Πανεπιστήμιο Ιωαννίνων

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Ανάκτηση Πληροφορίας

2η Φάση

Μέλη ομάδας:

Ελευθέριος-Χρήστος Δριτσώνας, 4668
Φωτόπουλος Στέφανος, 4829

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2023

ΠΑΡΑΔΟΣΗ: Παρασκευή 19 Μαΐου

1.Περιγραφή της εργασίας

Το link για το github είναι το παρακάτω:
https://github.com/stef-fot/Anaktisi-Pliriforias.git

Στην συγκεκριμένη αναφορά θα περιγράψουμε τα βήματα που σχετίζονται με την υλοποίηση της 2ης φάσης στο μάθημα της Ανάκτησης Πληροφορίας.

Εισαγωγή: Ποιος είναι ο στόχος και η λειτουργικότητα του συστήματος

Ο στόχος για την συγκεκριμένη εργασία είναι να δημιουργήσουμε ένα interface στο οποίο ο χρήστης θα μπορεί να θέτει ερωτήματα τα οποία θα σχετίζονται με τραγούδια που είναι αποθηκευμένα στην Βάση Δεδομένων. Αυτό επιτυγχάνεται μέσω μιας γραφικής διεπαφής (GUI) για παράδειγμα όπως το search bar της μηχανής αναζήτησης της Google.

Συλλογή Εγγράφω(corpus):
Σε αυτό το βήμα συλλέξαμε τα έγγραφα που θα χρησιμοποιήσουμε από τον παρακάτω σύνδεσμο: https://www.kaggle.com/datasets/notshrirang/spotify-million-song-dataset(είναι ο σύνδεσμος που υπάρχει και στις διαφάνειες του μαθήματος).Στο συγκεκριμένο λίνκ περιέχονται 643 καλλιτέχνες με συνολικά 44824 τραγούδια(έγγραφα). Στο GitHub όταν ανεβάσαμε το dataset διαμαρτυρήθηκε ότι το αρχείο ήταν μεγαλύτερο από τα 25MB με αποτέλεσμα να ανεβάσουμε το ίδιο αρχείο αλλά συμπιεσμένο(zip). Το format του συγκεκριμένου αρχείου είναι σε μορφή .csv (comma separated values).

1. Στην πρώτη στήλη του αρχείου βλέπουμε το όνομα του καλλιτέχνη ή του συγκροτήματος
2. Στη δεύτερη έχουμε το όνομα του τραγουδιού
3. Τέλος, η τρίτη στήλη έχει όλα τα lyrics(στίχους) των τραγουδιών.

Έχουμε σβήσει από το original αρχείο την στήλη με τα html καθώς την θεωρήσαμε περιττή

Ανάλυση κειμένου και κατασκευή ευρετηρίου:
Επεξήγηση της κλάσης LuceneIndexer:

Αρχικά, δημιουργούμε μια μεταβλητή INDEX_DIRECTORY, ο σκοπός αυτής της μεταβλητής είναι να αποθηκεύσει τη διαδρομή καταλόγου όπου θα δημιουργηθούν και θα αποθηκευτούν τα αρχεία ευρετηρίου Lucene.
Η readCsvFile είναι μια μέθοδος που διαβάζει ένα αρχείο CSV και μετατρέπει τα περιεχόμενά του σε μια λίστα αντικειμένων του εγγράφου Lucene. Αυτή η μέθοδος παίρνει ένα όνομα αρχείου ως είσοδο και επιστρέφει μια λίστα αντικειμένων εγγράφου.
Η μέθοδος ξεκινά δημιουργώντας ένα κενό ArrayList για την αποθήκευση των εγγράφων. Στη συνέχεια ανοίγει το αρχείο χρησιμοποιώντας ένα BufferedReader και ξεκινά την ανάγνωση κάθε γραμμής του αρχείου χρησιμοποιώντας τη μέθοδο readLine().
Για κάθε γραμμή, η μέθοδος ελέγχει ότι η γραμμή περιέχει τουλάχιστον τρία στοιχεία, χωρίζοντάς την χρησιμοποιώντας διαχωριστικό κόμμα και ελέγχοντας το μήκος της. Εάν η γραμμή έχει τουλάχιστον τρία στοιχεία, η μέθοδος δημιουργεί ένα νέο αντικείμενο Lucene Document και προσθέτει τρία αντικείμενα TextField σε αυτό.
Κάθε αντικείμενο TextField αντιστοιχεί σε ένα πεδίο στο ευρετήριο Lucene. Στο πεδίο καλλιτέχνης εκχωρείται το πρώτο στοιχείο της γραμμής, στο πεδίο τραγούδι εκχωρείται το δεύτερο στοιχείο και στο πεδίο κείμενο εκχωρείται το τρίτο στοιχείο.
Αφού ολοκληρωθεί η επεξεργασία όλων των γραμμών, το BufferedReader κλείνει και επιστρέφεται η

