Δουμάνη Τριανταφυλλιώ-4052

Τσίγκου Μαρία-4191

Links:

Github ***CSV dataset*** ***Word2Vec pretrained model*** ***Demo***

Μηχανή Αναζήτησης πληροφορίας για τραγούδια

Εισαγωγή

Η εργασία αφορά στο σχεδιασμό και υλοποίηση ενός συστήματος αναζήτησης στίχων τραγουδιών και άλλης πληροφορίας σχετικής με μουσικούς και τραγούδια. Για την υλοποίηση θα χρησιμοποιηθεί η βιβλιοθήκη Lucene, μια βιβλιοθήκη ανοικτού κώδικα για την κατασκευή μηχανών αναζήτησης κειμένου.

Η μηχανή αναζήτησης υποστηρίζει:

- Αναζήτηση με λὲξεις κλειδία
- Αναζήτηση σε πεδία
- Σημασιολογική αναζήτηση με χρήση pretrained model(Word2Vec)
- Ομαδοποίηση αποτελεσμάτων με βάση το πεδίο “Year”
- Διόρθωση τυπογραφικών λαθών
- Εναλλακτικά ερωτήματα με βάση το ιστορικό αναζήτησης
- Εμφάνιση εως 10 αποτελεσμάτων ανα σελίδα
- Highlighting των σχετικών αποτελεσμάτων

Να σημειώσουμε πως δεν βρέθηκε pretrained Word2Vec model το οποίο έχει εκπαιδευτεί πὰνω σε τραγούδια, κι αυτο τον λόγο τα αποτελέσματα πολλές φορές δέν είναι ακριβή. Ωστόσο χρησιμοποιήθηκε το GoogleNews-vectors-negative300.bin

Η μηχανή αναζήτησης υλοποιήθηκε στο περιβάλλον eclipse 4.26.0.

Η υλοποίηση του User Interface(GUI) έγινε με χρήση Java Swing και Java AWT(Abstract Window Toolkit) ενώ η υλοποίηση του ευρετηρίου και της αναζήτησης με χρήση της βιβλιοθήκης Lucene 9.5.0.

Για την πραγματοποίηση της εφαρμογής δημιουργήθηκαν τα εξής πακέτα με τις αντίστοιχες κλάσεις τους:

- **Org:**
    - **LuceneGUI.java:** Δημιουργία του κεντρικού παραθύρου του User Interface για την αλληλεπίδραση της εφαρμογής με τον χρήστη.
- **Org.Constants:**
    - **LuceneConstants.java:** Καθορισμός σταθερών που θα χρησιμοποιηθούν για την ανάκτηση/αποθήκυεση πληροφοριών, όπως είναι τα path των ευρετηρίων, του pretrained model και της συλογής δεδομένων.
- **Org.Indexer:**
    - **Indexer.java(Interface):** Αναφορά των κοινών και βασικών συναρτήσεων που υλοποιούν οι δύο διαφορετικοί Analyzers για την δημιουργία των ευρετηρίων(createIndexer(),close()).
    - **StandardIndexerImpl.java(implements Indexer):** Υλοποίηση Interface,δημιουργία ευρετηρίου με χρήση του StandardAnalyzer() για αναζήτηση σε πεδία, καθώς και αναπαράσταση των λέξεων ως διανύσματα, για την υποστήριξη σημασιολογικής αναζήτης.
    - **KeywordIndexerImpl.java(implements Indexer):** Υλοποίηση Interface, δημιουργία ευρετηρίου με χρήση του KeywordAnalyzer() για αναζήτηση λέξεων-κλειδιών σε όλα τα πεδία.
    - **Helper.java:** Υλοποίηση βοηθητικών συναρτήσεων για επεξεργασία κειμένου και εκτύπωση των ονομάτων των πεδίων με σκοπό έλεγχου.
- **Org.Searcher:**
    - **Searcher.java(Interface):** Αναφορά των κοινών και βασικών συναρτήσεων που υλοποιούν οι δύο διαφορετικοί Analyzers για την πραγματοποίηση της αναζήτησης.
    - **KeywordSearcherImpl.java(implements Searcher):** Υλοποίηση Interface,υλοποίηση αναζήτησης λέξεων-κλειδιών, ομαδοποίηση βάση συγκεκριμένου πεδίου.
    - **StandardSearcherImpl.java(imlements Searcher)):** Υλοποίηση Interface,υλοποίηση αναζήτησης σε πεδία που επιλέγει ο χρήστης, ομαδοποίηση βάση συγκεκριμένου πεδίου.
    - **SemanticSearcherImpl.java(imlements Searcher)):** Υλοποίηση Interface,υλοποίηση σημασιολογικής αναζήτησης σε πεδία που επιλέγει ο χρήστης, ομαδοποίηση βάση συγκεκριμένου πεδίου.
    - **History.java:** Υλοποίηση βοηθητικής συνάρτησης για την διατήρηση του ιστορικού αναζήτησης του χρήστη, με σκοπό στην επόμενη αναζήτηση να του προταθεί κάποιο εναλλακτικό ερώτημα.
    - **Helper.java:** Υλοποίηση βοηθητικών συναρτήσεων για την αναζήτηση όσον αφορά την επεξεργασία κειμένου.
    - **SearchEngine.java:** Υλοποίηση κεντρικής αναζήτησης και επιστροφή της λίστας των αποτελεσμάτων. Υπεύθυνη για τον υπολογισμό των αποτελεσμάτων ανα σελίδα.

