ΜΗΧΑΝΗ ΑΝΑΖΗΤΗΣΗΣ ΕΠΙΣΤΗΜΟΝΙΚΩΝ ΑΡΘΡΩΝ

Ομάδα: ΓΕΩΡΓΙΟΣ ΜΠΑΛΑΝΟΣ 5054
ΕΥΑΓΓΕΛΟΣ ΧΑΣΑΝΗΣ 5058

Github-link: https://github.com/Georgios-Mpalanos/Search_Engine

Εισαγωγή

Το σύστημα είναι μια μηχανή αναζήτησης σε μια συλλογή με επιστημονικά άρθρα. Η αναζήτηση μπορεί να γίνει με τους εξής τρόπους:

- με χρήση key word
- με χρήση field(Title, Author, Abstract, Full Text, Year)
- με phrase query

Οι χρήστες απλά πληκτρολογούν το ερώτημά τους στη γραμμή αναζήτησης και, μέσω των επιλογών που παρέχονται, μπορούν να ορίσουν τον τρόπο αναζήτησης. Τα αποτελέσματα εμφανίζονται κεντρικά στην οθόνη, ενώ αριστερά παρέχονται οι δέκα πιο πρόσφατες αναζητήσεις για γρήγορη πρόσβαση και επαναχρησιμοποίηση. Επιπλέον, υπάρχουν επιλογές πλοήγησης πάνω από τα αποτελέσματα για την προβολή των προηγούμενων και επόμενων δέκα αποτελεσμάτων, καθώς και για την εκκαθάριση της τρέχουσας αναζήτησης. Τέλος, παρέχεται η δυνατότητα ταξινόμησης των αποτελεσμάτων βάσει της χρονολογίας, προσφέροντας έναν ακόμη τρόπο οργάνωσης των ευρετηρίων.

Συλλογή εγγράφων (corpus)

Θα χρησιμοποιείστε επιστημονικά άρθρα την παρακάτω συλλογή από το kaggle:
https://www.kaggle.com/datasets/rowhitswami/nips-papers-1987-2019- updated/data?select=papers.csv

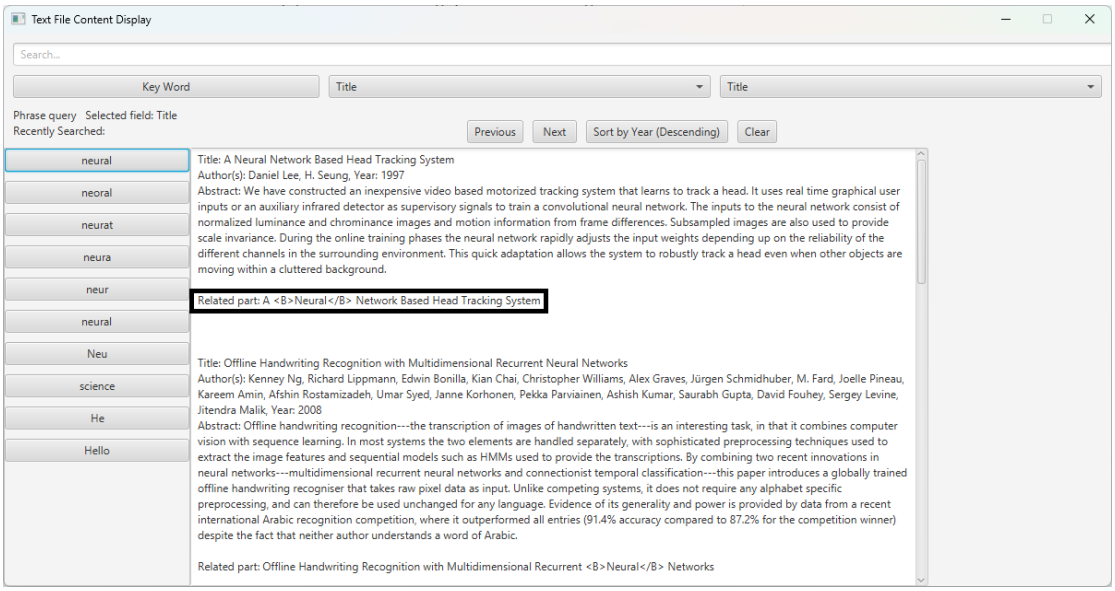
Για την συλλογή των εγγράφων θα χρησιμοποιηθεί Python script το οποίο θα επιλέξει τα πρώτα 200 έγγραφα βάσει του source id. Αντικαθίστανται οι στήλες first name και last name με μια στήλη full name. Αυτό γίνεται ώστε με χρήση του group by να συγκεντρωθούν όλοι οι authors του άρθρου σε μια γραμμή. Με αυτόν τον τρόπο, αποφεύγεται να αποθηκευτεί πολλαπλές φορές το άρθρο(όλες οι στήλες του άρθρου θα ήταν ίδιες αλλά θα αποθηκευόταν με διαφορετικό author). Χρησιμοποιήθηκαν όλα τα πεδία από το papers.csv και όλα τα πεδία εκτός του institution από το authors.csv. Για το τελικό αρχείο που δημιουργήθηκε για την ευρετηριοποίηση(merged_data.csv) χρησιμοποιήθηκαν τα cleaned_papers.csv και cleaned_authors.csv, τα οποία προέκυψαν από τα merge_papers_authors.py, new_authors.py και new_papers.py python script(αντίστοιχα).

