν. Φυσικά με τον όρο προβλέπει, εννοούμε ότι εξάγει μια πιθανότητα για κάθε λέξη που ανήκει στο λεξιλόγιο μας, και επιλέγει αυτή με την μεγαλύτερη πιθανότητα.

Αυτές οι ενέργειες αποσκοπούν στο να αναγκάσουν το νευρωνικό δίκτυο να δημιουργήσει ενσωματώσεις που αντικατοπτρίζουν τη σχέση μεταξύ των λέξεων, και συνεπώς να τους δώσουν νόημα.

Τέλος, τα διανύσματα ενσωμάτωσης που δημιουργούνται χρησιμοποιώντας τον αλγόριθμο Word2Vec έχουν πολλά πλεονεκτήματα σε σύγκριση με προηγούμενους αλγορίθμους όπως η λανθάνουσα σημασιολογική ανάλυση (Latent Semantic Analysis, LSA) η οποία οδηγεί σε βελτίωση της αποτελεσματικότητας των συστημάτων.

2.5 Glove

Το GloVe είναι ένας αλγόριθμος μάθησης χωρίς επίβλεψη, ο οποίος χρησιμοποιείται με σκοπό την απόκτηση διανυσματικών αναπαραστάσεων για τις λέξεις ενός συνόλου. Η εκπαίδευση πραγματοποιείται σε συγκεντρωτικά στατιστικά στοιχεία συν-εμφάνισης λέξεων από ένα σώμα και οι προκύπτουσες αναπαραστάσεις παρουσιάζουν ενδιαφέρουσες γραμμικές υποσυνθέσεις του διανυσματικού χώρου των λέξεων.

Η προσέγγιση του GloVe είναι ότι αντί να εξάγουμε τις ενσωματώσεις από ένα νευρωνικό δίκτυο που έχει σχεδιαστεί για να εκτελέσει μια εργασία όπως η πρόβλεψη γειτονικών λέξεων (Word2Vec), τις βελτιώνουμε κατευθείαν έτσι ώστε το εσωτερικό γινόμενο δύο διανυσμάτων από λέξεις να ισούται με το λογάριθμο του αριθμού των φορών που θα εμφανιστούν οι δύο λέξεις η μία κοντά στην άλλη (μέσα σε 5 λέξεις για παράδειγμα).

Για παράδειγμα, αν η λέξη σκύλος και γάτα εμφανίζονται κοντά η μία στην άλλη 10 φορές σε ένα μια συλλογή, τότε το εσωτερικό γινόμενο των διανυσμάτων των δύο λέξεων θα ισούται με το λογάριθμο του 10. Αυτό ωθεί τους πίνακες διανυσμάτων να κωδικοποιήσουν κατά κάποιο τρόπο την κατανομή συχνότητας των λέξεων που εμφανίζονται κοντά τους.

Οι δημιουργοί του Glove, ήθελαν να διατηρήσουν την αναλογία, χωρίς να χάσουν τη διαφάνεια και την ακριβή έννοια των λεξικών στατιστικών στοιχείων του σώματος (συλλογής), κάτι το οποίο θεωρούσαν ως θεμελιώδη αρχή του προβλήματος. Υπέθεσαν ότι βρίσκοντας ένα τρόπο να διατηρήσουν την αναλογία μέσω γραμμικής αριθμητικής, η οποία χρησιμοποιεί μόνο τις θεμελιώδεις στατιστικές ιδιότητες του σώματος ως είσοδο, θα παρουσίαζαν μια βελτίωση τόσο στην ακρίβεια όσο και στην ερμηνεία. [5]

2.6 Apache Solr

2.6.1 Εισαγωγή

Το Solr αποτελεί μια από τις πιο διαδεδομένες πλατφόρμες αναζήτησης ανοιχτού κώδικα, βασισμένη στο Apache Lucene και βρίσκεται υπό την αιγίδα του

Apache Software Foundation. Χαρακτηρίζεται ως μια πλατφόρμα εξαιρετικά αξιόπιστη, κλιμακωτή και ανεκτική σε λάθη. Προσφέρει μια πληθώρα χαρακτηριστικών όπως κατανεμημένη ευρετηριοποίηση, προχωρημένες δυνατότητες αναζήτησης κειμένου, αυτοματοποιημένη ανάκαμψη και ανάκτηση από συνθήκες σφαλμάτων,αλληλεπίδραση με βάσεις δεδομένων και τέλος κεντρικοποιημένη διαχείριση. [6]

Η πλατφόρμα του Solr είναι γραμμένη σε Java και θεωρείται από μόνη της ένα διακομιστής αναζήτηση πλήρους κειμένου. Χρησιμοποιεί την βιβλιοθήκη αναζήτησης Lucene για την καταχώρηση και την αναζήτηση κειμένου, μέσω REST - APIS καθίσταται δυνατή η χρησιμοποίηση της από άλλες γλώσσες προγραμματισμού.Επίσης υποστηρίζει την μαζική εισαγωγή εγγράφων της μορφής XML, CSV,JSON και την προσθήκη επεκτάσεων με σκοπό την προσαρμογή σε πολλούς τύπους εφαρμογών αλλά και επιπλέον παραμετροποίησης. [7]

Το κυριότερο πλεονέκτημα της Solr σε σχέση με τις κλασικές μεθόδους αναζήτησης είναι το ότι μπορεί να επιστρέψει αποτελέσματα στις αναζητήσεις εξαιρετικά γρήγορα, και αυτό συμβαίνει καθώς αντί να ψάξει κατευθείαν στο κείμενο, αναζητά ένα ευρετήριο.

Για να γίνει περισσότερο κατανοητό, χρειάζεται απλά να σκεφτούμε ότι η λειτουργία είναι ίδια με της ανάκτησης σελίδων σε ένα βιβλίο που σχετίζονται με μια λέξη-κλειδί, σαρώνοντας το ευρετήριο στο πίσω μέρος του βιβλίου, σε αντίθεση με την αναζήτηση κάθε λέξης σε κάθε σελίδα του βιβλίου.

Αυτός ο τύπος ευρετηρίου ονομάζεται ανεστραμμένος κατάλογος, και είναι ίδιος με αυτόν που περιγράψαμε στην ενότητα της Ευρετηριοποίησης Συλλογής Εγγράφων Με Ανεστραμμένο Κατάλογο. Ουσιαστικά αντιστρέφει μια δομή δεδομένων που είχε τη κλασσική βάση της σελίδας (σελίδα -> λέξεις), σε μια δομή δεδομένων με βάση αυτή τη φορά τις λέξεις-κλειδιά (λέξεις -> σελίδες). [8]

2.6.2 Δομικά στοιχεία

Για να αποκτήσουμε μια βαθύτερη κατανόηση της λειτουργίας του Solr, χρειάζεται επίσης να αναφέρουμε τον τρόπο με τον οποίο διασπά και επεξεργάζεται τα δεδομένα.

Υπάρχουν τρεις βασικές έννοιες, οι οποίες είναι: οι αναλυτές (analyzers), οι διασπαστές(tokenizers) και τα φίλτρα(filters).

Οι αναλυτές πεδίων χρησιμοποιούνται τόσο στην εισαγωγή των δεδομένων, κατά την ευρετηριοποίηση ενός εγγράφου, όσο και κατά την ώρα των αναζητήσεων. Ένας αναλυτής εξετάζει το κείμενο των πεδίων και δημιουργεί μια ροή δεδομένων. Οι αναλυτές μπορεί να είναι μία κλάση ή μπορεί να αποτελούνται από μια σειρά κατηγοριών διασπαστών και φίλτρων.

Οι Διασπαστές χωρίζουν τα δεδομένα του πεδίου σε λεξικές μονάδες.

Τα φίλτρα εξετάζουν μια ροή διασπασμένων δεδομένων και τα διατηρούν, τα μετατρέπουν ή τα απορρίπτουν ή ακόμη δημιουργούν νέα.

Οι διασπαστές και τα φίλτρα μπορούν να συνδυαστούν για να σχηματίσουν σωληνώσεις (pipelineσ) ή αλυσίδες (chains), όπου η έξοδος του ενός είναι η είσοδος

του επόμενου. Μια τέτοια ακολουθία διασπαστών και φίλτρων ονομάζεται αναλυτής και η προκύπτουσα έξοδος ενός αναλυτή χρησιμοποιείται για την αντιστοίχιση των αποτελεσμάτων των ερωτημάτων ή τη δημιουργία δεικτών. [9]

Όσον αφορά την χρήση του Solr στην περίπτωση μας αξίζει να αναφέρουμε κάποια βασικά βήματα και λειτουργίες που χρησιμοποιήσαμε γενικά στους πειραματισμούς μας και στην δόμηση του συστήματος.

Για να μπορέσει κάποιος να αξιοποιήσει και να δουλέψει με το Solr, πρέπει πρώτα να δημιουργήσει μια συλλογή (collection) όπου εκεί θα αποθηκευτεί το σύνολο των εγγράφων με το οποίο θέλει να πειραματιστεί.

Στη συνέχεια μέσω του αρχείου managed-schema της συλλογής, με την οποία δουλεύει ο χρήστης στην εκάστοτε περίπτωση, μπορεί να ρυθμίσει τι είδους φίλτρα, διασπαστές και αναλυτές θα χρησιμοποιεί κάθε πεδίο και κατά την διάρκεια της εισαγωγής - ευρετηριοποίησης (index) αλλά και κατά τη στιγμή της ερώτησης - αναζήτησης (query).

Επίσης, οφείλουμε να τονίσουμε το γεγονός ότι στη Solr διατηρείται η σειρά εκτέλεσης μεταξύ των φίλτρων, κάτι το οποίο σημαίνει ότι αυτά εκτελούνται με την σειρά με την οποία αναγράφονται στο managed-schema και όχι κάποιου είδους ειδική προτεραιότητα. Ακόμη, και η σειρά με την οποία δηλώνονται οι αναλυτές (analyzers), οι διασπαστές(tokenizers) και τα φίλτρα(filters) είναι δεδομένη και δεν υπόκειται σε αυθαίρετους κανόνες και προτιμήσεις.

Κάθε πεδίο(fieldtype) στο managed-schema, για να δουλέψει σωστά, θα πρέπει να είναι της μορφής:

analyzer

tokenizer

filter (ένα ή περισσότερα)

Αν χρειάζεται να υλοποιηθούν διαφορετικές τεχνικές κατά την περίοδο του indexing ή του querying σύμφωνα με την ακόλουθη:

analyzer (index)

tokenizer

filter (ένα ή περισσότερα)

analyzer (query)

tokenizer

filter (ένα ή περισσότερα)

2.6.3 Αναλυτές και φίλτρα

Ξεκινώντας με τους διασπαστές (tokenizers), το Solr διαθέτει μια πληθώρα διασπαστών, οι οποίοι αριθμούνται σε 15 στην έκδοση που εμείς χρησιμοποιούμε στο σύστημα μας (7.3) και κατά πάσα πιθανότητα θα ικανοποιήσουν μεγάλη μερίδα χρηστών.

Παρ' όλα αυτά, εμείς θα αναφερθούμε κυρίως σε κάποιους βασικούς που μπορούν να χρησιμοποιηθούν σε κάθε περίπτωση και να εναλλάσσεται η χρήση τους για πειραματικούς σκοπούς, οι οποίοι είναι:

1. Standard Tokenizer : όπου χωρίζει το πεδίο κειμένου σε τμήματα, αντιμετωπίζοντας το κενό και τα σημεία στίξης ως οριοθέτες.

2. Classic Tokenizer : λειτουργεί όπως ο Standard αλλά διατηρεί τα σημεία στίξης ανάμεσα σε λέξεις,

3. White Space Tokenizer : χωρίζει το κείμενο στα κενά και επιστρέφει τους μη κενές ακολουθίες χαρακτήρων

Όσον αφορά τα φίλτρα, ο αριθμός τους στην έκδοση 7.3 του Solr στη δική μας περίπτωση είναι 45, προσφέροντας έτσι πολλές δυνατότητες στον χρήστη. Και σε αυτή την περίπτωση θα αναφερθούμε ενδελεχέστερα σε αυτά που εμείς χρησιμοποιήσαμε.

Αρχικά, για μετατροπή όλων των χαρακτήρων του κειμένου σε πεζούς, υπάρχει το LowerCaseFilterFactory το οποίο κάνει τη συγκεκριμένη δουλειά χωρίς να χρειάζεται κάποια άλλη παραμετροποίηση.

Αν θέλουμε να αφαιρέσουμε τα stopwords από το κείμενο μας, μπορούμε να το κάνουμε με το φίλτρο StopFilterFactory, όπου χρειάζεται ένα αρχείο με τα stopword γραμμένα μέσα σ αυτό, για να μπορέσει να εκτελέσει την αφαίρεση.

Τέλος, παρόμοια μέθοδο λειτουργίας με αυτό για την αφαίρεση των stopwords, έχει και το φίλτρο KeywordMarkerFilterFactory, όπου με βάση ένα δοθέν αρχείο κρατάει ανέπαφες τις λέξεις που υπάρχουν σε αυτό, από οποιαδήποτε επεξεργασία ακολουθεί από τα επακόλουθα φίλτρα.

Κάποια εξίσου σημαντικά και ίσως περισσότερο σύνθετα φίλτρα μπορούν να χαρακτηριστούν τα WordDelimiterGraphFilterFactory, FlattenGraphFilterFactory και SynonymGraphFilterFactory τα οποία θα τα αναλύσουμε αμέσως.

Το FlattenGraphFilterFactory πρέπει να συμπεριλαμβάνεται στις προδιαγραφές του πεδίου τη στιγμή του indexing, όταν χρησιμοποιείται είτε το WordDelimiterGraphFilterFactory είτε το SynonymGraphFilterFactory σύμφωνα με τις οδηγίες του Solr. Ο λόγος είναι ότι τα προηγούμενα δύο φίλτρα παράγουν γράφους μετά την εφαρμογή τους και χρειάζεται να μετατραπεί το παραχθέν κείμενο σε φιλική και αποδοτική μορφή ως προς το indexing, ρόλος που επιτελείτε από το συγκεκριμένο φίλτρο.

Το WordDelimiterGraphFilterFactory διαχωρίζει τις λέξεις σε δευτερεύοντες λέξεις και πραγματοποιεί προαιρετικούς μετασχηματισμούς σε ομάδες υποομάδων. Οι λέξεις χωρίζονται σε λέξεις με τους ακόλουθους κανόνες:

1. Το generateWordParts προκαλεί τη δημιουργία τμημάτων λέξεων: η λέξη PowerShot μετατρέπεται για παράδειγμα σε Power, Shot.

2. Το generateNumberParts ωθεί στο να δημιουργηθούν δευτερεύουσες λέξεις: το 500-42 γνεται 500, 42.

3. Το catenateWords προκαλεί μέγιστες διαδρομές των τμημάτων λέξεων που πρέπει να κατανεμηθούν: αλλαγή από wi-fi σε wifi .

4. Το catenateNumbers προκαλεί μέγιστες διαδρομές των αριθμών που πρέπει να τεθούν σε διαχωρισμό: το 500-42 γίνεται 50042.

5. Το catenateAll προκαλεί κατακερματισμό όλων των τμημάτων του κειμένου: το wi-fi-4000 μετατρέπεται σε wifi4000.

6. Το splitOnCaseChange διασπά τις περιπτώσεις όπου έχουμε σύνθετες λέξεις ή φράσεις όπου κάθε επόμενη λέξη ή συντομογραφία αρχίζει με κεφαλαίο γράμμα: από το RoboBlaster/9000 προκύπτουν τα Robo, Blaster, 9000.

Για χρήση του αντίστοιχου κανόνα,πρέπει να τον θέσεις ίσον με 1 και για μη χρήση του ίσο με 0. Εμείς χρησιμοποιήσαμε τις εξής τιμές, δηλαδή για indexing μόνο το catenateAll μηδέν και οι υπόλοιποι κανόνες 1 και για query catenateWords, catenateNumbers και catenateAll 0 και οι υπόλοιποι 1.

Το SynonymGraphFilterFactory αλλάζει κάποιες λέξεις με τα συνώνυμα τους. Για να επιτευχθεί αυτό, είναι απαραίτητο ένα αρχείο που θα περιέχει τα συνώνυμα για κάθε λέξη που χρειάζεται να αντικατασταθεί. Κάθε γραμμή του αρχείου θα πρέπει να είναι σύμφωνα με μία από τις 4 ακόλουθες περιπτώσεις.

1. couch,sofa,divan: όπου η λέξη

2. $teh \Rightarrow the$

3. $huge, ginormous, humungous \Rightarrow large$

4. $small \Rightarrow tiny, teeny, weeny$

Όπου για την πρώτη περίπτωση ισχύει ότι το couch θα αντικατασταθεί από το σύνολο λέξεων couch, sofa, divan .

Στη δεύτερη περίπτωση το teh θα γίνει the

Στη τρίτη περίπτωση, οποιαδήποτε από τις λέξεις huge, ginormous, humungous βρεθεί θα αντικατασταθεί από τη large.

Τέλος στην τέταρτη περίπτωση στη θέση του small θα έχουμε πια τα tiny, teeny, weeny.