λίστα των εγγράφων. Αυτή η μέθοδος είναι χρήσιμη για τη μετατροπή δεδομένων από ένα αρχείο CSV σε μορφή που μπορεί να ευρετηριαστεί και να αναζητηθεί χρησιμοποιώντας το Lucene.

Η μέθοδος searchIndex είναι μια βασική λειτουργικότητα ενός συστήματος ανάκτησης πληροφοριών που βασίζεται στο Lucene, το οποίο ανακτά μια λίστα αντίστοιχων εγγράφων από ένα ευρετήριο με βάση το ερώτημα ενός χρήστη.
Η μέθοδος περιλαμβάνει τρεις παραμέτρους - ένα αντικείμενο IndexSearcher, μια συμβολοσειρά ερωτήματος και μια συμβολοσειρά πεδίου. Αρχικά δημιουργεί ένα αντικείμενο QueryParser με το καθορισμένο πεδίο προς αναζήτηση και έναν StandardAnalyzer, έναν δημοφιλή τύπο αναλυτή στο Lucene. Στη συνέχεια αναλύει τη συμβολοσειρά ερωτήματος σε ένα αντικείμενο Query χρησιμοποιώντας το QueryParser.
Στη συνέχεια, το IndexSearcher πραγματοποιεί αναζήτηση στο ευρετήριο για το καθορισμένο Ερώτημα και επιστρέφει τα κορυφαία 10 αποτελέσματα σε ένα αντικείμενο TopDocs. Τα αποτελέσματα αποθηκεύονται σε έναν πίνακα αντικειμένων ScoreDoc.
Εν συνεχεία, ανακτάται ένα αντικείμενο Document από το ευρετήριο για κάθε ScoreDoc χρησιμοποιώντας τη μέθοδο doc του IndexSearcher και προστίθεται σε μια λίστα με αποτελέσματα.
Τέλος, η μέθοδος επιστρέφει τη λίστα αντικειμένων Document . Αυτή η λειτουργία αναζήτησης αποτελεί θεμελιώδες μέρος πολλών συστημάτων ανάκτησης πληροφοριών και μπορεί να προσαρμοστεί και να επεκταθεί για διάφορους σκοπούς, όπως η κατασκευή μηχανών αναζήτησης, συστήματα συστάσεων και εφαρμογές εξόρυξης δεδομένων.