Επίσης χρησιμοποιήθηκε το word.txt, για να ελεγχθεί αν υπάρχει κάποιο τυπογραφικό λάθος στην λέξη και αν υπάρχει, την αντικαθιστά με την προτεινόμενη(αυτή με το μεγαλύτερο σκορ). Για τον υπολογισμό του σκορ χρησιμοποιήθηκε ο SpellChecker της Lucene.

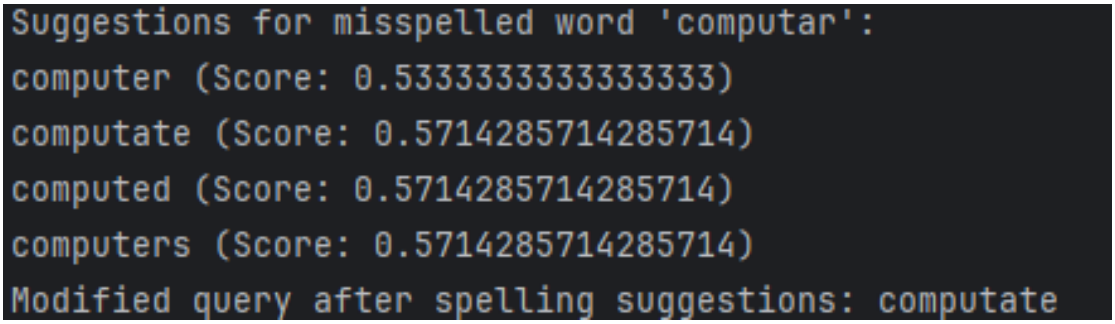
Σημείωση*: Το αρχείο word.txt παρατίθεται στο Github.*