2.6.4 Stemming και lemmatization

Το Stemming (αποκατάληξη) συνήθως αναφέρεται σε μια ακατέργαστη ευριστική διαδικασία που απομακρύνει τα άκρα των λέξεων με την ελπίδα να επιτύχει αυτό το στόχο σωστά τις περισσότερες φορές και συχνά περιλαμβάνει την αφαίρεση των παραλλαγών της παραγωγής.

Stemming στο Solr μπορούμε να πετύχουμε μέσω 4 φίλτρων όπου το καθένα πραγματοποιεί διαφορετικό αλγόριθμο stemming. Οι 4 διαθέσιμοι είναι οι ακόλουθοι:

1. SnowballPorterFilterFactory : Υποστηρίζει 19 γλώσσες και εφαρμόζει διαφορετική τεχνική για κάθε μία
2. PorterStemFilterFactory : Υποστηρίζει μόνο αγγλικά, είναι γρήγορος και αφαιρεί τα κοινές καταλήξεις από τις λέξεις. Επίσης είναι από τους παλιότερους.
3. HunspellStemFilterFactory: Μπορεί να χρησιμοποιηθεί για 99 γλώσσες και χρησιμοποιεί και λεξικό και κανόνες γραμματικής. Γι αυτό το λόγο χρειάζεται ένα αρχείο .dic και ένα .affix για να δουλέψει.
4. KStemFilterFactory : Προορίζεται μόνο για εφαρμογή σε αγγλικό κείμενο και είναι πιο γρήγορος αλγόριθμος και λιγότερος επιθετικός από τον Porter

Με σκοπό την καλύτερη κατανόηση της λειτουργίας τους, ο ακόλουθος πίνακας περιγράφει τη λειτουργίας τους (Πίνακας 2.1).

Πίνακας 2.1: Παράδειγμα λειτουργίας stemming αλγόριθμων

Λέξη	SnowballPorter	PorterStem	HunspellStem	KStem
Conditioner	condition	condition	condition	condition
Conditioning	condit	condit	conditioning; condition	condition

Lemmatization, στα ελληνικά σημαίνει λημματοποίηση, δηλαδή αναγωγή στον πρώτο κλιτικό τύπο, έχει ως στόχο την αποκοπή καταλήξεων της λέξης σύμφωνα με μορφολογικά και λεξιλογικά πρότυπα, επιστρέφοντας έτσι στην βασική ή λεξικολογική μορφή της λέξης.

Στο Solr δεν υπάρχει κάποιο φίλτρο όπως στην περίπτωση του Stemming, αλλά από την έκδοση 7.3 υποστηρίζεται λημματοποίηση με τη βοήθεια του OpenNLP. Το OpenNLP Lemmatizer Filter μπορεί να λειτουργήσει είτε με ένα λεξικολογικό αρχείο της μορφής .txt, είτε με ένα μοντέλο λημματοποίησης της μορφής .bin είτε και με τα δύο ταυτόχρονα.

2.6.5 TF-IDF και BM25

Η προκαθορισμένη επιλογή αναπαράστασης στην έκδοση του Solr που χρησιμοποιήσαμε (7.3) είναι το BM25, παρόλα αυτά πάντως, παρέχετται στον χρήστη

δυνατότητα αλλαγής και χρήσης του TF-DF .

Αυτό μπορεί να επιτευχθεί μέσω μιας απλής δήλωσης είτε στην αρχή του managed-schema της συλλογής μας, για κοθολική εφαρμογή, είτε μέσα σε κάποιο πεδίο (fieldtype), αν επιθυμούμε συγκεκριμένη και περιορισμένη χρήση. Η εντολή είναι η παρακάτω:
<similarity class=”org.apache.lucene.search.similarities.ClassicSimilarity”/> ,
η οποία είναι υποκλάση της TFIDFSimilarity.

Και η κλάση TFIDFSimilarity, αλλά και η BM25Similarity, είναι και οι δύο υποκλάσεις της κλάσης Similarity της Solr, η οποία εκτός από αυτές τις δύο υποκλάσεις, περιλάμβανει και τις BooleanSimilarity, MultiSimilarity, PerFieldSimilarityWrapper και SimilarityBase.

Μέχρι τις 8 Απριλίου του 2016 η προκαθαρισμένη κλάση ομοιότητας της Lucene, την οποία χρησιμοποιεί το Solr, ήταν το TF-IDF, αλλά με την την κυκλοφορία της έκτης έκδοσης, έγινε αλλαγή στην BM25, η οποία θεωρήθηκε ανώτερη και καλύτερη εναλλακτική.

2.6.6 Word2Vec και Glove

Στη Lucene και κατ’ επέκταση στη Solr, δεν υπάρχει υλοποίηση word-embeddings, όπως των Word2Vec και Glove κι επομένως δεν μπορούν να χρησιμοποιηθούν μέσω κάποιας κλάσης ή φίλτρου.

Παρ’ όλα αυτά, υπάρχει τρόπος να πειραματιστεί κάποιος, κάτι το οποίο πραγματοποιήσαμε και εμείς από την πλευρά μας.

Η μέθοδος που ακολουθήσαμε και στις δύο περιπτώσεις ήταν πριν το indexing των άρθρων στο Solr, να εκπαιδεύσουμε τα αντίστοιχα μοντέλα Word2Vec και Glove, να διασπάσουμε το λεξιλόγιο σε clusters μέσω του αλγόριθμου KMeans, να δημιουργήσουμε ένα αρχείο της μορφής που υποστηρίζει το SynonymGraphFilterFactory, και με τη βοήθεια αυτού να πραγματοποιήσουμε την ευρετηριοποίηση αλλά και τα ερωτήματα.

2.7 Angular

Η Angular (συνήθως αναφέρεται ως ”Angular 2”+ ή ”Angular v2+”) είναι μια πλατφόρμα front-end web εφαρμογών ανοιχτού κώδικα βασισμένη σε TypeScript, η οποία καθοδηγούμενη από μια ομάδα της Google υπεύθυνη για την Angular και από μια κοινότητα μεμονωμένων ατόμων και εταιρειών. Η Angular είναι πρότζεκτ από την ίδια ομάδα που δημιούργησε το AngularJS.

Αρχικά, η επανεγγραφή του AngularJS ονομάστηκε ”Angular 2” από την ομάδα δημιουργίας, αλλά αυτό οδήγησε σε σύγχυση μεταξύ των προγραμματιστών. Για να αποσαφηνιστεί, αποφασίστηκε να χρησιμοποιούνται διαφορετική όροι για η ομάδα ανακοίνωσε ότι πρέπει να χρησιμοποιηθούν ξεχωριστοί όροι για κάθε framework με το ”AngularJS” να αναφέρεται στις εκδόσεις 1.X και το ”Angular” χωρίς το ”JS” να αναφέρεται στις εκδόσεις 2 και άνω. [10]

Κεφάλαιο 3

Συναφείς Εργασίες

3.1 Υπάρχοντα συστήματα

Τα υπάρχοντα συστήματα που θα αναφερθούν αποτελούν ένα μέρος μόνο από αυτά που υπάρχουν στο σύνολο του παγκοσμίου ιστού. Η επιλογή τους έγινε με βάση της δημοφιλίας τους και για να δώσουμε μια εικόνα της κατάστασης που επικρατεί όσον αφορά την κάλυψη σε συστήματα πρότασης περιοδικών και πρότασης συνεδρίων και περιοδικών

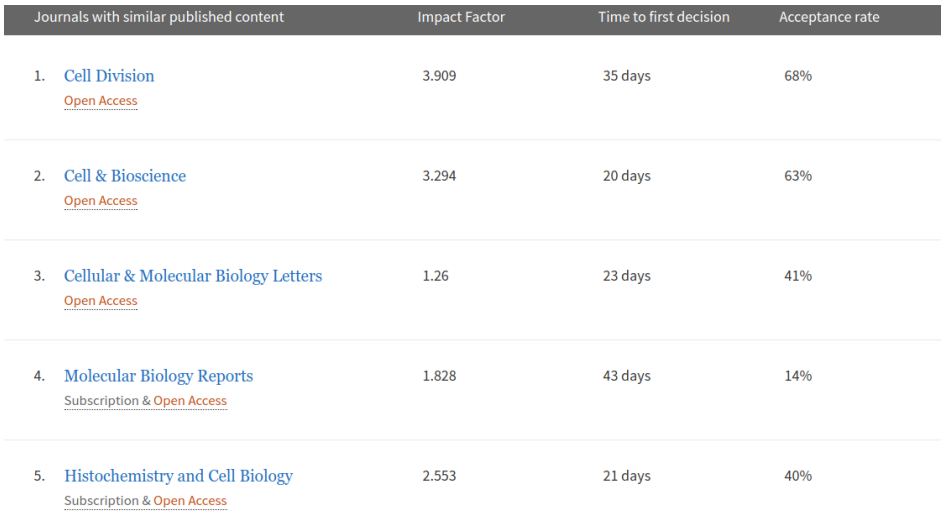
Τα υπάρχοντα συστήματα που θα παρουσιαστούν αποτελούν ένα μέρος μόνο από αυτά που υπάρχουν στο σύνολο του παγκοσμίου ιστού. Η επιλογή τους έγινε με βάση τη δημοφιλία τους και για να δώσουμε μια εικόνα της κατάστασης που επικρατεί όσον αφορά την κάλυψη σε συστήματα πρότασης περιοδικών και πρότασης συνεδρίων και περιοδικών

Αρχικα ας αναφερθούμε στα συστήματα που προτείνουν μόνο επιστημονικά περιοδικά.

Το Journal Suggester[1] του εκδοτικού οίκου Spinger προσφέρει μια ξεκάθαρη και κατανοητή διεπαφή όπου καταχωρώντας τον τίτλο και την περίληψη στα αντίστοιχα πεδία και επιλέγοντας προαιρετικά την θεματική περιοχή μπορείς να ανακαλύψεις ποια περιοδικά σου προτείνονται πατώντας το κουμπί αναζήτησης. Επιπλέον προσφέρει και κάποιες προχωρημένες επιλογές αναζήτησης δίνοντας την δυνατότητα να ορίσει ο χρήστης κάποιες προτιμήσεις σε σχέση με τα περιοδικά που θα εμφανιστούν ως προς τον συντελεστή απήχησης, το ποσοστό αποδοχής, την μέγιστη διάρκεια αναμονής μέχρι το περιοδικό να πάρει την *πρώτη απόφαση*[2] και τέλος την επιλογή ανοιχτών ή με συνδρομή περιοδικών. Τα αποτελέσματα εμφανίζονται σε λίστα με βάση τη σχετικότητα και με διακριτά τα πεδία του τίτλου του περιοδικού, του συντελεστή απήχησης, του χρόνου που μεσολαβεί μέχρι την πρώτη απόφαση και του ποσοστού αποδοχής (Εικόνα 3.1).

[1] `https://journalsuggester.springer.com`

[2] Απόφαση που λαμβάνεται από ένα περιοδικό σχετικά με μια εργασία έπειτα από την αρχική της υποβολή, και αφορά είτε απόρριψη είτε αποδοχή με αναθεώρηση ή χωρίς



Journals with similar published content	Impact Factor	Time to first decision	Acceptance rate
1. Cell Division Open Access	3.909	35 days	68%
2. Cell & Bioscience Open Access	3.294	20 days	63%
3. Cellular & Molecular Biology Letters Open Access	1.26	23 days	41%
4. Molecular Biology Reports Subscription & Open Access	1.828	43 days	14%
5. Histochemistry and Cell Biology Subscription & Open Access	2.553	21 days	40%

Εικόνα 3.1: Σελίδα αποτελεσμάτων Spinger Journal Suggester.

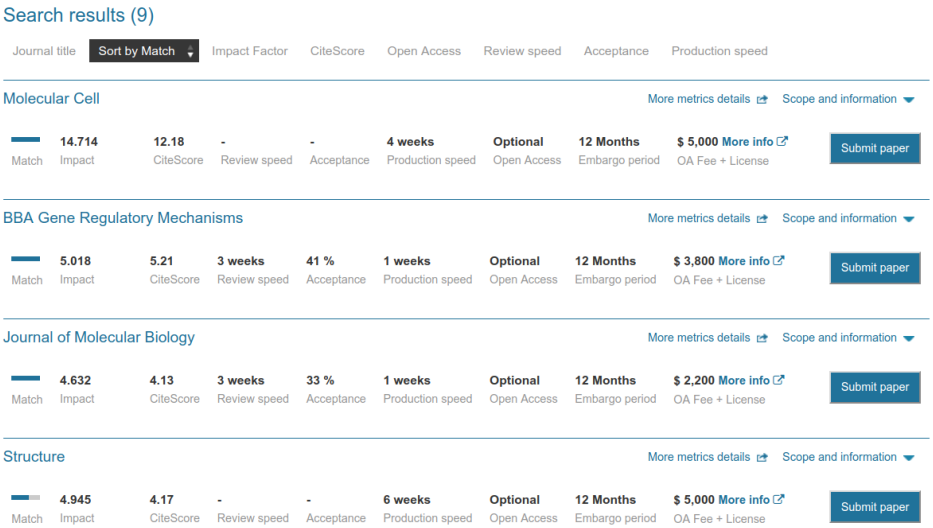
Το Journal Finder[3] του οίκου Elsevier παρέχει πεδία καταχώρησης τίτλου και περίληψης της εργασίας, δυνατότητα πολλαπλής επιλογής θεματικών ενοτήτων και επίσης επιλογής ανοικτών σε πρόσβαση μόνο επιστημονικών περιοδικών σε μια αρκετά απλουστευμένη διεπαφή. Η σελίδα των αποτελεσμάτων εμφανίζει αρχικά το πλήθος των αποτελεσμάτων, τα οποία είναι ταξινομημένα με βάση την σχετικότητα αλλά δίνεται επίσης η δυνατότητα στον χρήστη μέσω επιλέξιμων πεδίων πάνω από το σύνολο των περιοδικών να ταξινομήσει τα αποτελέσματα με βάση τον τίτλο του περιοδικού, το συντελεστή απήχησης, το CiteScore[4] (ειδική μετρική του οίκου Elsevier), το συντελεστή απήχησης και την ταχύτητα αξιολόγησης και δημοσίευσης (Εικόνα 3.2).

Ερχόμενοι στο σύστημα της Edanz, το Edanz Journal Selector[5] συναντάμε μια αρκετά διαφορετική προσέγγιση μόλις επισκεπτόμαστε την ιστοσελίδα. Η σελίδα ξεφεύγει από την προηγούμενη μορφή αναζήτησης η οποία χρειαζόταν καταχώρηση τίτλου και περίληψης έτσι ώστε να γίνει η αναζήτηση. Σε αυτήν την περίπτωση υπάρχει μόνο μια μπάρα αναζήτησης ώστε να μπορεί ο χρήστης να βρει τα περιοδικά που επιθυμεί, καταχωρώντας το τίτλο, τον εκδοτικό οίκο, το επιστημονικό πεδίο ή μία περίληψη κάθε φόρα, χρησιμοποιώντας τις επιλογές που προσφέρονται στα αριστερά της μπάρας αναζήτησης. Επίσης προσφέρεται και επιπλέον φιλτράρισμα των αποτελεσμάτων όσον αφορά το επιστημονικό πεδίο (αν αυτό δεν έχει οριστεί πριν την αναζήτηση), τον ορισμό ορίων σχετικά με το συντελεστή απήχησης, την εμφάνιση αν επιθυμείται μόνο όσων είναι ανοικτής πρόσβασης

[3] `http://journalfinder.elsevier.com`

[4] `https://www.elsevier.com/editors-update/story/journal-metrics/citescore-a-new-metric-to-help-you-choose-the-right-journal`

[5] `https://www.edanzediting.com/journal-selector`



Εικόνα 3.2: Σελίδα αποτελεσμάτων Elsevier Journal Finder.

περιοδικών και όσων έχουν καταχωρηθεί στα SCI[6] (Science Citation Index) και SCI-E (Science Citation Index Expanded). Τα επιστρεφόμενα αποτελέσματα συνοδεύονται από τον ατομικό τους συντελεστή απήχησης και το έτος υπολογισμό του, καθώς και την πληροφορία σχετικά με το αν είναι καταχωρημένα στο SCI-E. Επιπλέον επιτρέπεται στο χρήστη να ταξινομήσει τα αποτελέσματα σύμφωνα με τον τίτλο, το συντελεστή απήχησης και την συχνότητα έκδοσης του περιοδικού με αύξουσα και φθίνουσα σειρά και στις τρεις περιπτώσεις αντίστοιχα (Εικόνα 3.3).

Παρόμοια λογική στην αναζήτηση ακολουθεί και το JournalGuide[7]. Κυριαρχεί μια μπάρα αναζήτησης με μέσω της οποίας μπορεί ο χρήστης να βρει τα περιοδικά που επιθυμεί δίνοντας τίτλο και περιγραφή εργασίας, τίτλο περιοδικού, εκδοτικό οίκο ή θεματική ενότητα. Τα αποτελέσματα εμφανίζονται με μορφή λίστας ταξινομημένα κατά μεγαλύτερο βαθμό σχετικότητας, αλλά αυτό μπορεί να προσαρμοστεί ανάλογα με τις προτιμήσεις του χρήστη με βάση τον τίτλο του περιοδικού, τον εκδότη, το συντελεστή απήχησης, τη ταχύτητα έκδοσης και της ύπαρξης ανοικτής ή κλειστής προσβασιμότητας. Επίσης πάνω από τα αποτελέσματα εμφανίζεται η μπάρα αναζήτησης με την προσθήκη επιπλέον φίλτρων αυτή την φορά που αφορούν τα πεδία του συντελεστή απήχησης, είδους πρόσβασης του περιοδικού, χρονικής περιόδου αναζήτησης των περιοδικών και αν θέλουμε να συμπεριληφθούν έγγραφα χωρίς SNIP [8](Source Normalized Impact per Paper) (Εικόνα 3.4).