Όσο αφορά την Main, δημιουργείται ένα στιγμιότυπο της κλάσης StandardAnalyzer, το οποίο θα χρησιμοποιηθεί για την κανονικοποίηση του κειμένου στα έγγραφα.
Στη συνέχεια, δημιουργείται ένα αντικείμενο Path με τη διαδρομή του καταλόγου όπου θα αποθηκευτεί το ευρετήριο. Στη συνέχεια, δημιουργείται ένα στιγμιότυπο της κλάσης FSDirectory με το αντικείμενο Path, το οποίο αντιπροσωπεύει τον φυσικό κατάλογο στο δίσκο όπου θα αποθηκευτεί το ευρετήριο.
Μια παρουσία της κλάσης IndexWriterConfig δημιουργείται με τον αναλυτή. Αυτή η κλάση χρησιμοποιείται για τη διαμόρφωση του IndexWriter, το οποίο θα χρησιμοποιηθεί για την εγγραφή των εγγράφων στο ευρετήριο. Στη συνέχεια, δημιουργείται μια παρουσία της κλάσης IndexWriter με τον κατάλογο και τη διαμόρφωση.
Μια λίστα αντικειμένων εγγράφου δημιουργείται με την ανάγνωση σε ένα αρχείο CSV. Κάθε γραμμή του αρχείου χωρίζεται σε έναν πίνακα τιμών και δημιουργείται ένα νέο αντικείμενο Document για κάθε γραμμή. Τα πεδία καλλιτέχνης, τραγούδι και κείμενο κάθε αντικειμένου εγγράφου συμπληρώνονται με τις αντίστοιχες τιμές από τον πίνακα. Στη συνέχεια, η λίστα των αντικειμένων του εγγράφου προστίθεται στο ευρετήριο χρησιμοποιώντας το πρόγραμμα εγγραφής.
Μετά την ευρετηρίαση των εγγράφων, δημιουργείται μια παρουσία της κλάσης IndexSearcher με τον κατάλογο. Καθορίζεται η συμβολοσειρά ερωτήματος και το συγκεκριμένο πεδίο για αναζήτηση και καλείται η μέθοδος searchIndex με παραμέτρους το πρόγραμμα αναζήτησης, το ερώτημα και το πεδίο. Η μέθοδος επιστρέφει μια λίστα αντικειμένων εγγράφου που ταιριάζουν με το ερώτημα. Τέλος, το πεδίο ονόματος αρχείου κάθε Εγγράφου στη λίστα αποτελεσμάτων εκτυπώνεται στην κονσόλα.

Αναζήτηση: Πως θα γίνεται η αναζήτηση, τα είδη των ερωτημάτων

Αρχικά, η αναζήτηση θα γίνεται μέσω ενός search box το οποίο θα εμφανίζεται με σκοπό να εισάγει ο χρήστης την επιθυμητή ερώτηση που θέλει να κάνει στην βάση με τα τραγούδια. Τα επιτρεπτά ερωτήματα που μπορεί να εισάγει είναι τα παρακάτω:

- Το όνομα του καλλιτέχνη
- Τον τίτλο κάποιου τραγουδιού
- Κάποιον συγκεκριμένο στίχο ή στίχους από τραγούδι

Ο χρήστης σε περίπτωση που δεν τσεκάρει κάποιο από τα δυο checkbox του artist,title η αναζήτηση του θα αφορά μόνο στίχους δηλαδή το πεδίο text αυτή έχουμε ορίσει ως default αναζήτηση. Σε κάθε άλλη περίπτωση βλέπουμε είτε τα αποτελέσματα του τίτλου είτε του καλλιτέχνη αναλόγως τι θα επιλέξει. Σε αυτό το σημείο θα επισημάνουμε ότι έχουμε πραγματοποιήσει ομαδοποίηση σύμφωνα με τον καλλιτέχνη. Επίσης, έχουμε δημιουργήσει μια νέα κλάση η επεξήγηση αυτής θα γίνει παρακάτω η οποία μας βοηθά να δώσουμε recommended queries στον χρήστη.

Παρουσίαση Αποτελεσμάτων: Πως θα παρουσιάζονται τα αποτελέσματα

Όταν ο χρήστης εισάγει ένα ερώτημα και κάνει κλικ στο κουμπί αναζήτησης, καλείται η μέθοδος actionPerformed. Το κείμενο του ερωτήματος εξάγεται από το πεδίο ερωτήματος και αναλύεται με βάση τα επιλεγμένα πλαίσια ελέγχου (τίτλος ή καλλιτέχνης). Στη συνέχεια, το αναλυμένο ερώτημα χρησιμοποιείται για αναζήτηση στο ευρετήριο χρησιμοποιώντας την κλάση IndexSearcher. Τα αποτελέσματα αναζήτησης εμφανίζονται στην περιοχή αποτελεσμάτων.