Ανάλυση κειμένου και κατασκευή ευρετηρίου

Στο πλαίσιο της ανάλυσης κειμένου και της κατασκευής του ευρετηρίου, πραγματοποιήθηκε αρχικά η καθαρισμός των δεδομένων χρησιμοποιώντας ένα script σε Python. Αυτό εξασφάλισε την αφαίρεση των γραμμών με null τιμές στο papers.csv, καθώς και τη διαγραφή των διπλότυπων στα δύο CSV αρχεία. Χρησιμοποιήθηκαν όλα τα πεδία του εγγράφου, περιλαμβανομένου του τίτλου, των συγγραφέων, της περίληψης (abstract) και του κειμένου του άρθρου. Μονάδα εγγράφου είναι το Document που παρέχεται από τη βιβλιοθήκη Lucene. Για την κατασκευή του ευρετηρίου, χρησιμοποιήθηκε ένα term-based ευρετήριο, προκειμένου να εξυπηρετηθούν τα διαφορετικά κριτήρια αναζήτησης του χρήστη. Τα πεδία που χρησιμοποιήθηκαν για την ευρετηριοποίηση είναι τα source_id, full_name, title, year, abstract, text τα οποία προστέθηκαν κατά την δημιουργία των αντικειμένων Document(με την εντολή document_name.add(field_name)). Απαλείφθηκαν stop words(η αφαίρεση αυτών των λέξεων μπορεί να βελτιώσει την ακρίβεια των αποτελεσμάτων αναζήτησης και να μειώσει το μέγεθος του ευρετηρίου) κατά την διάρκεια της ευρετηριοποίησης της συλλογής. Για τον σκοπό αυτό χρησιμοποιήθηκε ο *Standard Analyzer* της βιβλιοθήκης Lucene. Επίσης γίνεται διόρθωση τυπογραφικών λαθών στο ερώτημα που θέτει ο χρήστης, βελτιώνοντας έτσι την ακρίβεια των αποτελεσμάτων και την εμπειρία των χρηστών κατά τη χρήση της μηχανής αναζήτησης. Χρησιμοποιήθηκε το words.txt αρχείο(από το GitHub) για την δημιουργία ενός ευρετηρίου που "διορθώνει" τα τυπογραφικά λάθη του χρήστη με βάσει τις λέξεις που υπάρχουν μέσα σε αυτό κατά την διάρκεια της αναζήτησης(αντικαθιστάται η τυπογραφικά λάθος λέξη του query που εισάγει ο χρήστης με την προτεινόμενη από το ευρετήριο. Σε αυτό συνεισφέρει και ο QueryScorer σε συνδυασμό με τον Highlighter, και οι δύο μέρος της Lucene). Τα σημεία που επιστρέφονται από τον Highlighter εμφανίζονται κάθε φορά στο τέλος του αποτελέσματος και ανάμεσα σε ...., όπως φαίνεται παρακάτω:



Αυτό το ευρετήριο δίνεται ως παράμετρος στο αντικείμενο SpellChecker που αξιοποιείται για την διόρθωση τυπογραφικών λαθών στην Lucene(μόνο κατά την αναζήτηση ενός field query). Για να επιτευχθεί η αποθήκευση στο δίσκο χρησιμοποιήθηκε το FSDirectory κατά την δημιουργία του ευρετηρίου(αντίστοιχα και για το ευρετήριο διόρθωσης τυπογραφικών λαθών). Παρακάτω φαίνεται ένα παράδειγμα λανθασμένης λέξης και τα score των προτεινόμενων λέξεων:



Με αυτές τις τεχνικές επεξεργασίας κειμένου, θα επιτευχθεί η αναβάθμιση της απόδοσης και της λειτουργικότητας της μηχανής αναζήτησής σας, ενισχύοντας την εμπειρία των χρηστών και την ακρίβεια των αποτελεσμάτων αναζήτησης.

Ο τρόπος που γίνεται Analyze το query, είναι ίδιος με τον τρόπο που γίνεται και το κείμενο, έτσι ώστε τα αποτελέσματα της προεπεξεργασίας να είναι συναφή.

Keyword query:

Για να επιτευχθεί η αναζήτηση με λέξεις κλειδιά, δημιουργείται και εκτελείται ένα query σε κάθε ένα από τα διαθέσιμα πεδία(Title, Author, Abstract, Full Text, Year) και επιστρέφονται τα αποτελέσματα.

Field query:

Για να επιτευχθεί η αναζήτηση με field query, δημιουργείται και εκτελείται ένα query στο συγκεκριμένο πεδίο και επιστρέφονται τα αποτελέσματα. Για αυτό τον λόγο στην μέθοδο getDocuments( ) της κλάσης "CSVIndexer" κατά την δημιουργία κάθε αντικειμένου Document προσθέτουμε και τα αντίστοιχα πεδία("author", "year" , "title" , "abstract" , "full_text").
Η προσθήκη του εκάστοτε πεδίου γίνεται ως εξής:

doc.add(author);

Τελικά όλα τα documents προστίθενται στον πίνακα "docs" ο οποίος επιστρέφεται από την συνάρτηση.