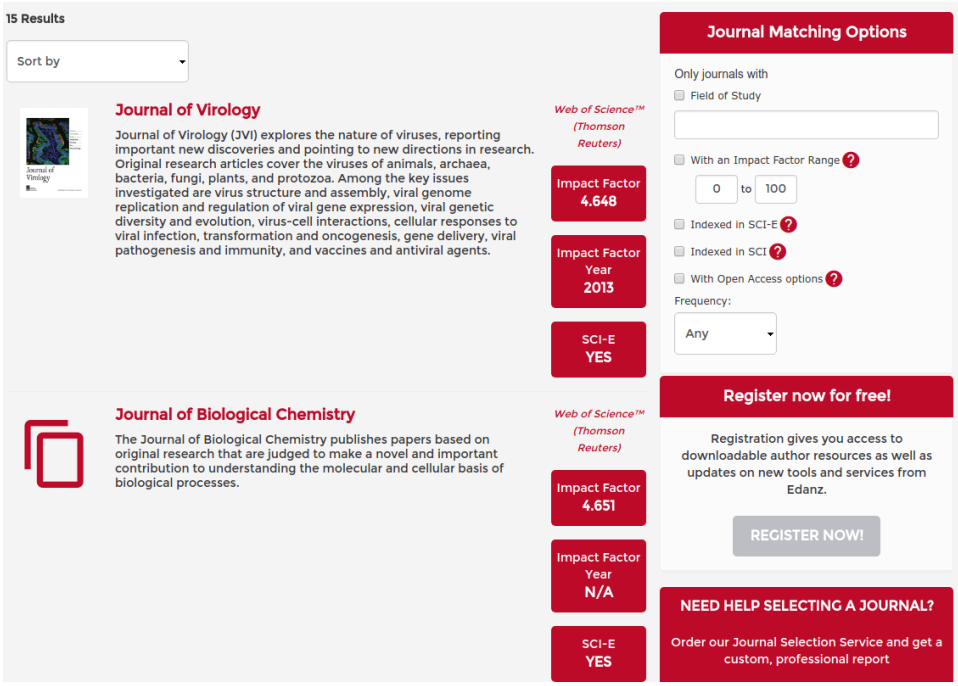
Τέλος θα αναφερθούμε στο IEEE Publication Reccomender[9] το οποίο συνδυά-

[6]δείκτης παραπομπής επιστημονικών αναφορών της Clarivate Analytics

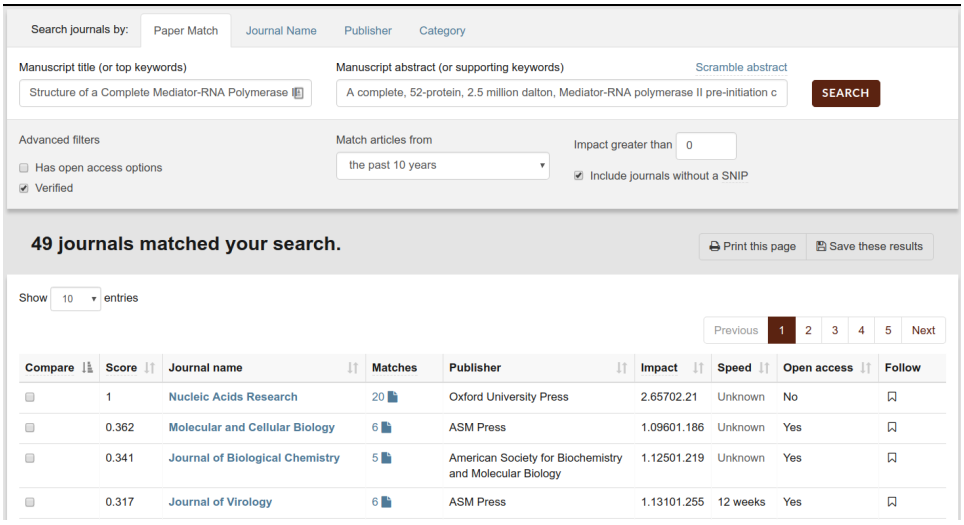
[7]`https://www.journalguide.com`

[8]Το SNIP μετρά τον αντίκτυπο της παραπομπής μιας πηγής με βάση την αναλογία βαρών βάσει του συνολικού αριθμού των αναφορών σε ένα πεδίο θεμάτων

[9]`http://publication-recommender.ieee.org`



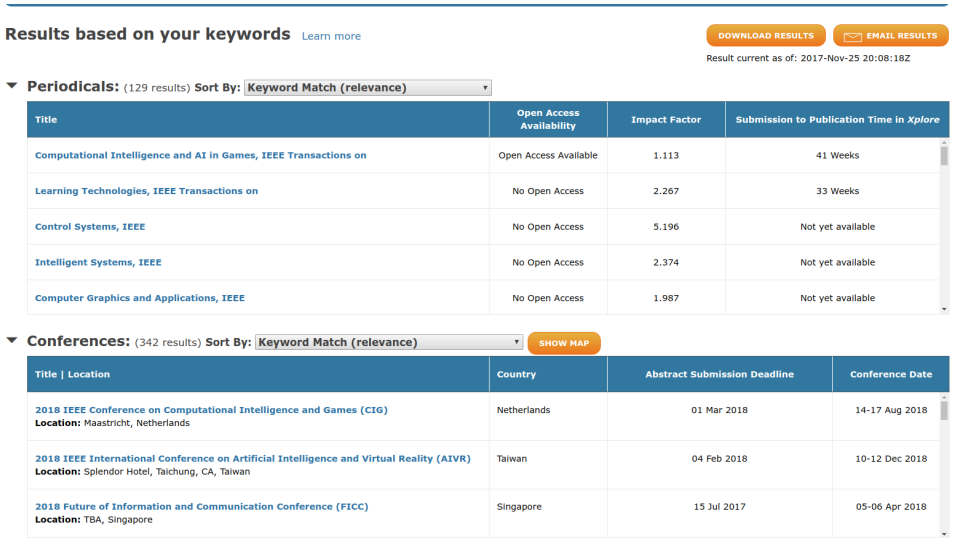
Εικόνα 3.3: Σελίδα αποτελεσμάτων Edanz Journal Selector.



Εικόνα 3.4: Σελίδα αποτελεσμάτων JournalGuide.

ζει την πρόταση συνεδρίων και περιοδικών. Αρχικά αξίζει να αναφέρουμε ότι είναι το μόνο το οποίο δίνει στον χρήστη τη δυνατότητα να ανεβάσει ένα αρχείο και το σύστημα να του δώσει τις ανάλογες συστάσεις βάση των λέξεων κλειδιών που θα εντοπίσει. Εναλλακτικά μπορεί να δώσει κάποιες λέξεις κλειδιά ή ένα τίτλο βάση των οποίων θα του επιστραφούν τα αποτελέσματα για περιοδικά, συνέδρια ή και

τα δύο. Επιπρόσθετα, μια ακόμη μοναδική υπηρεσία που προσφέρει είναι ότι ο χρήστης μπορεί να δώσει μία μελλοντική ημερομηνία μέχρι την οποία επιθυμεί η εργασία του να δημοσιευθεί και έτσι να του εμφανιστούν οι επιλογές που τον ικανοποιούν χρονικά. Όσον αφορά τα περιοδικά, τα αποτελέσματα δίνουν τις απαραίτητες πληροφορίες όσον αφορά το τίτλο, την απήχηση, το είδος προσβασιμότητας του περιοδικού και το χρόνο που χρειάζεται μέχρι τη δημοσίευση. Επίσης εμφανίζονται με σειρά σχετικότητας με βάση τις λέξεις κλειδιά, κάτι το οποίο μπορεί να αλλάξει ως προς την αλφαβητική σειρά του τίτλου των περιοδικών, του συντελεστή απήχησης και χρόνου που απαιτείται μέχρι την δημοσίευση με αύξουσα και φθίνουσα σειρά. Τα αποτελέσματα για τα συνέδρια ενημερώνουν τον χρήστη προφανώς για το ποιος είναι ο τίτλος, η ημερομηνία και η χώρα διεξαγωγής, και η προθεσμία υποβολής. Τα παραπάνω πεδία μπορούν να χρησιμοποιηθούν για την ταξινόμηση των αποτελεσμάτων με φθίνουσα ή αύξουσα σειρά και στην περίπτωσης της χώρας και του τίτλου του συνεδρίου, αλφαβητικά (Εικόνα 3.5).



Results based on your keywords Learn more

DOWNLOAD RESULTS | EMAIL RESULTS

Result current as of: 2017-Nov-25 20:08:18Z

Periodicals: (129 results) **Sort By:** Keyword Match (relevance)

Title	Open Access Availability	Impact Factor	Submission to Publication Time in *Xplore*
Computational Intelligence and AI in Games, IEEE Transactions on	Open Access Available	1.113	41 Weeks
Learning Technologies, IEEE Transactions on	No Open Access	2.267	33 Weeks
Control Systems, IEEE	No Open Access	5.196	Not yet available
Intelligent Systems, IEEE	No Open Access	2.374	Not yet available
Computer Graphics and Applications, IEEE	No Open Access	1.987	Not yet available

Conferences: (342 results) **Sort By:** Keyword Match (relevance) SHOW MAP

Title | Location	Country	Abstract Submission Deadline	Conference Date
2018 IEEE Conference on Computational Intelligence and Games (CIG) **Location:** Maastricht, Netherlands	Netherlands	01 Mar 2018	14-17 Aug 2018
2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR) **Location:** Splendor Hotel, Taichung, CA, Taiwan	Taiwan	04 Feb 2018	10-12 Dec 2018
2018 Future of Information and Communication Conference (FICC) **Location:** TBA, Singapore	Singapore	15 Jul 2017	05-06 Apr 2018

Εικόνα 3.5: Σελίδα αποτελεσμάτων IEEE Publication Reccomender.

Στο παρακάτω πίνακα (Πίνακας 3.1) θα παρουσιάσουμε συγκεντρωτικά τις διαφορές μεταξύ των συστημάτων όσον αφορά τα χαρακτηριστικά τους, τις επιλογές και τις δυνατότητες που προσφέρουν στον εκάστοτε χρήστη, με σκοπό να αποκτήσουμε μια πιο ξεκάθαρη εικόνα για το κάθε σύστημα.

Πίνακας 3.1: Σύγκριση Συστημάτων Συστάσεων.

Συστήματα / Χαρακτηριστικά	Spinger	Elsevier	Edanz	JournalGuide	IEEE
Συστάσεις περιοδικών	✓	✓	✓	✓	✓
Συστάσεις συνεδρίων					✓
Φίλτρα Αναζήτησης	✓	✓			✓
Αναζήτηση με περίληψη και τίτλο	✓	✓	✓	✓	✓
Αναζήτηση με λέξεις κλειδιά			✓	✓	✓
Αναζήτηση ανά θεματική ενότητα			✓	✓	
Αναζήτηση ανά εκδοτικό οίκο ή διοργανωτή			✓	✓	
Αναζήτηση με ανέβασμα αρχείου					✓
Ταξινόμηση αποτελεσμάτων		✓	✓	✓	✓

3.2 Εργασίες και διατριβές

Μια σχετική εργασία στην οποία αξίζει να αναφερθούμε είναι αυτή των Zaihan Yang, Brian D. Davison με τίτλο “Venue Recommendation: Submitting Your Paper with Style” [11].

Στη συγκεκριμένη τονίζεται το πρόβλημα της επιλογής περιοδικού από έναν ερευνητή ο οποίος είτε είναι νέος σε ένα ερευνητικό τομέα είτε για κάποιο λόγο δεν έγινε δεκτή η εργασία του από το περιοδικό που είχε σαν αρχική επιλογή και χρειάζεται εναλλακτικές.

Σύμφωνα με τους συγγραφείς, το σύστημα συστάσεων τους, πραγματοποιώντας φιλτράρισμα με βάση το θέμα αλλά και τη συγγραφική μορφή που πρέπει να τηρείται στο εκάστοτε περιοδικό, μπορεί να προτείνει στον ερευνητή ένα πλήθος περιοδικών τα οποία είναι κατάλληλα για δημοσίευση της εργασίας του. Τα δεδομένα που χρησιμοποιεί το σύστημα είναι από τις ηλεκτρονικές βιβλιοθήκες των ACM και Cite Seer, προκειμένου να προσφέρουν αποτελεσματικές συστάσεις.

Κεφάλαιο 4

Το Σύστημα μας

4.1 Πειραματισμός

4.1.1 Γενικες πληροφορίες

Για τους διάφορους πειραματισμούς μας χρησιμοποιήσαμε υλικό το οποίο ήταν διαθέσιμο μέσω του BioASQ[1] και περιέχει άρθρα από το PubMed. Κατεβάσαμε το αντίστοιχο JSON αρχείο που περιλάμβανε άρθρα μέχρι το 2017 και είναι το πιο πρόσφατο από τα διαθέσιμα, όπου για κάθε άρθρο υπήρχαν τα αντίστοιχα πεδία:

1. pmid: το μοναδικό αναγνωριστικό id κάθε άρθρου,
2. title: ο τίτλος του άρθρου,
3. abstractText: η περίληψη του άρθρου,
4. year: το έτος της δημοσίευσης του άρθου,
5. journal: το περιοδικό όπου το άρθρο δημοσιεύτηκε, και
6. meshMajor: μια λίστα με τις MeSH επικεφαλίδες κάθε άρθρου.

Στη προσέγγιση μας δημιουργήσαμε εκ νέου ένα πεδίο το οποίο προήλθε από την ένωση των πεδίων τίτλου και περίληψης, στο οποίο πραγματοποιήσαμε τις κάθε είδους επεξεργασίες και αναζητήσεις για την εύρεση των κοντινότερων περιοδικών. Δουλέψαμε κυρίως με τα πρώτα 10.000 άρθρα, ώστε να έχουμε ένα αρκετά ευέλικτο αριθμό για συνεχείς αλλαγές και πειραματισμούς.

Αρχικά υπήρχε η σκέψη να δώσουμε μια κάποια βαρύτητα στο τίτλο είτε μέσω ενός query boost(dismax ή edismax) του Solr, που δίνει βαρύτητα σε ένα πεδίο, είτε μέσω τόνωσης των όρων του τίτλου που υποστηρίζεται επίσης στο Solr, παρατηρήσαμε ότι έτσι ερχόταν σε δεύτερη μοίρα το περιεχόμενο της περίληψης.

Σαν αποτέλεσμα είχαμε επιστρεφόμενα περιοδικά τα οποία μπορεί να έμοιαζαν στον τίτλο, σε ότι αφορά το περιεχόμενο όμως δεν ήταν και τόσο κοντά. Λόγω

[1] `http://participants-area.bioasq.org/general_information/Task5a/`

αυτού του φαινομένου προτιμήσαμε την συνένωση των πεδίων του τίτλου και της περίληψης σε ένα ενιαίο πεδίο.

Ακόμη δουλέψαμε και με τα δύο είδα αναπαράστασης BM25 και TF-IDF, για να διαπιστώσουμε αν η ανωτερότητα του BM25 επιβεβαιώνεται και στο δικό μας σύνολο δεδομένων. Στη παρουσίαση των πειραμάτων θα παρουσιάζουμε πάντα πρώτο το BM25 και στη συνέχεια το TF-IDF για να διατηρήσουμε μια συνέπεια.

Τέλος για να συγκρίνουμε τις επιδόσεις κάθε δοκιμής προσπαθήσαμε σε όσο το δυνατόν ιδανικότερο περιβάλλον, να χρονομετρήσουμε τον χρόνο που απαιτούσε κάθε μέθοδος για indexing, querying και καταγραφή αποτελεσμάτων, ώστε να έχουμε μια πιο ολοκληρωμένη εικόνα της κάθε εφαρμογής για σύνολο 10.000 άρθρων, που αποτελούσε και το γενικό δοκιμαστικό μας σύνολο.

4.1.2 Μέδοθος αξιολόγησης MRR

Για την αξιολόγηση των αποτελεσμάτων δουλέψαμε με το Mean reciprocal rank (MRR) [12] και το εφαρμόσαμε σε επαναληπτική αναζήτηση για το ιδανικό περιοδικό, για κάθε ένα από τα 10.000 άρθρα της συλλογής.

Η μέση αμοιβαία κατάταξη (MRR) είναι ένα στατιστικό μέτρο για την αξιολόγηση κάθε διαδικασίας που παράγει μια λίστα με τις πιθανές απαντήσεις για κάθε μια ερώτηση, σε ένα σύνολο ερωτήσεων, ταξινομημένα κατά τη πιθανότητα ορθότητας.