Εάν ο αριθμός των αποτελεσμάτων αναζήτησης υπερβαίνει τα δέκα, προστίθεται ένα κουμπί More δίπλα στο κουμπί αναζήτησης. Κάνοντας κλικ στο κουμπί More εμφανίζονται τα επόμενα δέκα αποτελέσματα έχοντας την επιλογή εάν διατρέξουμε όλες τις σελίδες να επιστρέψουμε στην πρώτη. Το ιστορικό αναζήτησης ενημερώνεται επίσης με το ερώτημα του χρήστη το αρχείο .txt που δημιουργούμε μέσα στον φάκελο index μας βοηθά να κρατήσουμε τις προηγούμενες αναζητήσεις των χρηστών έτσι ώστε να προτείνουμε queries στους χρήστες. Στην εκφώνηση αναφέρεται ότι οι λέξεις-κλειδιά που παρουσιάζονται στην περιοχή αποτελέσματος θα πρέπει να είναι έντονα τονισμένες. Για αυτό τον λόγο υλοποιήσαμε την συνάρτηση **highlightResults().** Αρχικά , εντός της συνάρτησης αυτής δημιουργούμε έναν highlighter και έναν painter στον οποίο αναθέτουμε το κίτρινο χρώμα. Μετά αναθέτουμε στο πεδίο resultArea τον highlighter καθώς εκεί αποθηκεύονται όλα τα αποτελέσματα που βλέπουμε να εκτυπώνονται, οπότε εκεί θα βρούμε τις λέξεις κλειδιά που θέλουμε. Έπειτα αφαιρούμε την λέξη “song”, ή “artist” ή “text” από το πεδίο myquery το οποίο περιέχει το πεδίο επέλεξε ο χρήστης κατά την αναζήτησή του. Τώρα, με την χρήση της μεθόδου indexOf ψάχνουμε όλες τις εμφανίσεις της λέξης ,που εισήγαγε ο χρήστης κατά την αναζήτηση,εντός της resultArea. Όταν το position της λέξης που ψάχνουμε είναι >= του μηδέν τότε συμπεραίνουμε ότι βρήκαμε την επιθυμητή λέξη και απλά την χρωματίζουμε με το κίτρινο χρώμα χρησιμοποιώντας τον painter που δημιουργήσαμε προηγουμένως. Έπειτα ανανεώνουμε το song_index για να πάμε στην επόμενη λέξη.

Η ίδια δουλειά γίνεται και στο επόμενο loop με την διαφορά όμως ότι αναζητούμε την ίδια λέξη που πρέπει να συμπληρώσουμε με την διαφορά ότι το πρώτο γράμμα της συγκεκριμένης λέξης είναι κεφαλαίο, ενώ στην προηγούμενη περίπτωση ήταν μικρό. Τέλος, αυτή η συνάρτηση καλείται στο τέλος της dispayResults() επειδή θέλουμε το πεδίο resultArea να έχει αποθηκεύσει όλη την πληροφορία που θέλουμε και να την αξιοποιήσουμε όπως αναφέραμε. Έτσι, επιτυγχάνεται η υπογράμμιση των λέξεων.

2.Περιγραφή για το προαιρετικό ερώτημα

Όπως περιγράφεται και στην εκφώνηση, εκτός από τις παραπάνω λειτουργίες που υποστηρίζει το σύστημα που υλοποιήσαμε, προσθέσαμε να υπάρχει και η λειτουργία της σημασιολογικής αναζήτησης. Πιο συγκεκριμένα, σε αυτή την λειτουργία παρουσιάζονται στον χρήστη , στο κάτω μέρος του ίδιου παραθύρου, μαζί με τα υπόλοιπα αποτελέσματα κάποιοι τίτλοι τραγουδιών οι οποίοι περιέχουν στον τίτλο τους κάποια λέξη η οποία είναι κοντά σημασιολογικά(έχει κοντινή ερμηνεία) στο query που υπέβαλλε ο χρήστης πατώντας το κουμπί Search.