Ειδικά για τα field queries όπου δίνεται η δυνατότητα "διόρθωσης" των τυπογραφικών λαθών όπως έχει ήδη περιγραφεί, το κριτήριο υπολογισμού των score είναι κυρίως πόσο απέχει η λέξη που έδωσε ο χρήστης από την υποψήφια προτεινόμενη λέξη. Τελικά η λέξη που θα επιλεχθεί είναι αυτή με το μεγαλύτερο score.

Σημείωση:
Ο ορισμός της απόστασης έγινε ως εξής:

spellChecker.setStringDistance(new LuceneLevenshteinDistance());

Phrase query:

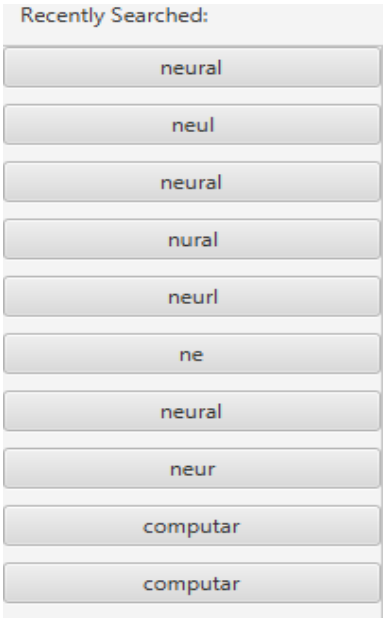
Για την δημιουργία ενός phrase query χρησιμοποιείται το παρακάτω συντακτικό:

Query query = parser.parse("\"" + phrase + "\"");

Στην συνέχεια εκτελείται το παραπάνω query με την συνάρτηση executeQuery(query, fieldName). Όπως και στην περίπτωση του field query δίνεται η δυνατότητα αναζήτησης μιας φράσης σε διαφορετικά πεδία του Document.
Τα αποτελέσματα που εμφανίζονται πρώτα έχουν είτε όλοκληρο το phrase query είτε ένα μεγάλο μέρος αυτού.

Ιστορικό Αναζήτησης

Στο ιστορικό αναζήτησης διατηρούνται οι τελευταίες 10 αναζητήσεις οι οποίες είναι και αποθηκευμένες στο αρχείο query_history.txt. Για την φόρτωση τους κατά την έναρξη της εφαρμογής γίνεται προσπέλαση αυτού του αρχείου. Κατά την εκτέλεση της εφαρμογής για κάθε καινούργια αναζήτηση αφαιρείται η πιο παλιά αναζήτηση και προστίθεται η τρέχουσα. Γίνεται πάλι προσπέλαση του αρχείου για την ανανέωση του ιστορικού. Δίνεται η δυνατότητα στον χρήστη να επιλέξει μία αναζήτηση από το ιστορικό πατώντας το αντίστοιχο κουμπί.



Παρουσίαση αποτελεσμάτων:

Ο χρήστης πληκτρολογεί στην γραμμή αναζήτησης και έπειτα επιλέγει με την χρήση των κουμπιών τον τρόπο αναζήτησης. Στο κέντρο της οθόνης εμφανίζονται τα αποτελέσματα. Αριστερά υπάρχουν οι 10 τελευταίες αναζητήσεις που έχουν πραγματοποιηθεί, οι οποίες μπορούν να χρησιμοποιηθούν από τον χρήστη για την δική του αναζήτηση. Πάνω από τα αποτελέσματα υπάρχει η επιλογή εμφάνισης των επόμενων 10 αποτελεσμάτων(Next), των προηγούμενων 10 αποτελεσμάτων(Previous), εκκαθάριση αναζήτησης(Clear) και η επιλογή ταξινόμησης των αποτελεσμάτων βάσει της χρονολογίας(Sort By Year).