Η αμοιβαία κατάταξη της απάντησης ενός ερωτήματος είναι το πολλαπλασιαστικό αντίστροφο της κατάταξης της πρώτης σωστής απάντησης: 1/1 για την πρώτη θέση, 1/2 για τη δεύτερη θέση, 1/3 για την τρίτη θέση και ούτω καθεξής. Η μέση αμοιβαία κατάταξη (MRR) είναι ο μέσος όρος των αμοιβαίων κατατάξεων των αποτελεσμάτων για ένα συγκεκριμένο δείγμα ερωτήσεων.

Για παράδειγμα ας θεωρήσουμε ότι έχουμε τρεις αναζητήσεις άρθρων στη συλλογή με σκοπό να βρούμε το ιδανικό περιοδικό δημοσίευσης και σε κάθε ερώτημα ότι επιστρέφονται τρία περιοδικά για λόγους ευκολίας.

Πίνακας 4.1: Παράδειγμα υπολογισμού MRR

Ερώτημα	Αποτελέσματα	Κατάταξη	Reciprocal rank
Άρθρο του Περ. 1	Περ. 5, Περ. 5, Περ. 1	3	1/3
Άρθρο του Περ. 3	Περ. 5, Περ. 3, Περ. 6	2	1/2
Άρθρο του Περ. 4	Περ. 4, Περ. 5, Περ. 4	1	1/1

Δεδομένου αυτών τριών ερωτημάτων, θα μπορούσαμε να υπολογίσουμε το MRR (mean reciprocal rank) ως εξής: (1/3 + 1/2 + 1)/3 = 11/18 ή περίπου 0.61. Αν κανένα από τα επιστρεφόμενα αποτελέσματα δεν μας δώσει το αρχικό περιοδικό τότε το reciprocal rank είναι 0.

4.1.3 Stemming

Η πρώτη φάση πειραματισμού στο σύστημα μας, αφορούσε την εφαρμογή Stemming στο πεδίο όπου είχαμε αποθηκεύσει το τίτλο και την περίληψη.

Στις αρχικές μας δοκιμές, πριν από το κατάλληλο φίλτρο της Solr για stemming, χρησιμοποιήσαμε για διάσπαση του κειμένου τον Standard Tokenizer, για μετατροπή σε πεζά το Lower Case Filter Factory και για αφαίρεση των stopwords το Stop Filter Factory και στη συνέχεια το αντίστοιχο φίλτρο για κάθε περίπτωση stemming.

Λόγω της μεγάλης ποικιλίας από φίλτρα και διασπαστές που διαθέτει η Solr όμως, μας δόθηκε η δυνατότητα να δοκιμάσουμε διάφορους συνδυασμούς ώστε να επιδιώξουμε να βελτιώσουμε τα αποτελέσματα που είχαμε από την κλασσική εφαρμογή των αλγόριθμων stemming.

Έτσι μετά από πειραματισμό, καταλήξαμε στο να υλοποιήσουμε και δεύτερο κύκλο δοκιμών stemming με βάση το εξής μοντέλο:
Stemming με tokenizer τον WhiteSpace και τα εξής φίλτρα με την ακόλουθη σειρά:

1. WordDelimiterGraphFilterFactory
2. LowerCaseFilterFactory
3. StopFilterFactory
4. Εκάστοτε φίλτρο αλγόριθμου stemming
5. FlattenGraphFilterFactory (μόνο στο indexing)

Για τη μέθοδο HunspellStem χρησιμοποιήσαμε Αγγλικό (Ηνωμένων Πολιτειών) λεξιλόγιο. Οι μετρήσεις έγιναν σε όσο ήταν δυνατόν πανομοιότυπες συνθήκες, για 10.000 ερωτήματα μαζί με την καταγραφή των εκάστοτε αλλά και γενικών αποτελεσμάτων.
Τα αποτελέσματα που προέκυψαν περιγράφονται στη συνέχεια αναλυτικά για κάθε περίπτωση.

4.1.3.1 BM25

Τα αποτελέσματα του MRR για το πρώτη δοκιμή stemming, με χρήση BM25 για 10.000 άρθρα, ο χρόνος εκτέλεσης της εισαγωγής και αναζήτησης φαίνονται στο παρακάτω πίνακα (Πίνακας 4.2).

Οι διαφορές των μεθόδων όσον αφορά το MRR είναι πάρα πολύ μικρές όπως φαίνεται, με καλύτερα αποτελέσματα αυτά του KStem, στην συνέχεια του HunspellStem και στο τέλος των SnowballPorter και PorterStem όπου ισοβαθμούν.

Επίσης και χρονικά δεν εντοπίζονται μεγάλες αποκλίσεις μεταξύ των μεθόδων, τόσο στο κομμάτι του indexing, τόσο και στο κομμάτι του querying, εκτός από τη μέθοδο HunspellStem. Πάντως μπορούν να μας δώσουν κάποιες ενδείξεις για τα πια θα είναι η πιο γρήγορη λειτουργία stemming.

Πίνακας 4.2: BM25 Αποτελέσματα MRR Solr stemming για 10.000 άρθρα.

Μέθοδος Stemming	MRR	Index	Query
KStem	0.4199	00:00:13	00:05:12
PorterStem	0.4163	00:00:13	00:05:13
SnowballPorter	0.4163	00:00:14	00:05:14
HunspellStem	0.4186	00:00:17	00:05:25

Αν χρειαζόταν δηλαδή να τα βάλουμε σε μια σειρά αύξουσας συνολικής χρονικής απόδοσης, αυτή θα ήταν (από το πιο γρήγορο stemming στο πιο αργό) :

1. KStem με 5 λεπτά και 25 δευτερόλεπτα
2. Porter με 5 λεπτά και 26 δευτερόλεπτα
3. SnowballPorter με 5 λεπτά και 28 δευτερόλεπτα
4. Hunspell με 5 λεπτά και 42 δευτερόλεπτα

Για το δεύτερο τρόπο εφαρμογής stemming σε 10.000 άρθρα, τα αποτελέσματα του κάθε αλγόριθμου, μαζί με το χρόνο εκτέλεση της εισαγωγής και αναζήτησης, παρουσιάζονται στον επόμενο πίνακα (Πίνακας 4.3).

Πίνακας 4.3: BM25 Αποτελέσματα MRR Solr stemming (2η) για 10.000 άρθρα.

Μέθοδος Stemming	MRR	Index	Query
KStem	0.4191	00:00:15	00:05:20
PorterStem	0.4138	00:00:14	00:05:20
SnowballPorter	0.4137	00:00:15	00:05:22
HunspellStem	0.4154	00:00:19	00:05:32

Αυτό που μπορούμε να διακρίνουμε με ευκολία είναι ότι όλες οι μετρήσεις παρουσίασαν μείωση σε σχέση με τη πρώτη μέθοδο stemming. Παρ'όλα αυτά τα καλύτερα αποτελέσματα μας τα προσφέρει πάλι η μέθοδος KStem και ακολουθούν οι HunspellStem, SnowballPorter, και PorterStem.

Λόγω της εφαρμογής επιπλέον φίλτρων σε σχέση με τη πρώτη δοκιμή stemming, ο χρόνος κάθε εφαρμογής αυξήθηκε οδηγώντας στην ακόλουθη αύξουσα χρονικά ταξινόμηση:

1. Porter με 5 λεπτά και 34 δευτερόλεπτα
2. KStem με 5 λεπτά και 35 δευτερόλεπτα
3. SnowballPorter με 5 λεπτά και 37 δευτερόλεπτα

4. Hunspell με 5 λεπτά και 51 δευτερόλεπτα

Δεδομένου της χρονικής αύξησης της εκτέλεσης των μεθόδων stemming με το δεύτερο τρόπο και της μείωσης έστω και σε μικρό ποσοστό των επιμέρους αποτελεσμάτων MRR, θα χαρακτηρίζαμε αρνητική την εφαρμογή αυτή.

Τα συγκριτικά αποτελέσματα MRR παρουσιάζονται στο πίνακα που ακολουθεί(Πίνακας 4.7).

Πίνακας 4.4: Σύγκριση MRR BM25 δοκιμών Solr stemming για 10.000 άρθρα.

Μέθοδος Stemming	Δοκιμή 1	Δοκιμή 2
KStem	0.4199	0.4191
PorterStem	0.4163	0.4138
SnowballPorter	0.4163	0.4137
HunspellStem	0.4186	0.4154

Από το πίνακα μπορούμε να καταλάβουμε ότι το αποδοτικότερο stemming το προσφέρει η εφαρμογή του KStem σύμφωνα με τον πρώτο τρόπο.

4.1.3.2 TF-IDF

Τα αποτελέσματα του MRR για το πρώτο τρόπο stemming, με εφαρμογή του TF-IDF, για σύνολο 10.000 άρθρων, ο χρόνος εκτέλεσης της εισαγωγής και αναζήτησης φαίνονται στο παρακάτω πίνακα (Πίνακας 4.5).

Πίνακας 4.5: TF-IDF Αποτελέσματα MRR Solr stemming για 10.000 άρθρα.

Μέθοδος Stemming	MRR	Index	Query
KStem	0.4175	00:00:13	00:04:59
PorterStem	0.4146	00:00:14	00:04:59
SnowballPorter	0.4152	00:00:14	00:05:01
HunspellStem	0.4161	00:00:18	00:05:11

Όπως παρατηρούμε, οι μετρήσεις είναι πάρα πολύ κοντά. Μπορούμε να θεωρήσουμε ότι πρακτικά έχουμε τα ίδια αποτελέσματα. Αν και βάζοντας τα σε σειρά φθίνουσας απόδοσης θα είχαμε την εξής ταξινόμηση: KStem, HunspellStem, SnowballPorter, PorterStem

Από άποψη χρόνου οι διαφορές δεν ήταν πάρα πολύ μεγάλες μεταξύ των μεθόδων, παρατηρώντας τους χρόνους indexing και querying. Αισθητή διαφορά εντοπίζουμε πάλι στη μέθοδο HunspellStem, κάτι το οποίο μπορεί να μας οδηγήσει στην ακόλουθη αύξουσα σειρά συνολικής χρονικής απόδοσης (από το πιο γρήγορο stemming στο πιο αργό) :

1. KStem με 5 λεπτά και 12 δευτερόλεπτα

2. SnowballPorter με 5 λεπτά και 12 δευτερόλεπτα
3. Porter με 5 λεπτά και 15 δευτερόλεπτα
4. Hunspell με 5 λεπτά και 29 δευτερόλεπτα

Όσον αφορά το δεύτερο κύκλο δοκιμών stemming, τα αποτελέσματα κάθε αλγόριθμου για 10.000 άρθρα, μαζί με το χρόνο εκτέλεση της εισαγωγής και αναζήτησης, φαίνονται στον ακόλουθο πίνακα (Πίνακας 4.6).

Πίνακας 4.6: TF-IDF Αποτελέσματα MRR Solr stemming (2η) για 10.000 άρθρα.

Μέθοδος Stemming	MRR	Index	Query
KStem	0.4191	00:00:15	00:05:06
PorterStem	0.4144	00:00:14	00:05:05
SnowballPorter	0.4152	00:00:15	00:05:07
HunspellStem	0.4167	00:00:19	00:05:18

Και σε αυτή τη δοκιμή, τα αποτελέσματα είναι πάρα πολύ κοντά. Βάζοντας τα σε σειρά φθίνουσας απόδοσης θα είχαμε ξανά την ίδια ταξινόμηση με την προηγούμενη δοκιμή: KStem, HunspellStem, SnowballPorter, PorterStem, παρατηρόντας όμως ότι μόνο για η μέθοδος KStem, παρουσίασε κάποια βελτίωση, καθώς οι υπόλοιπες τρεις διατήρησαν τα ίδια ποσοστά, επομένως είναι ο προτιμητέος.

Ο συνολικός χρόνος εκτέλεσης όμως, αυξήθηκε για όλες τις μεθόδους, κάτι το οποίο ήταν φυσικό επόμενο λόγω και της αύξησης του αριθμού των φίλτρων που εκτελούνται.

Έτσι οδηγηθήκαμε στην εξής αύξουσα χρονικά ταξινόμηση των μεθόδων:

1. Porter με 5 λεπτά και 29 δευτερόλεπτα
2. KStem με 5 λεπτά και 30 δευτερόλεπτα
3. SnowballPorter με 5 λεπτά και 31 δευτερόλεπτα
4. Hunspell με 5 λεπτά και 36 δευτερόλεπτα

Ο χρόνος εκτέλεσης της όλης αυτής επαναληπτικής διαδικασίας αξιολόγησης παρουσιάζει διαφοροποιήσεις, όπου έχουμε και ανακατατάξεις στην ταξινόμηση των μεθόδων ως προς τη χρονική απόδοση. Παρ' όλα αυτά τα οι χρόνοι είναι τόσο κοντά ο ένας με τον άλλον όπου δεν μπορούν αν βγουν ασφαλή και ισόβια χρονικά συμπεράσματα.

Ακολουθεί ένας συγκριτικός πίνακας για τις δύο δοκιμές όσον αφορά το αποτέλεσμα του MRR(Πίνακας 4.7).

Αυτό που μπορούμε να παρατηρήσουμε είναι ότι ο αλγόριθμος KStem είναι ο μόνος που στη δεύτερη δοκιμή βελτίωσε τα αποτελέσματα του, σε σχέση με τους υπόλοιπους οι οποίοι παρέμειναν στα ίδια ποσοστά.

Πίνακας 4.7: Σύγκριση MRR TF-IDF δοκιμών Solr stemming για 10.000 άρθρα.

Μέθοδος Stemming	Δοκιμή 1	Δοκιμή 2
KStem	0.4175	0.4191
PorterStem	0.4146	0.4144
SnowballPorter	0.4152	0.4152
HunspellStem	0.4161	0.4167

4.1.3.3 Σύγκριση BM25 και TF-IDF

Ο πίνακας 4.8 παρουσιάζει συγκεντρωτικά τα αποτελέσματα stemming για BM25 και TF-IDF

Πίνακας 4.8: Σύγκριση MRR BM25 και TF-IDF δοκιμών Solr stemming για 10.000 άρθρα.

Μέθοδος Stemming	BM25 Δοκιμή 1	TF-DIF Δοκιμή 1	BM25 Δοκιμή 2	TD-IDF Δοκιμή 2
KStem	0.4199	0.4175	0.4191	0.4191
PorterStem	0.4163	0.4146	0.4138	0.4144
SnowballPorter	0.4163	0.4152	0.4137	0.4152
HunspellStem	0.4186	0.4161	0.4154	0.4167

Από το παραπάνω συγκριτικό πίνακα προκύπτει ότι η καλύτερη εφαρμογή για την επίτευξη του μεγαλύτερου MRR είναι να δουλέψουμε με StanadardTokenizer και KStemFilter, συστατικά της πρώτης δοκιμής δηλαδή, καθώς έτσι επιτυγχάνουμε το καλύτερο MRR το 0.4199 (πρώτη σειρά. πρώτη στήλη του πίνακα 4.8).

Μια ακόμη διαπίστωση μας αφορά την επίδραση που είχε η χρήση WhiteSpaceTokenizer και WordDelimiterGraphFilterFactory πριν το φίλτρο του stemming. Ενώ το TF-IDF στο πρώτο γύρο δοκιμών και στις 4 περιπτώσεις έχει χαμηλότερο σκορ MRR απ' όταν χρησιμοποιούμε BM25, στη δεύτερη δοκιμή πετυχαίνει την ανατροπή των ισορροπιών. Αρχικά έχουμε το καλύτερο MRR με εφαρμογή KStem (0.4191), όπου μπορεί σ αυτή τη περίπτωση να έχει ίδιο MRR με το BM25, στις περιπτώσεις όμως PorterStem, SnowballPorter και HunspellStem βρίσκεται πάλι πιο πάνω στη κατάταξη από το BM25.

Στο τελικό συμπέρασμα όμως, ο συνδυασμός BM25 με StanadardTokenizer και KStemFilter, αποτελεί την καλύτερη επιλογή για stemming σε σύνολο 10.000 άρθρων.

4.1.4 Lemmatization

Η δεύτερη φάση πειραματισμού στο σύστημα μας αφορούσε την εφαρμογή Lemmatization στο πεδίο όπου είχαμε αποθηκεύσει το τίτλο και την περίληψη.

Στο Solr όπως αναφέραμε δεν υπάρχει κάποιο φίλτρο όπως στην περίπτωση του Stemming, αλλά από την έκδοση 7.3 υποστηρίζεται λημματοποίηση με τη βοήθεια

του OpenNLP. Τηρώντας κατά γράμμα τις οδηγίες που υπάρχουν στην επίσημη ιστοσελίδα του Solr [2] προσπαθήσαμε να εφαρμόσουμε αυτό τον τρόπο αλλά ο χρόνος ευρετηροποίησης και αναζήτησης ήταν πολύ μεγάλος για 10.000 άρθρα και το σύστημα δεν ήταν καθόλου αποδοτικό.

Στις δοκιμές μας στη Solr για lemmatization, χρησιμοποιήσαμε για διάσπαση του κειμένου τον Standard Tokenizer, για μετατροπή σε πεζά το LowerCaseFilterFactory και για αφαίρεση των stopwords το StopFilterFactory. Επίσης χρειαστήκαμε και το FlattenGraphFilterFactory κατά το indexing , σύμφωνα με τις οδηγίες του Solr, για τη χρήση του SynonymGraphFilterFactory σε κάποιες από τις ακόλουθες δοκιμές μας.