Συλλογή Εγγράφω(corpus):
Στην σημασιολογική αναζήτηση χρησιμοποιούμε διανυσματικές αναπαραστάσεις(word embeddings) για κάθε λέξη. Για αυτό τον λόγο λοιπόν, αποφασίσαμε να χρησιμοποιήσουμε ένα pretrained σύνολο δεδομένων, με βάση και την υπόδειξη που υπάρχει και στην εκφώνηση, το οποίο περιέχει πάρα πολλές λέξεις και συγκεκριμένα 6Billion**.** Το pretrained σύνολο το κατεβάσαμε από τον ακόλουθο σύνδεσμο: **https://nlp.stanford.edu/projects/glove/**
Πρακτικά, ένα pretrained σύνολο περιέχει έτοιμα και υπολογισμένα τα διανύσματα για πάρα πολλές λέξεις. Το pretrained σύνολο που χρησιμοποιήσαμε από το παραπάνω λινκ είναι αυτό που έχει μέγεθος 822MB. Στο συμπιεσμένο αρχείο υπάρχουν 4 αρχεία κειμένου (.txt) .
Το 1^{ο} αναπαριστά κάθε λέξη με ένα διάνυσμα διάστασης 50,το 2^{ο} με διάνυσμα διάστασης 100, το 3^{ο} με διάνυσμα διάστασης 200 και το 4^{ο} με διάνυσμα διάστασης 300. Αυτό με το οποίο αποφασίσαμε να δουλέψουμε εμείς είναι αυτό με τα διανύσματα που έχουν διάσταση 50(δηλαδή με το όνομα glove.6B.50d.txt).

Επεξήγηση υλοποίησης:

Όλα τα παρακάτω υλοποιήθηκαν εντός της κλάσης LuceneIndexerGUI.java .
Για να μπορέσουμε να δουλέψουμε φτιάξαμε 2 μεθόδους για αυτό το ερώτημα.
Η πρώτη μέθοδος είναι η openFile(). Σε αυτή την μέθοδο ανοίγουμε το txt αρχείο με το οποίο αναφέραμε παραπάνω ότι θα δουλέψουμε και αποθηκεύουμε κάθε γραμμή του σε ένα λεξικό της Java με το όνομα **wordMap** στο οποίο κλειδιά θα είναι το όνομα της λέξης και για τιμή το διάνυσμα που την περιγράφει. Αυτή είναι πρακτικά και η αρμοδιότητα του while, το οποίο διατρέχει το αρχείο glove.6B.50d.txt γραμμή-γραμμή. Προφανώς, για να μπορέσουμε να πάρουμε τις λέξεις που είναι σημασιολογικά κοντά στο query που εισάγει ο χρήστης θα πρέπει να αποθηκεύσουμε κάπου ξεχωριστά το διάνυσμα που αφορά την λέξη του query έτσι ώστε να μπορούμε να το χειριζόμαστε πιο εύκολα. Αυτό αποθηκεύεται στην ArrayList **queryVector** μόλις συναντήσουμε την λέξη του query στο έγγραφο glove.6B.50d.txt.
Στο σημείο αυτό, καλό είναι να εισάγουμε την έννοια της ομοιότητας. Επειδή, οι λέξεις στο συγκεκριμένο πρόβλημα αναπαρίστανται με την μορφή διανυσμάτων(αριθμών δηλαδή) θέλουμε να βρούμε έναν εύκολο τρόπο με τον οποίο θα μπορούμε να αποφασίζουμε για το πότε 2 λέξεις είναι σημασιολογικά κοντά η μια στην άλλη. Αυτή η λειτουργία επιτυγχάνεται μέσω της έννοιας

Cosine Similarity. Όπως έχουμε δει και στην θεωρία αυτό θεμελιώνεται μαθηματικά με τον παρακάτω τύπο:
sim(a,b)= a * b / || a|| * ||b|| , όπου a,b διανύσματα της ίδιας διάστασης. Αυτός ο τύπος παίρνει τιμές στο δίαστημα [0,1].Όσο πιο κοντά στο 1 είναι τότε τόσο πιο κοντά σημασιολογικά είναι η λέξη που αναπαρίσταται από το διάνυσμα a με την λέξη που αναπαρίσταται από το διάνυσμα b, ενώ όσο είναι κοντά στο 0 δεν έχουν και τόσο μεγάλη σχέση μεταξύ τους.