- **SearcherFactory.java:** Υπεύθυνη να επιλέξει ανάλογα με την είσοδο του χρήστη, τι είδους αναζήτηση θα πραγματοποιηθεί(αναζήτηση σε πεδίο, λέξης-κλειδιού ή σημασιολογική).

Η πρώτη φάση αποτελείται από τη συλλογή των δεδομένων η οποία πραγματοποιήθηκε με τη βοήθεια του Kaggle και περιγράφεται παρακάτω στην πρώτη ενότητα.

---

1. Συλλογή εγγράφων

Η συλλογή που θα χρησιμοποιηθεί για την εργασία αποτελείται από πολλαπλά .csv αρχεία με το καθένα να περιέχει πληροφορίες και στίχους για τραγούδια από έναν καλλιτέχνη. Ο σκοπός είναι να συγχωνευτούν όλα σε ένα αρχείο. Η διαδικασία έγινε σε γλώσσα Python με τη βοήθεια της βιβλιοθήκης Pandas και Glob.

Τα βήματα που ακολουθήθηκαν:

- Χρήση της βιβλιοθήκη glob για εντοπισμό των .csv αρχείων στον φάκελο, αποθηκεύοντας τα ονόματά τους σε μία λίστα με τη χρήση της συνάρτησης ‘glob.glob('*.csv’)’.
- Τοποθέτηση των dataframes των διαφορετικών .csv αρχείων στη λίστα ‘df_list’ με χρήση της συνάρτησης ‘pd.read_csv()’.
- Χρήση της συνάρτησης ‘pd.concat()’ για συνένωση όλων των dataframes που υπάρχουν στη λίστα ‘df_list’ σε ένα μεγάλο dataframe (‘df_merged’).
- Χρήση της συνάρτησης ‘df_merged.dropna()’ για εύρεση τυχόν τιμών που έχουν χαθεί και διαγραφή της αντίστοιχης εγγραφής από το αρχείο.

Το Dataframe είναι στην παρακάτω μορφή:

Artist, Title, Album, Date, Lyrics, Year

Ariana Grande,"thank u, next","thank u, next",2018-11-03,<Lyrics>,2018

Η κάθε καταχώρηση αποτελείται από τα εξής πεδία: καλλιτέχνης, τίτλος τραγουδιού, άλμπουμ, ημερομηνία κυκλοφορίας, στίχοι, χρονιά κυκλοφορίας.

Παρακάτω φαίνεται και ο αναλυτικός κώδικας για την παραπάνω διαδικασία:

```
import pandas as pd
import glob

#get a list of all CSV files in the directory
file_list = glob.glob('*.csv')

#create an empty list to hold the individual dataframes
df_list = []
```

```
#get to each gile in the file list
for f in file_list:

        #read the csv file and append it to the list
        df = pd.read_csv(f)
        df_list.append(df)

#merge all the dataframes into one
df_merged = pd.concat(df_list, ignore_index=True)



#remove the field you dont want
df_merged = df_merged.drop('Unnamed: 0',axis=1)

# Rename the 'song_name' column to 'title'
df_merged = df_merged.rename(columns={'Lyric': 'Lyrics'})

#drop missing values
df_merged = df_merged.dropna()

#write the final dataset back out to a file
df_merged.to_csv('final_dataset.csv',index=False)
```

---

2. Ανάλυση κειμένου και κατασκευή ευρετηρίου.

Υποστηρίζονται 2 Analyzers για να καλυφθούν όλες οι ανάγκες.

a) StandardIndexerImpl.java

Η συγκεκριμένη κλάση είναι υπεύθυνη για την υλοποίηση του ευρετηρίου χρησιμοποιώντας τον StandardAnalyzer(). Ο StandardAnalyzer χρησιμοποιήθηκε καθώς υποστηρίζει tokenization, lowercasing και απαλοιφή των stopwords. Με χρήση αυτού πραγματοποιείται και η αναζήτηση σε πεδία.