Επομένως για να πραγματοποιήσουμε Lemmatization, χρειάστηκε να καταφύγουμε σε άλλες λύσεις.

Η πρώτη περίπτωση ήταν να χρησιμοποιήσουμε την nltk [3] βιβλιοθληκη της Python.

Το Natural Language Toolkit, ή πιο συχνά το NLTK, είναι μια σουίτα βιβλιοθηκών και προγραμμάτων για συμβολική και στατιστική επεξεργασία φυσικής γλώσσας (NLP) για αγγλικά γραμμένα κείμενα στη γλώσσα προγραμματισμού Python.

Έτσι με τη βοήθεια της NLTK, καταφέραμε να πραγματοποιούμε lemmatization και στη συνέχεια να καταχωρούμε το αποτέλεσμα στο ειδικό πεδίο του Solr που χρησιμοποιούμε για τις αναζητήσεις μας. Lemmatization επίσης κάνουμε και στο κείμενο το οποίο στέλνουμε για αναζήτηση. Καθυστέρηση δεν παρατηρήθηκε ούτε στην εισαγωγή αλλά και ούτε στην πραγματοποίηση ερωτημάτων.

Στη δεύτερη περίπτωση χρησιμοποιήσαμε το φίλτρο της Solr, το SynonymGraphFilterFactory το οποίο έχει τη δυνατότητα να αντικαθιστά μία λέξη με την αντίστοιχη που υποδεικνύεται μέσα στο αρχείο.

Το αρχείο αυτό περιέχει τις λέξεις με την εξής μορφή:

$$word1 \Rightarrow newWord1$$

$$word2 \Rightarrow newWord2$$

$$word3 \Rightarrow newWord3$$

$$word4 \Rightarrow newWord4$$

Με άλλα λόγια, από την αριστερή πλευρά του βέλους είχαμε την λέξη που βρισκόταν στο κείμενο και στη δεξιά πλευρά είχαμε την λέξη που πρέπει να την αντικαταστήσει λόγω lemmatization.

Επομένως χρησιμοποιώντας πάλι τη βιβλιοθήκη της Python την nltk, μπορέσαμε να φτιάξουμε μια λίστα με όλες τι λέξεις, τις οποίες η λημματοποίηση τους είχε επίδραση πάνω τους, να τις γράψουμε στο αρχείο με την κατάλληλη μορφή και να χρησιμοποιήσουμε το φίλτρο SynonymGraphFilterFactory ώστε να επιτύχουμε

[2] `https://lucene.apache.org/Solr/guide/7_3/language-analysis.html`

[3] `https://www.nltk.org/`

lemmatization με Solr. Ο χρόνος δημιουργίας αυτού τ' αρχείου ήταν 30 δευτερόλεπτα.

Τέλος, στην τρίτη και στη τέταρτη και τελευταία περίπτωση, σε μια ακόμη προσπάθεια συνδυασμού μεθόδων, χρησιμοποιήσαμε το λεξιλόγιο που δημιουργήθηκε από την εφαρμογή Word2Vec και την εφαρμογή Glove αντίστοιχα,στο σύνολο των 10.000 άρθρων. Σε αυτό το λεξιλόγιο, πραγματοποιήσαμε lemmatization και δημιουργήσαμε ένα αντίστοιχο αρχείο για κάθε μία περίπτωση, με την προβλεπόμενη πάντα μορφή, για να το χρησιμοποιήσουμε πάλι μέσω του φίλτρου SynonymGraphFilterFactory για λημματοποίηση μέσω Solr.
Ο χρόνος για τη δημιουργία του αρχείου για Word2Vec και Glove ήταν και στις δύο περιπτώσεις 5 δευτερόλεπτα, χωρίς να προσμετρείται ο χρόνος που απαιτείται για να προκύψει το τελικό των Word2Vec και Glove (χρόνος εκτέλεσης των μοντέλων 18 και 40 δευτερόλεπτα αντίστοιχα).

Τα πειράματα μας χωρίζονται σε δύο γύρους, όπου ο ένας αφορά την εφαρμογή των τεσσάρων αυτών τρόπων lemmatization και ο δεύτερος γύρος την εφαρμογή KStem μετά το lemmatization. Ο λόγος που χρησιμοποιήσαμε τον KStem και όχι κάποια άλλη μέθοδο stemming είναι λόγω της καλύτερη του απόδοσης σε σχέση με τους υπόλοιπους αλγορίθμους.

4.1.4.1 BM25

Για το πρώτο γύρο δοκιμών lemmatization σε συνδυασμό με BM25, ο πίνακας 4.9 παρουσιάσει συγκεντρωτικά τα αποτελέσματα MRR και ο απαιτούμενος χρόνος Querying και Indexing, για το σύνολο των 10.000 άρθρων.

Πίνακας 4.9: BM25 Αποτελέσματα MRR Solr lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	MRR	Index	Query
NLTK Lemm	0.4151	00:00:24	00:05:06
SynonymFilterNLTK	0.4217	00:00:15	00:05:13
SynonymFilterWord2Vec	0.4214	00:00:14	00:05:13
SynonymFilterGlove	0.4214	00:00:14	00:05:13

Παρατηρώντας το πίνακα , προκύπτει το συμπέρασμα ότι χρησιμοποιώντας το SynonymGraphFilterFactory και ένα αρχείο με τις λέξεις που πρόκειται να αντικατασταθούν από τη λήμμα τους, πετύχαμε καλύτερα αποτελέσματα, σε σχέση με την απευθείας λημματοποίηση μέσω της βιβλιοθήκης NTLK. Επίσης το αρχείο λημματοποίησης που προέκυψε από το λεξιλόγιο του Word2Vec και Glove περιείχε 3022 μοναδικές λέξεις που θα άλλαζαν τελικά με το λήμμα τους ενώ το αρχείο που προέκυψε από την εφαρμογή του ntlk lemmatizer κατευθείαν στις λέξεις του συνόλου μας περιείχε 6020, περίπου τις διπλάσιες δηλαδή. Παρ' όλη αυτή τη μεγάλη διαφορά στον αριθμό των λέξεων που θα λημματοποιούνταν, τα αποτελέσματα είναι ελάχιστα καλύτερα με χρήση του αρχείου των 6020 λέξεων, έναντι 3022 των άλ-

λων δύο περιπτώσεων, χωρίς βέβαια να μπορεί να θεωρηθεί η διαφορά των 0.0003 μονάδων ιδιαίτερα σημαντική.

Σχετικά με το χρόνο, μόνο στην πρώτη περίπτωση η τεχνική της λημματοποίησης προσμετρήθηκε στο χρόνο του indexing και του querying, καθώς στις υπόλοιπες 3 το αρχείο με τις λημματοποιημένες λέξεις έπρεπε να ήταν έτοιμο πριν την ευρετηριοποίηση και αναζήτηση, καθώς θα χρησιμοποιόταν και στις 2 αυτές διαδικασίες.

Αν θέλουμε να κάνουμε μια συνολική εκτίμηση του χρόνου εκτέλεσης του lemmatization, η οποία θα περιλαμβάνει τη χρονική καταμέτρηση όλων των βημάτων αρκεί να ρίξουμε μια ματιά στον πίνακα 4.10.

Με αυτές τι μετρήσεις ο πρώτος τρόπος λημματοποίησης μέσω nltk lemmatizer είναι μακράν πιο γρήγορος απ όλους. Παρ' όλα αυτά μια ισορροπημένη επιλογή από άποψη MRR και χρόνου μπορεί να θεωρηθεί η λημματοποίηση με το λεξιλόγιο του Word2Vec.

Πίνακας 4.10: BM25 Χρόνοι lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	Index + Query	File Creation	Model Preparation	Sum
NLTK Lemm	00:05:30	00:00:00	00:00:00	00:05:30
SynonymFilterNLTK	00:05:28	00:00:30	00:00:00	00:05:58
SynonymFilterWord2Vec	00:05:27	00:00:05	00:00:18	00:05:50
SynonymFilterGlove	00:05:27	00:00:05	00:00:40	00:06:12

Στο επόμενο στάδιο πειραματισμού, πραγματοποιήσαμε και stemming με το KStem Filter, αφού είχαμε βέβαια πραγματοποιήσει lemmatization ανάλογο της κάθε περίπτωσης, είτε μέσω nltk είτε μέσω του SynonymGraphFilterFactory με το αντίστοιχο αρχείο κάθε φορά.

Τα συνολικά αποτελέσματα MRR και χρόνου Query και Index για το σύνολο των 10.000 άρθρων, εμφανίζονται με τη σειρά εκτέλεσης κάθε περίπτωσης, στο παρακάτω πίνακα(Πίνακας 4.11).

Πίνακας 4.11: BM25 Αποτελέσματα MRR Solr lemmatization με KStem για 10.000 άρθρα.

Μέθοδος Lemmatization	MRR	Index	Query
NLTK Lemm	0.4132	00:00:30	00:05:16
SynonymFilterNTLK	0.4194	00:00:17	00:05:14
SynonymFilterWord2Vec	0.4193	00:00:15	00:05:03
SynonymFilterGlove	0.4193	00:00:15	00:05:04

Παρατηρώντας το πίνακα, μπορούμε να εξάγουμε το συμπέρασμα ότι το ο συνδυασμός lemmatization και stemming με χρήση του BM25 δε βοήθησε στη βελτίωση των αποτελεσμάτων, καθώς όλα τα αποτελέσματα εμφάνισαν μείωση στο MRR. Η λημματοποίηση με την εφαρμογή του SynonymGraphFilterFactory και

με τους τρεις τρόπους, παρουσιάζει όμως καλύτερα αποτελέσματα από την απευθείας με την εφαρμογή του ntlk lemmatizer και σε αυτή την εφαρμογή.

Επίσης ο χρόνος αυξήθηκε σε όλες τις περιπτώσεις λόγω της προσθήκης ενός ακόμη φίλτρου, αυτό του KStem, κάτι το οποίο ήταν αναμενόμενο. Οι χρόνοι συνολικής προετοιμασίας για κάθε μία από τις 4 εφαρμογές παρουσιάζονται στο πίνακα 4.12,

Πίνακας 4.12: BM25 Χρόνοι lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	Index + Query	File Creation	Model Preparation	Sum
NLTK Lemm	00:05:46	00:00:00	00:00:00	00:05:46
SynonymFilterNLTK	00:05:31	00:00:30	00:00:00	00:06:01
SynonymFilterWord2Vec	00:05:18	00:00:05	00:00:18	00:05:41
SynonymFilterGlove	00:05:19	00:00:05	00:00:40	00:06:04

όπου είναι περισσότερο ξεκάθαρο να θεωρήσουμε ως καλύτερη επιλογή από πλευράς χρόνου και MRR το lemmatization με το λεξιλόγιο του Word2Vec.

4.1.4.2 TF-IDF

Όσον αφορά την εκτέλεση των πειραμάτων με χρήση TF-IDF, για τις δοκιμές απλού lemmatization, όλα τα απαραίτητα στοιχεία φαίνονται στον πίνακα 4.13.

Πίνακας 4.13: TF-IDFΑποτελέσματα MRR Solr lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	MRR	Index	Query
NLTK Lemm	0.4102	00:00:24	00:04:55
SynonymFilterNTLK	0.4159	00:00:15	00:05:00
SynonymFilterWord2Vec	0.4158	00:00:14	00:05:03
SynonymFilterGlove	0.4158	00:00:15	00:05:03

Όσον αφορά τα αποτελέσματα MRR, η χρήση του SynonymGraphFilterFactory και με τους τρεις τρόπους, δίνει εμφανώς καλύτερα αποτελέσματα από την απευθείας χρήση του ntlk lemmatizer στα δεδομένα που εισάγουμε και αναζητούμε στο σύστημα μας.

Ο χρόνος στο πίνακα 4.13, αφορά μόνο τις στιγμές λειτουργίας του Solr. Αν θέλουμε να εκτιμήσουμε τη προσφορά κάθε τεχνικής lemmatization υπολογίζοντας όλο το χρόνο που χρειάζεται για να εκτελεστεί επιτυχώς, αρκεί να κοιτάξουμε τις μετρήσεις του πίνακα 4.14.

Ακόμη μια φορά την ισορροπία ταχύτητας και αποτελεσματικότητας τη προσφέρει η μέθοδος lemmatization με το λεξιλόγιο του Word2Vec.

Ο επόμενος πίνακας 4.15 παρουσιάζει τα σχετικά αποτελέσματα αφού χρησιμοποιήσαμε και το KStem Filter.

Πίνακας 4.14: TF-IDF Χρόνοι lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	Index + Query	File Creation	Model Preparation	Sum
NLTK Lemm	00:05:19	00:00:00	00:00:00	00:05:19
SynonymFilterNLTK	00:05:15	00:00:30	00:00:00	00:05:45
SynonymFilterWord2Vec	00:05:17	00:00:05	00:00:18	00:05:40
SynonymFilterGlove	00:05:18	00:00:05	00:00:40	00:06:03

Πίνακας 4.15: TF-IDF Αποτελέσματα MRR Solr lemmatization με KStem για 10.000 άρθρα.

Μέθοδος Lemmatization	MRR	Index	Query
NTLK Lemm	0.4078	00:00:30	00:05:16
SynonymFilterNTLK	0.4171	00:00:17	00:05:14
SynonymFilterWord2Vec	0.4171	00:00:15	00:04:59
SynonymFilterGlove	0.4171	00:00:15	00:04:59

Παρατηρώντας τον μπορούμε να καταλάβουμε ότι στη περίπτωση του TF-IDF ο συνδυασμός lemmatization και stemming οδήγησε σε βελτίωση των αποτελεσμάτων, διατηρώντας όμως τη σειρά αποτελεσματικότητας, όπου το SynonymGraphFilterFactory παραμένει αποδοτικότερο από την απευθείας με την εφαρμογή του ntlk lemmatizer.

Λόγω της προσθήκης του KStem φίλτρου, έχουμε σε κάθε περίπτωση μια μικρή χρονική αύξηση, σε σχέση με την προηγούμενη εφαρμογή.

Ο χρόνος στο πίνακα 4.13, παρουσιάζει μετρήσεις μόνο κατά τη διάρκεια λειτουργίας του Solr. Οι χρόνοι στο πίνακα 4.16 παρουσιάζουν τους τελικούς χρόνους, οι οποίοι περιλαμβάνουν οποιαδήποτε προετοιμασία αρχείων.

Πίνακας 4.16: TF-IDF Χρόνοι lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	Index + Query	File Creation	Model Preparation	Sum
NLTK Lemm	00:05:46	00:00:00	00:00:00	00:05:46
SynonymFilterNLTK	00:05:31	00:00:30	00:00:00	00:06:01
SynonymFilterWord2Vec	00:05:14	00:00:05	00:00:18	00:05:37
SynonymFilterGlove	00:05:14	00:00:05	00:00:40	00:05:59

Με αυτά τα επιπλέον χρονικά στοιχεία, η επιλογή του αρχείου lemmatization από το λεξιλόγιο Word2Vec αποτελεί τη καλύτερη μέθοδο, λόγω υψηλότερου MRR και μικρότερου συνολικού χρόνου.

4.1.4.3 Σύγκριση BM25 και TF-IDF

Ο πίνακας 4.17 παρουσιάζει συγκεντρωτικά τα αποτελέσματα lemmatization για BM25 και TF-IDF

Πίνακας 4.17: Σύγκριση MRR BM25 και TF-IDF δοκιμών Solr lemmatization για 10.000 άρθρα.

Μέθοδος Lemmatization	BM25 Δοκιμή 1	TF-DIF Δοκιμή 1	BM25 Δοκιμή 2	TD-IDF Δοκιμή 2
NLTK Lemm	0.4151	0.4102	0.4132	0.4078
SynonymFilterNLTK	0.4217	0.4159	0.4194	0.4171
SynonymFilterWord2Vec	0.4214	0.4158	0.4193	0.4171
SynonymFilterGlove	0.4214	0.4158	0.4193	0.4171

Η πρώτη παρατήρηση είναι ότι καλύτερα αποτελέσματα και στις δοκιμές σκέτου lemmatization (δοκιμή 1) και με τον συνδυασμό lemmatization και KStem stemming (δοκιμή 2) μας τα δίνει συνολικά το BM25.

Όσον αφορά την εφαρμογή του KStem φαίνεται να λειτουργεί αρνητικά για το BM25, καθώς μειώνονται τα αποτελέσματα στη δεύτερη δοκιμή, ενώ το TF-IDF αντίθετα, έχει καλύτερα αποτελέσματα όταν εφαρμόζεται συνδυαστικά lemmatization και stemming.

Πιο συγκεκριμένα όμως το καλύτερο MRR (0.4217) το βρίσκουμε στο πρώτο γύρο δοκιμών, με σκέτη εφαρμογή του SynonymGraphFilterFactory, χωρίς stemming, από το BM25. Αυτό μας το δίνει η χρήση του αρχείου που προήλθε από lemmatization στις λέξεις της συλλογής των 10.000 άρθρων (με αποτέλεσμα 6020 μοναδικές λημματοποιημένες λέξεις) και το δεύτερο καλύτερο MRR (0.4214) τα αρχεία που προήλθαν από lemmatization στο λεξιλόγιο των Word2Vec και Glove (3022 μοναδικές λημματοποιημένες λέξεις).