Για το παραπάνω λόγο φτιάξαμε την συνάρτηση **cosineSimilarity()** η οποία παίρνει σαν ορίσματα 2 διανύσματα τύπου ArrayList και εκτελεί πάνω σε αυτά τον παραπάνω μαθηματικό τύπο. Σημειώνουμε ότι όπου υπήρχε αρνητική τιμή σε κάποιο διάνυσμα την μετατρέπαμε σε θετική. Στον αριθμητή του τύπου το * συμβολίζει το εσωτερικό γινόμενο μεταξύ των 2 διανυσμάτων. Σαν αποτέλεσμα επιστρέφεται η τιμή που προκύπτει από την παραπάνω πράξη (μέσω της μεταβλητής **similarity**).
Έτσι, αφού εξηγήσαμε τι κάνει και η cosineSimilarity() επιστρέφουμε στην openFile(). Αφότου έχουμε πλέον όλες τις λέξεις και τα διανύσματα τους στο wordMap και το διάνυσμα που αντιστοιχεί στο query καλούμε την συνάρτηση cosineSimilarity που υλοποιήσαμε για κάθε λέξη του wordMap και της λέξης του query. Τα αποτελέσματα για κάθε λέξη τα αποθηκεύουμε στο λεξικό cosineSimResults το οποίο έχει για κλειδί κάθε λέξη και για τιμή το αποτέλεσμα που επιστρέφει η κλήση της cosineSimilarity(). Προφανώς, από αυτά θέλουμε να πάρουμε τα 10(τυχαίος αριθμός) μεγαλύτερα , δηλαδή με απλά λόγια τις 10 λέξεις που είναι οι πιο κοντινές σημασιολογικά με την λέξη του query. Για να γίνει αυτό ταξινομούμε το λεξικό με την χρήση της **sort()** μεθόδου και αντιστρέφουμε την σειρά(καλούμε την **reverse())** με την οποία εμφανίζονται επειδή μας βολεύει καλύτερα να εμφανίζονται σε φθίνουσα σειρά. Τέλος, κρατάμε το αποτέλεσμα με τις 10 πιο κοντινές λέξεις στην λέξη του query στην **ArrayList semanticwords** η οποία έχει προστεθεί σαν πεδίο.Η openFile() είναι void οπότε δεν επιστρέφει κάτι.

Τέλος, μετά τον σχηματισμό της semanticwords καλούμε την openFile() εντός της displayResults() . Έπειτα, χρησιμοποιώντας κάποιες συναρτήσεις της Lucene (ανοίγωντας ξανά το ευρετήριο κτλ όπως και στην κανονική άσκηση) αναζητούμε για κάθε λέξη που υπάρχει στην λίστα semanticwords έναν τίτλο τραγουδιού που να περιέχει αυτή την λέξη. Το αποτέλεσμα το τυπώνουμε κάνοντας το append στο πεδίο resultArea το οποίο διατηρεί και τα υπόλοιπα αποτελέσματα της κανονικής αναζήτησης. Πριν αρχίσουν να εκτυπώνονται στο παράθυρο τα αποτελέσματα της σημασιολογικής αναζήτησης έχουμε βάλει να εκτυπώνεται το μήνυμα *SEMANTIC SEARCH RESULTS* ειδοποιώντας με αυτό τον τρόπο τον χρήστη ότι από εκεί και κάτω θα ξεκινήσουν να εκτυπώνονται τα αποτελέσματα της σημασιολογικής αναζήτησης.

Εκτυπώνονται με το εξής μήνυμα: **Song: ….**

Σημείωση: Λόγω του ότι το αρχείο txt με τα διανύσματα είναι αρκετά μεγάλο υπάρχει το ενδεχόμενο να υπάρχει μια μικρή καθυστέρηση(3-4 δευτερόλεπτα) στην εμφάνιση των αποτελεσμάτων στο παράθυρο.