Σε πρώτη σκέψη η καλύτερη επιλογή θα ήταν αυτή το καλύτερο MRR (0.4217), δηλαδή να χρησιμοποιήσουμε το αρχείο όπου έγινε κατευθείαν λημματοποίηση στις λέξεις της συλλογής (6020 μοναδικές λημματοποιημένες λέξεις).

Μελετώντας όμως τη περίπτωση του αρχείου λημματοποίησης από το λεξιλόγιο του Word2Vec (3022 μοναδικές λημματοποιημένες λέξεις), διαπιστώνουμε ότι συγκεκριμένη επιλογή ήταν αρχικά πιο γρήγορη τόσο όσον αφορά το χρόνο indexing και querying (πίνακας 4.9), με διαφορά 1 δευτερόλεπτο, όσο και το συνολικό χρόνο εκτέλεσης (πίνακας 4.10), με διαφορά 8 δευτερολέπτων. Επίσης ήταν μικρότερο το αρχείο κάτι που είναι πλεονέκτημα για μεγάλες συλλογές (σχεδόν μισές λέξεις) και είχε το δεύτερο αποτέλεσμα για διαφορά μόλις 0.0003 μονάδων. Επομένως μπορεί να θεωρηθεί εξίσου ιδανική επιλογή για lemmatization.

4.1.5 Word2Vec

Όπως αναφέραμε και στο κεφάλαιο 2, δεν υπάρχει τρόπος να χρησιμοποιήσουμε Word2Vec στη Solr, παρά μόνο να δουλέψουμε με το SynonymGraphFilterFactory και με ένα αρχείο το οποίο θα αντιστοιχεί κάθε

λέξη του λεξιλογίου του Word2Vec μοντέλου με το cluster στο οποίο ανήκει.

Αναλυτικότερα όσον αφορά το fieldtype στο managed schema της Solr, χρησιμοποιήσαμε για διάσπαση του κειμένου τον Standard Tokenizer, για μετατροπή σε πεζά το LowerCaseFilterFactory και για αφαίρεση των stopwords το StopFilterFactory. Επίσης χρειαστήκαμε και το FlattenGraphFilterFactory κατά το indexing , σύμφωνα με τις οδηγίες του Solr, για τη χρήση του SynonymGraphFilterFactory.

Για να δημιουργήσουμε τα απαιραίτητα αρχεία, δουλέψαμε με κάποιες ειδικές βιβλιοθήκες της python, τις gensim, nltk, sklearn.
Μέσω nltk αφαιρέσαμε τα stopwords από το λεξιλόγιο του Word2Vec.
Με τη gensim δημιουργήσαμε το μοντέλο του Word2Vec.
Με το sklearn χωρίσαμε το λεξιλόγιο σε clusters με χρήση του KMeans αλγόριθμου.

Αναλυτικά τα βήματα που ακολουθήσαμε σε καθορισμένη σειρά ήταν:

1. Διάβασμα από το αρχείο.
2. Αφαίρεση stopwords και προετοιμασία για εισαγωγή στο λεξιλόγιο του Word2Vec.
3. Δημιουργία και εκπαίδευση του μοντέλου Word2Vec.
4. Διαχωρισμός σε clusters με εκτέλεση KMeans
5. Καταγραφή των αποτελεσμάτων σε αρχείο.

Συγκεκριμένα η εκπαίδευση του μοντέλου έγινε με τις εξής ρυθμίσεις:

- size = 200, όπου είναι το μέγεθος της διάστασης του παραγόμενου πίνακα
- windows = 5, μέγιστη απόσταση της τωρινής με την λέξης πρόβλεψης μέσα σε μια πρόταση
- epochs = 10, οι επαναλήψεις εκπαίδευσης πάνω στο σώμα του λεξιλογίου
- workers = 4, αριθμός thread που θα χρησιμοποιηθούν για ταχύτερη εκτέλεση (4 στην περίπτωση μας)
- min count = 5, όριο σύμφωνα με το οποίο αγνοούμε όλες τις λέξεις με συνολική συχνότητα μικρότερη από αυτή.

Το μοντέλο Word2Vec δημιουργήθηκε μία φορά, το αποθηκεύσαμε στο δίσκο και στη συνέχεια το φορτώναμε κάθε φορά που θέλαμε να δημιουργήσουμε clusters.
Ο χρόνος για τη δημιουργία και αποθήκευση του μοντέλου, με ανάγνωση από το αρχείο ήταν 18 δευτερόλεπτα με χρήση και των τεσσάρων threads του υπολογιστή.

Αφού έχουμε δημιουργήσει το μοντέλο, στη συνέχεια πραγματοποιούμε το clustering. Χρησιμοποιήσαμε είπαμε τον αλγόριθμο KMeans (K μέσων) με τις προκαθορισμένες ρυθμίσεις και παραμέτρους. Ακόμη, ο αριθμός των cluster με τον οποίο πειραματιστήκαμε ήταν 1000, 2000, 3000, 3500 και 5000.

Τα αποτελέσματα τα γράφουμε σε ένα αρχείο το οποίο είναι δομημένο όπως φαίνεται παρακάτω:

$word1 \Rightarrow cluster2$

$word2 \Rightarrow cluster4$

$word3 \Rightarrow cluster100$

$word4 \Rightarrow cluster250$

$word5 \Rightarrow cluster100$

Σύμφωνα με τέτοιας μορφής αρχείο, το SynonymGraphFilterFactory αντικαθιστούσε τη κάθε λέξη με το cluster στο οποίο άνηκε.

Οι χρόνοι για τη δημιουργία των clusters για κάθε μία από τις 5 αυτές περιπτώσεις μαζί με την εγγραφή στο κατάλληλο αρχείο ήταν οι ακόλουθοι:

- 1000 clusters: 2 λεπτά και 59 δευτερόλεπτα
- 2000 clusters: 7 λεπτά και 16 δευτερόλεπτα
- 3000 clusters: 10 λεπτά και 35 δευτερόλεπτα
- 3500 clusters: 12 λεπτά και 06 δευτερόλεπτα
- 5000 clusters: 15 λεπτά και 42 δευτερόλεπτα

Για να δημιουργήσουμε συνολικό χρόνο όμως για τις 5 αυτές περιπτώσεις πρέπει να προσμετρήσουμε και το χρόνο που χρειάζεται για τη δημιουργία του μοντέλου Word2Vec. Οι τελικοί χρόνοι που χρειάζονται για τη δημιουργία των πέντε αρχείων που περιέχουν τα clusters φαίνονται στο πίνακα 4.18

Πίνακας 4.18: Τελικοί χρόνοι για δημιουργία αρχείων clustering για 10.000 άρθρα.

Άρθμός Clusters	Χρόνος
1000	00:03:17
2000	00:07:34
3000	00:10:53
3500	00:12:24
5000	00:15:00

Αφού ολοκληρώθηκε αυτή η διαδικασία, επόμενο βήμα είναι οι δοκιμές στο Solr με BM25 και TF-IDF.
Εκτελέσαμε δύο γύρους δοκιμών, ένα με απλή εφαρμογή του SynonymGraphFilterFactory και ένα με επιπρόσθετη εφαρμογή του KStem φίλτρου για stemming, τα αποτελέσματα των οποίων σχολιάζονται στις επόμενες υποενότητες.

4.1.5.1 BM25

Στο πρώτο γύρο δοκιμών προέκυψαν τα αποτελέσματα του πίνακα 4.19

Πίνακας 4.19: BM25 Αποτελέσματα MRR Solr Word2Vec για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.3704	00:00:15	00:05:24
2000	0.3943	00:00:15	00:05:18
3000	0.3997	00:00:15	00:05:19
3500	0.4025	00:00:15	00:05:19
5000	0.4068	00:00:15	00:05:17

Εκ πρώτης όψεως, με την αύξηση του αριθμού των clusters βελτιώνεται τα αποτεσμέτα MRR. Παρατηρώντας όμως τη βελτίωση από 3500 σε 5000 clusters, αλλά και γενικά την πορεία του MRR από τα 1000 clusters και μετά, συνειδητοποιούμε ότι είναι πολύ μικρή η διαφορά αυτή ώστε να συνεχίσουμε να κάνουμε δοκιμές με μεγαλύτερο αιριθμό clusters.

Όσον αφορά τα νούμερα του χρόνου, μετρήσαμε μόνο τη λειτουργία indexing και querying που πραγματοποιείται στο Solr με τη χρήση του εκάστοτε αρχείου. Φέρνοντας κατά νου, τις μετρήσεις του πίνακα 4.18, μπορούμε εύκολο να καταλάβουμε ότι χρονικά ξεφεύγουμε πολύ σε σχέση με τις προηγούμενες τεχνικές που δοκιμάσαμε. Αυτό είναι απόλυτα λογικό καθώς για να δημιουργηθούν τα 5 αρχεία των δοκιμών, προηγείται προεργασία και εκπαίδευση των μοντέλων.

Επιστέφοντας όμως στα χρονικά αποτελέσματα του πίνακα 4.19, διαπιστώνουμε ότι οι χρόνοι εισαγωγής είναι πολύ κοντά χωρίς ιδιαίτερες διαφοροποιήσεις, εκτός από την περίπτωση των 1000 clusters. Στην συγκεκριμένη εφαρμογή έχουμε τον πιο αργό χρόνο επαναληπτικής αναζήτησης (5 λεπτά και 24 δευτερόλεπτα), με διαφορά 5 δευτερολέπτων από το δεύτερο μεγαλύτερο (5 λεπτά και 19 δευτερόλεπτα). Επίσης η περίπτωση των 5000 clusters, αποδείχθηκε και η πιο γρήγορη (5 λεπτά και 17 δευτερόλεπτα).

Στο δεύτερο γύρο δοκιμών με την εφαρμογή του KStem οδηγηθήκαμε στα ακόλουθα αποτελέσματα: Πίνακας 4.20

Σε αυτή τη περίπτωση η εφαρμογή του KStem δεν επέφερε κάποια ουσιώδης βελτίωση των αποτελεσμάτων MRR, καθώς τα αποτελέσματα MRR παρέμεινα ίδια για τα 2000, 3000, 3500 clusters και για τα 1000 και 5000 αυξήθηκαν σε 0.3808

Πίνακας 4.20: BM25 Αποτελέσματα MRR Solr Word2Vec KStem για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.3708	00:00:16	00:05:15
2000	0.3945	00:00:16	00:05:20
3000	0.3997	00:00:16	00:05:12
3500	0.4025	00:00:16	00:05:10
5000	0.4071	00:00:16	00:05:09

και 0.4071 αντίστοιχα.

Χρονικά όμως, εκτός από την κοινή αύξηση του ενός δευτερολέπτου κατά το indexing, οι χρόνοι της επαναληπτικής αναζήτησης διαφοροποιήθηκαν χωρίς να μπορεί να εντοπιστεί κάποιο μοτίβο. Για 2000 clusters είχαμε μια λογική αύξηση λόγω του επιπλέον φίλτρου KStem, αλλά για τις υπόλοιπες περιπτώσεις είχαμε μειώσεις σε σχέση με την εφαρμογή μόνο του SynonymGraphFilterFactory , στην προηγούμενη δοκιμή.
Και σε αυτές τις χρονικές μετρήσεις δεν συμπεριλάβαμε τις μετρήσεις του πίνακα 4.18, ώστε να επικεντρωθούμε κυρίως στη λειτουργία και αποδοτικότητα του Solr, έχοντας ήδη όλα τα απαραίτητα αρχεία στη διάθεση μας.

4.1.5.2 TF-IDF

Τα αποτελέσματα της εφαρμογής μόνο του SynonymGraphFilterFactory παρουσιάζονται στο πίνακα 4.21

Πίνακας 4.21: TF-IDF Αποτελέσματα MRR Solr Word2Vec για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.3777	00:00:15	00:05:07
2000	0.3912	00:00:16	00:05:09
3000	0.3954	00:00:15	00:05:05
3500	0.3973	00:00:15	00:05:06
5000	0.4010	00:00:15	00:05:09

Παρατηρώντας το πίνακα, εύλογα καταλήγουμε στο συμπέρασμα, ότι αυξάνοντας τον αριθμό των clusters, οδηγούμαστε σε μεγαλύτερο MRR. Παρ' όλα αυτά ο ρυθμός βελτίωσης του MRR υποδεικνύει ότι μεγαλύτερος αριθμός clusters από 5000 δεν θα οδηγήσει σε πολύ καλύτερο αποτέλεσμα.

Από χρονική άποψη πάντως, ο αριθμός clusters δεν φαίνεται να επηρεάζει κάπου στην ταχύτητα εκτέλεσης και ούτε κάποια ακολουθία μπορεί να εντοπιστεί. Τα

αποτελέσματα είναι παρόμοια και δεν βοηθάνε στη δημιουργία χρονικής κατάταξης ώστε να διακρίνουμε κάποιον τρόπο ως πιο γρήγορο ή πιο αργό.

Η εφαρμογή του φίλτρου για stemming KStem μετά το SynonymGraphFilterFactory είχε τα αποτελέσματα του πίνακα 4.22

Πίνακας 4.22: BM25 Αποτελέσματα MRR Solr Word2Vec KStem για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.3779	00:00:16	00:05:05
2000	0.3916	00:00:16	00:05:04
3000	0.3951	00:00:16	00:05:06
3500	0.3972	00:00:16	00:05:05
5000	0.4010	00:00:16	00:05:05

Η εφαρμογή του KStem δεν μας δίνει τα περιθώρια για μια καθολική εκτίμηση της εφαρμογής του.
Αποτελέσματα όπως αυτά που προέκυψαν με χρήση 1000 και 2000 clusters εκδήλωσαν έστω μια μικρή βελτίωση (0.3779 και 0.3916 αντίστοιχα), για 3000 και 3500 clusters είχαμε πολύ μικρή μείωση (0.3951 και 0.3972) και στη τελευταία περίπτωση των 5000 clusters το αποτέλεσμα παρέμεινε ως είχε.

Επίσης στις χρονικές μετρήσεις παρατηρούμε αύξηση κατά ένα δευτερόλεπτο σε όλες τις περιπτώσεις στο Indexing και στο querying αύξηση κατά ένα δευτερόλεπτο μόνο στη περίπτωση των 3000 clusters, ενώ στις υπόλοιπες μείωση.

Οι χρονικές μετρήσεις και στους δύο γύρους δοκιμών του TF-IDF αφορούν την εκτέλεση του Solr. Οι χρόνοι που παρουσιάζονται στο πίνακα 4.18 δίνουν μια εικόνα του χρονικού κόστους απόκτησης των 5 αρχείων που περιέχουν το λεξιλόγιο του Word2Vec χωρισμένο σε clusters. Δεν συμπεριλαμβάνονται στο πίνακα 4.22 καθώς θέλουμε να εστιάσουμε τη προσοχή μας στην αποτελεσματικότητα της μεθόδου αυτής στο Solr και μετέπειτα να υπολογίσουμε και αν αξίζει το κόστος υλοποίησης, σύμφωνα με τα παραγόμενα αποτελέσματα.

4.1.5.3 Σύγκριση BM25 και TF-IDF

Ο πίνακας 4.23 περιέχει όλες τι μετρήσεις MRR για BM25 και TF-IDF

Το καλύτερο MRR το παίρνουμε από τη χρήση του αρχείου με τα 5000 clusters στην εκτέλεση με BM25. Είναι η πρώτη φορά μέχρι στιγμής όπου η χρήση KSTem σε εφαρμογή του BM25 δεν είχε αρνητικό πρόσημο, καθώς στη περίπτωση μας όλα τα αποτελέσματα του BM25 είχαν τα ίδια (περίπτωση 3000 και 3500 clusters) ή καλύτερα αποτελέσματα (περίπτωση 1000, 2500 και 5000 clusters). |

Το TF-IDF φαίνεται να είναι πιο αποτελεσματικό από το BM25 μόνο στη περίπτωση των 1000 clusters, και με και χωρίς εφαρμογή stemming, αλλά για τις υπόλοιπες 4 επιλογές clusters δεν έχει καταφέρει να ξεπεράσει τα αποτελέσματα

Πίνακας 4.23: Σύγκριση MRR BM25 και TF-IDF δοκιμών Solr Word2Vec για 10.000 άρθρα.

Clusters	BM25 Δοκιμή 1	TF-DIF Δοκιμή 1	BM25 Δοκιμή 2	TD-IDF Δοκιμή 2
1000	0.3704	0.3777	0.3708	0.3779
2000	0.3943	0.3912	0.3945	0.3916
3000	0.3997	0.3954	0.3997	0.3951
3500	0.4025	0.3973	0.4025	0.3972
5000	0.4068	0.4010	0.4071	0.4010

του BM25.

Γενικά η εφαρμογή του Word2Vec με τη τεχνική των clusters είναι πολύ χρονοβόρα όπως αναφέραμε στην αρχή της υποενότητας και τα αποτελέσματα MRR έστω και στην καλύτερη περίπτωση δεν ήταν καλύτερα απ' ότι έχουμε δει μέχρι στιγμής από τις προηγούμενες μεθόδους (stemming και lemmatization).

Τέλος για μια ακόμη φορά το καλύτερο αποτέλεσμα μας το προσφέρει το BM25, και μάλιστα με εφαρμογή stemming η οποία βελτίωση τα αποτελέσματα, εν αντιθέσει με αυτά που είχαμε δει μέχρι στιγμής.

4.1.6 Glove

Όπως με το Word2Vec, έτσι και με το Glove, δουλέψαμε με το SynonymGraphFilterFactory και με ένα αρχείο το οποίο θα αντιστοιχεί κάθε λέξη του λεξιλογίου του Glove μοντέλου με το cluster στο οποίο ανήκει.

Χρημοποιήσαμε στο σύνολο τέσσερις βιβλιοθήκες της python, τις glove, gensim, nltk, sklearn.
Όπου μέσω nltk αφαιρέσαμε τα stopwords από το λεξιλόγιο του Glove.
Με τη glove δημιουργήσαμε το μοντέλο του Glove.
Με τη gensim μετατρέψαμε το μοντέλο του Glove στη μορφή του Word2Vec.
Και τέλος με το sklearn χωρίσαμε το λεξιλόγιο σε clusters με χρήση του KMeans αλγόριθμου.

Τα βήματα που ακολουθήσαμε ήταν με τη σειρά που περιγράφονται παρακάτων:

1. Διάβασμα από το αρχείο.
2. Αφαίρεση stopwords και προετοιμασία για εισαγωγή στο λεξιλόγιο του Glove.
3. Δημιουργία και εκπαίδευση του μοντέλου Glove.
4. Αποθήκευση του μοντέλου Glove και αντιγραφή του σε ειδικά διαμορφωμένο αρχείο.
5. Μετατροπή του αρχείου σε μοντέλο Word2Vec.

6. Διαχωρισμός σε clusters με εκτέλεση KMeans

7. Καταγραφή των αποτελεσμάτων σε αρχείο.

Συγκεκριμένα η εκπαίδευση του μοντέλου έγινε με τις ίδιες ρυθμίσεις που είχαμε χρησιμοποιήσει για το Word2Vec. Η μοναδική σημείωση είναι ότι επειδή η βιβλιοθήκη glove δεν υποστήριζε τη παράμετρο min count για να θέσουμε όριο ελάχιστης συχνότητας εμφάνισης, αφαιρέσαμε εμείς τις λέξεις με μικρότερη συχνότητα εμφάνισης από το όριο 5 , πριν γίνει η εκπαίδευση του μοντέλου. Οι υπόλοιπες παράμετροι έχουν ως εξής:

- size = 200, όπου είναι το μέγεθος της διάστασης του παραγόμενου πίνακα
- windows = 5, μέγιστη απόσταση της τωρινής με την λέξης πρόβλεψης μέσα σε μια πρόταση
- epochs = 10, οι επαναλήψεις εκπαίδευσης πάνω στο σώμα του λεξιλογίου
- workers = 4, αριθμός thread που θα χρησιμοποιηθούν για ταχύτερη εκτέλεση (4 στην περίπτωση μας)

Το μοντέλο Glove αφού το δημιουργήσαμε το αποθηκεύσαμε στο δίσκο, και στη συνέχεια δημιουργήσαμε ένα αρχείο το οποίο θα ήταν συγκεκριμένης μορφής. Το αρχείο αυτό ήταν ως εξής:

word1 number number number number ..... number

word2 number number number number ..... number

word3 number number number number ..... number

word4 number number number number ..... number

word5 number number number number ..... number

Όπου δηλαδή σε κάθε σειρά γράφουμε στην αρχή τη λέξη και στη συνέχεια τα στοιχεία του πίνακα του Glove που αντιστοιχούν σε αυτή τη λέξη. Τη συγκεκριμένη μορφή αρχείου, μπορεί η βιβλιοθήκη gensim να μετατρέψει σε αντίστοιχο Word2Vec μοντέλο.
Ο χρόνος για τη δημιουργία αποθήκευση και μετατροπή του μοντέλου σε κατάλληλο αρχείο, με ανάγνωση από τη συλλογή ήταν 40 δευτερόλεπτα

Αφού πια είχαμε το αρχείο έτοιμο, μέσω gensim δημιουρησαμε το Word2Vec μοντέλο και πραγματοποιήσαμε το clustering. Με τον αλγόριθμο KMeans (Κ μέσων), με χρήση των προκαθορισμένων ρυθμίσεων και παραμέτρων, δημιουργήσαμε αρχεία για αριθμό clusters 1000, 2000, 3000, 3500 και 5000 αντίστοιχα.

Τα αποτελέσματα καταγράφηκα σε αρχείο το οποίο μπορεί να χρησιμοποιηθεί σε από το SynonymGraphFilterFactory :

$word1 \Rightarrow cluster2$

$word2 \Rightarrow cluster4$

$word3 \Rightarrow cluster100$

$word4 \Rightarrow cluster250$

$word5 \Rightarrow cluster100$

Το SynonymGraphFilterFactory έχοντας πρόσβαση στο αντίστοιχο αρχείο της κάθε περίπτωσης, πραγματοποίησε την αλλαγή της κάθε λέξης με το cluster στο οποίο άνηκε.

Οι χρόνοι δημιουργίας των clusters για τις 5 περιπτώσεις, μαζί με καταγραφή σε αρχείο ήταν:

- 1000 clusters: 3 λεπτά και 39 δευτερόλεπτα
- 2000 clusters: 7 λεπτά και 10 δευτερόλεπτα
- 3000 clusters: 10 λεπτά και 35 δευτερόλεπτα
- 3500 clusters: 09 λεπτά και 38 δευτερόλεπτα
- 5000 clusters: 11 λεπτά και 43 δευτερόλεπτα

Για να δημιουργήσουμε συνολικό χρόνο όμως για τις 5 περιπτώσεις θα πρέπει να προσμετρήσουμε και το χρόνο που χρειάζεται για τη δημιουργία του μοντέλου Glove. Οι τελικοί χρόνοι που χρειάζονται για τη δημιουργία των πέντε αρχείων που περιέχουν τα clusters φαίνονται στο πίνακα 4.24

Πίνακας 4.24: Τελικοί χρόνοι για δημιουργία αρχείων clustering για 10.000 άρθρα.

Άρθμός Clusters	Χρόνος
1000	00:04:19
2000	00:07:50
3000	00:11:15
3500	00:13:18
5000	00:12:23

Αφού ολοκληρώσαμε όλα τα απαραίτητα βήματα, στη συνέχεια πραγματοποιήσαμε τα πειράματα μας στο Solr με BM25 και TF-IDF.
Εκτελέσαμε δύο γύρους δοκιμών, ένα με απλή εφαρμογή του SynonymGraphFilterFactory και ένα με επιπρόσθετη εφαρμογή του KStem φίλτρου για stemming.

4.1.6.1 BM25

Τα αποτελέσματα μας με χρήση μόνο του SynonymGraphFilterFactory παρουσιάζονται στο πίνακα 4.25

Πίνακας 4.25: BM25 Αποτελέσματα MRR Solr Glove για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.2462	00:00:15	00:05:55
2000	0.2995	00:00:15	00:06:05
3000	0.3192	00:00:15	00:06:03
3500	0.3329	00:00:16	00:05:59
5000	0.3675	00:00:15	00:05:44

Το MRR διαφοροποιείται αρκετά ανάμεσα στις 5 περιπτώσεις. Σαφέστατα το καλύτερο αποτέλεσμα το έχει η υλοποίηση με τα 5000 clusters. Από την πρώτη εφαρμογή των 1000 clusters έχουμε αύξηση 0.12 μονάδων (από 0.2462 σε 0.3675). Παρ' όλα αυτά, τα αποτελέσματα τόσο στο σύνολο άλλο τόσο και στη καλύτερη περίπτωση είναι πολύ πιο κάτω από τις χαμηλότερες τιμές MRR που έχουμε συναντήσει στις προηγούμενες δοκιμές.

Ο χρόνος indexing είναι στα ίδια επίπεδα για όλες τις περιπτώσεις, ενώ στο querying παρατηρούμε ότι η τελευταία περίπτωση με τα 5000 clusters είναι και η πιο γρήγορη, όπου σε συνδυασμό με το καλύτερο MRR, της δίνεις της θέσης της καλύτερης περίπτωσης.

Ο χρόνος του querying για τις άλλες τέσσερις περιπτώσεις δεν παρουσιάζει κάποια συγκεκριμένα χαρακτηριστικά αυξομείωσης ώστε να αιτιολογήσουμε γιατί προκύπτουν οι συγκεκριμένοι χρόνοι.

Για να επικεντρωθούμε στην συμπεριφορά του Solr αναφερόμαστε στους χρόνους μόνο της λειτουργίας του και όχι της προετοιμασίας όπως αναφέρονται εδώ (Πίνακας 4.24).

Με την εφαρμογή του KStem μετά το SynonymGraphFilterFactory οδηγηθήκαμε στα αποτελέσματα του πίνακα 4.26

Πίνακας 4.26: BM25 Αποτελέσματα MRR Solr Glove KStem για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.2461	00:00:16	00:05:44
2000	0.2993	00:00:16	00:05:56
3000	0.3190	00:00:16	00:05:50
3500	0.3325	00:00:16	00:05:49
5000	0.3678	00:00:16	00:05:34

Η μόνη περίπτωση που διακρίνουμε μια αύξηση στο MRR, είναι της πέμπτης

των 5000 clusters, όπου η αύξηση μπορεί να θεωρηθεί και αμελητέα (0.3675 σε 0.3678) αλλά είναι η μόνη καθώς στις υπόλοιπες είχαμε μείωση .

Το indexing είναι σταθεροποιημένο στα 16 δευτερόλεπτα για όλες τις περιπτώσεις, ενώ στο queyring ακολουθείτε μείωση χρόνου επίσης για όλες. Οι χρόνοι αφορούν μόνο τη λειτουργία του Solr και όχι τη διαδικασία για τη δημιουργία των αρχείων όπως φαίνονται στο πίνακα 4.24.

Συμπερασματικά μπορούμε να χαρακτηρίσουμε τη περίπτωση των 5000 clusters ως την αποδοτικότερη και χρονικά αλλά και όσον αφορά το MRR.

4.1.6.2 TF-IDF

Για εφαρμογή μόνο του SynonymGraphFilterFactory έχουμε τα ακόλουθα αποτελέσματα (Πίνακας 4.27)

Πίνακας 4.27: TF-IDF Αποτελέσματα MRR Solr Glove για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.2455	00:00:15	00:05:27
2000	0.3126	00:00:15	00:05:40
3000	0.3294	00:00:15	00:05:40
3500	0.3410	00:00:15	00:05:40
5000	0.3719	00:00:15	00:05:26

Παρατηρώντας το πίνακα, μπορούμε να διακρίνουμε ότι το μεγαλύτερο MRR το αποκτάμε όταν χρησιμοποιήσουμε το αρχείο των 5000 clusters. Επίσης ευδιάκριτη είναι είναι και η βελτίωση από τη πρώτη εφαρμογή με 1000 clusters μέχρι αυτήν με τα 5000 (αύξηση 0.13 μονάδων). Τα αποτελέσματα παραμένουν σε χαμηλό επίπεδο όμως παρά τη βελτίωση.

Από χρονική άποψη, δε μπορούμε να θεωρήσουμε ότι ο αριθμός clusters επηρεάζει κατά κάποιο τρόπο την ταχύτητα εκτέλεσης καθώς έχουμε ίδιο χρόνο indexing για όλες τις περιπτώσεις (15 δευτερόλεπτα). Επίσης τα 5000 clusters έχουν την μικρότερο χρόνο querying με 5 λεπτά 26 δευτερόλεπτα και μετά τα 1000 clusters με 5 λεπτά 27 δευτερόλεπτα, όταν για τις υπόλοιπες περιπτώσεις έχουμε σταθερό χρόνο τα 5 λεπτά και 40 δευτερόλεπτα.

Για ακόμη μία φορά η χρησιμοποίηση του αρχείου με τα 5000 clusters φαίνεται ως η καλύτερη μεταξύ άλλων.

Η εφαρμογή του φίλτρου stemming με τη χρήση KStem μετά το SynonymGraphFilterFactory είχε τα αποτελέσματα του πίνακα 4.28

Η εφαρμογή του KStem στις περιπτώσεις με 1000, 2000 και 3000 clusters επέφερε μια μικρή βελτίωση στο MRR, ενώ για τα 3500 και 5000 είχαμε πτωτικά αποτελέσματα. Γενικότερα όμως οι αυξομειώσεις δεν ήταν μεγάλου βαθμού.

Στο χρονικό κομμάτι όμως, είχαμε σταθερή αύξηση ενός δευτερολέπτου για το indexing και στο querying είχαμε ίδια ή μικρότερα ποσοστά. Ειδικότερα στη

Πίνακας 4.28: BM25 Αποτελέσματα MRR Solr Glove KStem για 10.000 άρθρα.

Άρθμός Clusters	MRR	Index	Query
1000	0.2458	00:00:16	00:05:26
2000	0.3127	00:00:16	00:05:39
3000	0.3298	00:00:16	00:05:40
3500	0.3409	00:00:16	00:05:35
5000	0.3718	00:00:16	00:05:24

περίπτωση των 3000 clusters είχαμε διατήρηση του χρόνου (5 λεπτά και 40 δευτερόλεπτα) ενώ σε όλες τις υπόλοιπες είχαμε μείωση.

Οι χρόνοι δεν περιλαμβάνουν τη προετοιμασία των αρχείων, η οποία αναφέρεται στο πίνακ 4.28

Η περίπτωση με τα 5000 clusters ήταν αυτή με το μικρότερο χρόνο (5 λεπτά και 24 δευτερόλεπτα) και αυτή με το μεγαλύτερο MRR (0.3718) , επομένως μπορεί να χαρακτηριστεί η καλύτερη επιλογή από αυτές τις πέντε.

4.1.6.3 Σύγκριση BM25 και TF-IDF

Ο πίνακας 4.29 περιέχει όλες τι μετρήσεις MRR για BM25 και TF-IDF

Πίνακας 4.29: Σύγκριση MRR BM25 και TF-IDF δοκιμών Solr Glove για 10.000 άρθρα.

Clusters	BM25 Δοκιμή 1	TF-DIF Δοκιμή 1	BM25 Δοκιμή 2	TD-IDF Δοκιμή 2
1000	0.2462	0.2455	0.2461	0.2458
2000	0.2995	0.3126	0.2993	0.3127
3000	0.3192	0.3294	0.3190	0.3298
3500	0.3329	0.3410	0.3325	0.3409
5000	0.3675	0.3719	0.3678	0.3718

Όπως φαίνεται από τα αποτελέσματα MRR, τη καλύτερη εφαρμογή την επιτυγχάνουμε με χρήση του αρχείου των 5000 clusters , σε εκτέλεση με TF-IDF, μόνο με SynonymGraphFilterFactory και χωρίς stemming (MRR ίσο με 0.3719). Επίσης το δεύτερο καλύτερο MRR, πάλι της στήλης του TF-IDF, είναι το αποτέλεσμα συνδυαστικής εφαρμογής του SynonymGraphFilterFactor με το φίλτρο KStem (0.3718), μόλις 0.0001 μονάδα μικρότερο από το πρώτο σε κατάταξη MRR. Λόγω αυτής τη πολύ μικρής διαφοράς και επειδή σύμφωνα με τους πίνακες 4.27 και 4.28 είναι και 1 δευτερόλεπτο πιο γρήγορος αυτός ο τρόπος, αποτέλει μια εξίσου αξιόλογη επιλογή.

Το BM25, φαίνεται να αποδίδει καλύτερα και στα δυο είδη δοκιμών, μόνο για 1000 clusters, καθώς για 2000, 3000, 3500 και 5000 clusters η επιλογή του TF-IDF επιφέρει καλύτερα αποτελέσματα MRR και για τα δυο είδη δοκιμών.

Η εφαρμογή του Glove με τη τεχνική των clusters αποδείχθηκε περισσότερο χρονοβόρα από αυτή του Word2Vec όπως φαίνεται από τους χρόνους του πίνακα 4.24 στην αρχή της υποενότητας. Ακόμη τα αποτελέσματα MRR ακόμη και στην καλύτερη περίπτωση παραμένουν σε αρκετά χαμηλά επίπεδα είτε επιλέξουμε BM25 είτε TF-IDF.

Τέλος, είναι ξεκάθαρο ότι το καλύτερο αποτέλεσμα για το Glove προέρχεται από την επιλογή του TF-IDF, κατά κύριο λόγο χωρίς stemming, αλλά και ίσως και με εφαρμογή stemming, αναλόγως τη περίπτωση.

4.1.7 Συνολικά αποτελέσματα

Στους πίνακες 4.31 και 4.31 παρουσιάζονται ταξινομημένα τα τα αποτελέσματα κάθε μεθόδου για χρήση BM25 και TF-IDF αντίστοιχα.

Επίσης υπάρχει ένα ακόμα πεδίο το οποίο δε το έχουμε συναντήσει στις προηγούμενες σελίδες. Αυτό είναι το Simple στο οποίο ουσιαστικά κάνουμε διάσπαση κειμένου με το Standard Tokenizer, μετατροπή σε πεζά με το LowerCaseFilterFactory και αφαίρεση των stopwords με το StopFilterFactory. Μια απλή εφαρμογή των βασικών φίλτρων, για να δούμε τη συμπεριφορά του συστήματος μας χωρίς εξειδικευμένες επεμβάσεις πάνω του.

Παρατηρώντας τα αποτελέσματα των πινάκων διαπιστώνουμε ότι αν επιλέξουμε το BM25, υψηλότερο MRR (0.4234) για 10.000 άρθρα θα έχουμε με την εφαρμογή της Simple μεθόδου, όπου ο Standard Tokenizer, το LowerCaseFilterFactory και το StopFilterFactory είναι ότι ακριβώς χρειαζόμαστε για να πάρουμε το καλύτερο δυνατό αποτέλεσμα.

Εάν επιλέξουμε να χρησιμοποιήσουμε το TF-IDF το υψηλότερο MRR (0.4191) θα το πετύχουμε μέσω της δεύτερης δοκιμής του KStem στην οποία το fieldtype ήταν ως εξής δομημένο:

1. WordDelimiterGraphFilterFactory
2. LowerCaseFilterFactory
3. StopFilterFactory
4. KStemFilter
5. FlattenGraphFilterFactory (μόνο στο indexing)

Στο επόμενο βήμα συγκρίναμε τις πιο αποτελεσματικές μεθόδους για BM25 (Simple) και TF-IDF (Kstem 2) σε σύνολο 100.000 άρθρων, τα αποτελέσματα των οποίων φαίνονται στο πίνακα 4.32

Όπως μπορούμε να δούμε το BM25 υπερέχει σε όλους του τομείς (καλύτερο MRR, μικρότερος χρόνος στο indexing και στο querying), κάτι το οποίο στη περίπτωση μας, δικαιολογεί τη ύπαρξη του ως προκαθορισμένη επιλογή από το Solr.

Μπορεί η εφαρμογή stemming να παρήγαγε τα καλύτερα αποτελέσματα για το TF-IDF, για το BM25 όμως, η αποφυγή χρήσης σύνθετων και παρεμβατικών φίλτρων φαίνεται ως η συνιστώμενη επιλογή.

Πίνακας 4.30: Συνολικά Ταξινομημένα Αποτελέσματα MRR Solr BM25 για 10.000 άρθρα.

Μέθοδος	MRR	Μέθοδος	MRR
Simple	0.4234	Word2Vec 4000	0.4025
SynonymFilterNTLK	0.4217	Word2Vec 4000 KStem	0.4025
SynonymFilterW2V	0.4214	Word2Vec 3000	0.3997
SynonymFilterGlove	0.4214	Word2Vec 3000 KStem	0.3997
KStem	0.4199	Word2Vec 2000 KStem	0.3945
SynonymFilterNTLK KStem	0.4194	Word2Vec 2000	0.3912
SynonymFilterGlove KStem	0.4193	Word2Vec KStem	0.3708
KStem 2	0.4191	Word2Vec	0.3704
HunspellStem	0.4186	Glove 5000 KStem	0.3678
SynonymFilterW2V KStem	0.4183	Glove 5000	0.3675
SnowballPorter	0 4163	Glove 4000	0.3329
PorterStem	0.4163	Glove 4000 KStem	0.3325
HunspellStem 2	0.4154	Glove 3000	0.3192
NLTK Lemm	0.4151	Glove 3000 KStem	0.3190
PorterStem 2	0.4138	Glove 2000	0.2995
SnowballPorter 2	0.4137	Glove 2000 KStem	0.2993
NLTK Lemm KStem	0.4132	Glove	0.2462
Word2Vec 5000 KStem	0.4071	Glove KStem	0.2461
Word2Vec 5000	0.4068		

4.2 BackEnd

Σε αυτό το κομμάτι θα παρουσιάσουμε τις βασικές κλάσεις που έχουμε υλοποιήσει για το backend κομμάτι του συστήματος. Για το κομμάτι των πειραμάτων, χρησιμοποιήσαμε τη python, λόγω της πληθώρας βιβλιοθηκών που διαθέτει, η χρήση των οποίων θα μας διευκόλυνε στην υλοποίηση των πειραμάτων.

Υλοποιήσαμε ένα flask restful api το οποίο ουσιαστικά βρισκόταν ενδιάμεσα στο FrontEnd κομμάτι της εφαρμογής και στο Solr το οποίο είχαμε στήσει.

Το python api μας αποτελείται από 5 αρχεία:

Πρώτο είναι το app.py στο οποίο έχουμε την αρχικοποίηση του api και μια συνάρτηση post μέσω της οποίας δεχόμαστε το τίτλο και την περίληψη που στέλνει ο χρήστης μέσω του FrontEnd, τα ενώνουμε σε ένα ενιαίο string, πραγματοποιούμε αναζήτηση και επιστρέφουμε τα αποτελέσματα.

Δεύτερο αρχείο είναι το indexing.py όπου η αρχικοποίηση της κλάσης Indexing που περιέχει γίνεται στο app.py. Το συγκεκριμένο αρχείο περιλαμβάνει τη συνάρ-

Πίνακας 4.31: Συνολικά Ταξινομημένα Αποτελέσματα MRR Solr TF-IDF για 10.000 άρθρα.

Μέθοδος	MRR	Μέθοδος	MRR
KStem 2	0.4191	Word2Vec 4000	0.3973
Simple	0.4177	Word2Vec 4000 KStem	0.3972
KStem	0.4175	Word2Vec 3000	0.3954
SynonymFilterNTLK KStem	0.4171	Word2Vec 3000 KStem	0.3951
SynonymFilterW2V KStem	0.4171	Word2Vec 2000 KStem	0.3916
SynonymFilterGlove KStem	0.4171	Word2Vec 2000	0.3912
HunspellStem 2	0.4167	Word2Vec KStem	0.3779
HunspellStem	0.4161	Word2Vec	0.3777
SynonymFilterNTLK	0.4159	Glove 5000	0.3719
SynonymFilterW2V	0.4158	Glove 5000 KStem	0.3718
SynonymFilterGlove	0.4158	Glove 4000	0.3410
SnowballPorter	0 4152	Glove 4000 KStem	0.3409
SnowballPorter 2	0.4152	Glove 3000 KStem	0.3298
PorterStem	0.4146	Glove 3000	0.3294
PorterStem 2	0.4144	Glove 2000 KStem	0.3127
NLTK Lemm	0.4102	Glove 2000	0.3126
NLTK Lemm KStem	0.4078	Glove KStem	0.2458
Word2Vec 5000	0.4010	Glove	0.2455
Word2Vec 5000 KStem	0.4010		

Πίνακας 4.32: Αποτελέσματα MRR Solr BM25 και TF-IDF για 100.000 άρθρα.

Είδος Αναπαράστασης	MRR	Index	Query
BM25	0.2548	00:02:37	01:23:56
TF-IDF	0.2464	00:02:45	01:31:57

τηση search_Solr, η οποία πραγματοποιεί αναζήτηση στη συλλογή της Solr εγκατάστασης μας και αυτή χρησιμοποιεί και η post του app.py για αναζήτηση. Τη συνάρτηση index, μέσω της οποίας κάνουμε απλή ευρετηριοποίηση της συλλογής στο Solr. Τη create_lemm_file, όπου δημιουργούμε το αρχείο με τις λημματοποιημένες λέξεις της συλλογής. Τη statistics, για να καταγράφουμε συγκεκριμένα χαρακτηριστικά όπως τον αριθμό εμφάνισης περιοδικών ή χρονολογιών σε ορισμένο αριθμό άρθρων. Τις word2vec και glove, για εκτέλεση των word2vec και glove αντίστοιχα. Τελος, τις συναρτήσεις Solr_testing_repeat για ορισμό των πεδίων managed schema της Solr όπου θα γίνουν οι δοκιμές και επαναληπτική κλήση

της my_utlimate_test με την οποία κάνουμε indexing και querying για το εκάστοτε πεδίο του schema, καταγραφή αποτελεσμάτων MRR και χρόνων εκτέλεσης.

Επόμενο αρχείο είναι το tokenizer.py το οποίο περιλαμβάνει συναρτήσεις για αφαίρεση ειδικών συμβόλων από φράσεις, για δημιουργία αρχείου με προστατευόμενες λέξεις και για λημματοποίηση ενός string.

Το αρχείο Solr_sorting_mrr, περιέχει τη συνάρτηση get_final_rank με την οποία κρατάμε από τη λίστα των ονομάτων των περιοδικών τα 10 πρώτα σε φθίνουσα κατάταξη και τη get_rr η οποία υπολογίζει το reciprocal rank των αποτελεσμάτων μιας αναζήτησης.

Στο αρχείο myprint.py, έχουμε συναρτήσεις όπως την print_results για καταγραφή των αποτελεσμάτων μιας αναζήτησης στο αρχείο, τη print_time_MRR για καταγραφή των αποτελεσμάτων MRR και χρόνου για το επαναληπτικό κύκλο αναζητήσεων και τη print_MRR_time_comparison για την καταχώρηση αποτελεσμάτων μια δοκιμής στο αρχείο σύγκρισης όλων των δοκιμών.

Τέλος το αρχείο word_embeddings.py περιέχει όλο το κώδικα εκτέλεσης word2vec και glove. Δηλαδή μια συνάρτηση read_prepare όπου διαβάζουμε το αρχείο με τα άρθρα και δημιουργούμε το λεξιλόγιο των word2vec και glove, τη create_clusters για δημιουργία των clusters και των ανάλογων αρχείων και τη create_file για το αρχείο με τις λημματοποιημένες λέξεις των λεξιλογίων word2vec και glove.

Στη τελική εγκατάσταση του συστήματος το flask api δεν χρειάζεται πια και η υπάρχει μόνο η σύνδεση του FrontEnd με το Solr.

4.3 FrontEnd

Όσον αφορά το FrontEnd η υλοποίηση έγινε σε Angular. Τα βασικά components ήταν το app.component το οποίο είναι η βάση και επίσης έχουμε το header.component που αφορά το τίτλο, το input_field.component το οποίο αφορά τη φόρμα καταχώρησης τίτλου, περίληψης και το κουμπί αναζήτησης find και το subcomponent του το results.componenent υπεύθυνο για τα αποτελέσματα αναζήτησης.

Ακόμη το αρχείο server.service.ts πραγματοποιεί την επικοινωνία με τη Solr εγκατάσταση μας, μέσω της συνάρτηση του findJournals, την οποία καλεί το input_field.component.ts τη στιγμή που ο χρήστης πατάει το find button .

Κεφάλαιο 5

Συμπεράσματα και Μελλοντικές Επεκτάσεις

5.1 Συμπεράσματα

Αφού έχουν ολοκληρώθηκαν οι κύκλοι δοκιμών, μπορούμε με ασφάλεια να καταλήξουμε στο συμπέρασμα ότι η αποτελεσματικότερη υλοποίηση προκύπτει από τη χρήση του fieldtype Simple στο managed-schema της Solr. Χρησιμοποιώντας δηλαδή το StandardTokenizer για διάσπαση κειμένου, το LowerCaseFilterFactory για μετατροπή σε πεζά και το StopFilterFactory για αφαίρεση StopWords, χωρίς κάποιο άλλα διαθέσιμο φίλτρο. Παρά τις διάφορες δοκιμές και τεχνικές που χρησιμοποιήσαμε, η απλούστερη εφαρμογή φίλτρων και ο συνδυασμός με τη προκαθορισμένη επιλογή ομοιότητας της Solr το BM25, αναδείχθηκαν ως ο επικρατέστερος συνδυασμός τόσο όσον αφορά χρόνο εκτέλεσης αλλά και όσον αφορά τα αποτελέσματα MRR.

Αυτή η επιλογή δεν μπορεί να θεωρηθεί η βέλτιστη για οποιεσδήποτε συλλογές κειμένου και για όλα τα συστήματα, παρά μόνο για τη δική μας περίπτωση, καθώς κάθε μία από τις φάσεις πειραματισμού που εκτελέσαμε (Stemming, Lemmatization και οι παραλλαγές των Word2Vec και Glove) μπορεί να επιδείξουν διαφορετική συμπεριφορά και να παράγουν διαφορετικά αποτελέσματα MRR.

5.2 Μελλοντικές επεκτάσεις

Η πρώτη χρήσιμη επέκταση που θα μπορούσε να ενσωματωθεί είναι η δυνατότητα αναζήτησης και κατάλληλων συνεδρίων εκτός περιοδικών.

Επίσης όσον αφορά τη βελτίωση της διεπαφής, θα μπορούσε να δοθεί η δυνατότητα επιλογής φίλτρων όσον αφορά την αναζήτηση, όπως επιλογή ερευνητικού πεδίου αναζήτησης και η επιλογή ανοικτού ή συνδρομητικού περιεχομένου περιοδικών. Όσον αφορά τα αποτελέσματα, η δυνατότητα ταξινόμησης με βάση τη χρόνος αναθεώρησης της εργασίας από το περιοδικό ή χρονική διαθέσιμη περίοδος μέχρι

την πραγματοποίηση ενός συνεδρίου. Ακόμη η αλφαβητική ταξινόμηση των περιοδικών, θα αποτελούσε μια επιπλέον δυνατότητα.

Σχετικά με τον πειραματισμό σε περισσότερα δεδομένα, οι επαναληπτικές δοκιμές που εκτελέστηκαν στο πειραματικό στάδιο, οι οποίες αναφέρονται αναλυτικά στο κεφάλαιο 4, θα μπορούσαν να δοκιμαστούν σε περισσότερα άρθρα από 10.000 είτε με αυθαίρετη επιλογή είτε με σταθερή αύξηση, για να μελετηθεί η αποδοτικότητα κάθε δοκιμής στο εκάστοτε σύνολο άρθρων αλλά και μεταβολή της σε σχέση με τις προηγούμενες αποτελέσματα σε μικρότερο σύνολο .

Επιπρόσθετος πειραματισμός στη χρήση Word2Vec και Glove θα ήταν αρκετά ενδιαφέρον. Δεδομένου ότι δεν έχει γηγενής υποστήριξη το Word2Vec και το Glove στο Solr, θα μπορούσε να υπάρξει περαιτέρω πειραματισμός είτε με τον αριθμό των clusters είτε όσον αφορά τη δημιουργία των μοντέλων σχετικά με τις διάφορες ρυθμίσεις. Ακόμη ίσως και μια δημιουργία επέκτασης για χρήση ενός αρχείου που θα περιέχει για κάθε λέξη τους κοντινότερους γείτονες (λέξεις) μαζί με το βάρος αντί για κατηγοριοποίηση του λεξιλογίου σε clusters ίσως να οδηγούσε σε καλύτερα αποτελέσματα.

Ακόμη χρήσιμη μπορεί να αποδειχθεί και η εξαγωγή StopWords από τη συλλογή. Αυτό που μπορεί να γίνει είναι να η κατάλληλη επεξεργασία στο λεξιλόγιο της συλλογής κάθε περίπτωσης ώστε να αφαιρεθούν ή να χαρακτηριστούν ως stopwords, λέξεις της συλλογής οι οποίες θα εμφανίζονται πάνω από ένα συγκεκριμένο μεταβλητό ποσοστό συχνότητας.

Τέλος, επιπλέον διαφορετικοί συνδυασμοί Solr Filters και Tokenizers θα μπορούσαν να δοκιμαστούν, καθώς η Solr διαθέτει έναν αρκετά μεγάλο αριθμό και παρέχεται αυτή η δυνατότητα στο χρήστη.

Βιβλιογραφία

[1] M. Ernst, "Choosing a venue: conference or journal?," 2006. https://homes.cs.washington.edu/~mernst/advice/conferences-vs-journals.html.

[2] S. DeGroote, "Subject and course guides: Selecting publication venues: Where to publish?," 2014. https://researchguides.uic.edu/journalselection/journalevaluation.

[3] L. P. Lewallen and P. B. Crane, "Choosing a publication venue.," *Journal of professional nursing : official journal of the American Association of Colleges of Nursing*, vol. 26, no. 4, 2010. http://dx.doi.org/10.1016/j.profnurs.2009.12.005.

[4] "Okapi bm25 — Wikipedia, the free encyclopedia," 2018 (accessed May 29, 2018). https://en.wikipedia.org/wiki/Okapi_BM25.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. http://www.aclweb.org/anthology/D14-1162.

[6] Apache Software Foundation, "Apache Solr," 2017 (accessed May 3, 2018). http://lucene.apache.org/solr/.

[7] Apache Solr, "Apache solr — Wikipedia, the free encyclopedia," 2018 (accessed May 3, 2018). https://en.wikipedia.org/wiki/Apache_Solr.

[8] Kelvin Tan , "Basic Solr Concepts," 2018 (accessed May 3, 2018). http://www.solrtutorial.com/basic-solr-concepts.html.

[9] Apache Software Foundation, "Understanding Analyzers, Tokenizers, and Filters," 2018 (accessed May 3, 2018). https://lucene.apache.org/solr/guide/7_3/understanding-analyzers-tokenizers-and-filters.html.

[10] "Angular (application platform) — Wikipedia, the free encyclopedia," 2018 (accessed May 29, 2018). `https://en.wikipedia.org/wiki/Angular_(application_platform)`.

[11] Z. Yang and B. D. Davison, "Venue recommendation: Submitting your paper with style," *2012 11th International Conference on Machine Learning and Applications*, 2012. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6406648&isnumber=6406569`.

[12] "Mean reciprocal rank — Wikipedia, the free encyclopedia," 2018 (accessed May 5, 2018). `https://en.wikipedia.org/wiki/Mean_reciprocal_rank`.