Το πρώτο στάδιο είναι η δημιουργία των Documents και των Fields με την βοήθεια αντίστοιχων κλάσεων της Lucene. Τα βήματα είναι:

- **Αρχικοποίηση του index directory και αποθήκευση στο καθορισμένο path.**
  ```
  Directory index = FSDirectory.open(Paths.get(LuceneConstants.STANDARD_INDEX_FILE_PATH));
  ```

- **Αρχικοποίηση του index configuration με χρήση του StandardAnalyzer()**
```
IndexWriterConfig config = new IndexWriterConfig(new StandardAnalyzer());
```

- **Δημιουργία και αρχικοποίηση του indexWriter.** Υπεύθυνος για την δημιουργία του ευρετηρίου.
```
this.indexWriter = new IndexWriter(index, config);
```

- **Φόρτωση του csv file χωρισμένο με κόμμα(“,”) και κατοχύρωση της πρώτης γραμμής ως επικεφαλίδα (header).** Δηλαδή τα ονόματα των πεδίων. (π.χ. Artist,Title..κλπ)
```
CSVFormat format = CSVFormat.DEFAULT.withFirstRecordAsHeader();
this.parser = CSVParser.parse(new
        File(LuceneConstants.CSV_PATH_AND_FILE_NAME),
        StandardCharsets.UTF_8, format);
```

- **Δημιουργία Document**

Γενικά, το κάθε τραγούδι αντιστοιχεί σε μία μονάδα εγγράφου, οπότε για κάθε τραγούδι(δηλαδή για κάθε εγγραφή στο csv αρχείο) δημιουργείται ένα document ως εξής: *Document doc = new Document();.*

Στη συνέχεια σε κάθε document προστίθενται τα πεδία (fields), τα οποία είναι τα εξής: “Artist”,”Title”,”Album”,”Date”,”Lyrics”,”Year” με τη πληροφορία που αυτά έχουν αποθηκευμένη για κάθε εγγραφή τραγουδιού.

```
    for (CSVRecord record : parser) {

            Document doc = new Document();

            for (String header : headerMap.keySet()) {

                String text = record.get(header);

            }}
```

Αν το πεδίο που πρόκειται να αποθηκευτεί είναι το “Year”, (που είναι καθορισμένο ως static στο LuceneConstants.java), τότε προστίθεται στο document ως SortedDocValuesField εφόσον είναι το πεδίο βάση του οποίου θα γίνει η ταξινόμηση των αποτελεσμάτων, εφόσον ο χρήστης το επιθυμεί. Ακόμα ,αποθηκεύεται και ως StringField καθώς αποτελείται από ένα μόνο String, άρα δεν χρειάζεται κάποια διάσπαση.

Διαφορετικά, το πεδίο αποθηκεύεται ως TextField (επιτυγχάνεται η διάσπαση σε όρους) με παράμετρο ‘Field.Store.YES’ έτσι ώστε να αποθηκευτεί η τιμή του στο ευρετήριο.

Τα υπόλοιπα πεδία για τον ίδιο λόγο αποθηκεύονται ως TextField.

```
if (header.equals(LuceneConstants.GROUP)) {
```

```
        doc.add(new SortedDocValuesField (header, new BytesRef(text) ));

        doc.add(new StringField(header, text));

}else {

         doc.add(new TextField(header, text, Field.Store.YES));}
```

Τέλος, προστίθεται το document στο ευρετήριο: *this.indexWriter.addDocument(doc);*

Τροποποίηση δημιουργίας ευρετηρίου για την υποστήριξη σημασιολογικής αναζήτησης, με χρήση του μοντέλου Word2Vec.

Για την υποστήριξη της σημασιολογικής αναζήτητης είναι απαραίτηση η τροποποίηση του τρόπου δημιουργίας του ευρετηρίου, με σκοπό την αναπαράσταση των λέξεων ως διανύσματα, και την αποθήκευση τους σε επιπλέον fields του ευρετηρίου. Τα βήματα είναι τα εξής:

- **Φορτώση Word2Vec:** Φορτωση ένός pretrained μοντέλου Word2Vec που αναπαριστά τις λέξεις σε έναν διανυσματικό χώρο.

```
    Word2Vec word2Vec = WordVectorSerializer.readWord2VecModel(
                                    newFile(LuceneConstants.MODEL_PATH_AND_FILE_NAME));
```

- **Υπολογισμός μέσου διανύασματος:** Τα διανύσματα αναπαράστασης των λέξεων συγκεντρώνονται και υπολογίζεται το μέσο διάνυσμα αναπαράστασης για το κάθε κείμενο. Αυτό το μέσο διάνυσμα αναπαράστασης αποθηκεύεται ως ένα πεδίο στο έγγραφο.

```
double[] vector = new double[word2Vec.getLayerSize()];
for (String word : words) {
   double[] wordVector = word2Vec.getWordVector(word);
   if (wordVector != null) {
      for (int i = 0; i < vector.length; i++) {
         vector[i] += wordVector[i];
      } }}
// Κανονικοποίηση του διανύσματος για τον υπολογισμό μέσου διανύσματος
for (int i = 0; i < vector.length; i++) {
   vector[i] /= words.length;}
```

- **Προσθήκη εγγράφων:** Κατά την προσθήκη εγγράφων, χρησιμοποιείται το pretrained μοντέλο Word2Vec για να υπολογιστεί η αναπαράσταση του κάθε κειμένου. Οι λέξεις του κειμένου αναλύονται και αντιστοιχίζονται σε διανύσματα αναπαράστασης με βάση το μοντέλο.

```
// Δημιουργία και προσθήκη του πεδίου αναπαράστασης στο έγγραφο
```

```
ByteBuffer byteBuffer = ByteBuffer.allocate(Double.BYTES * vector.length);
for (double v : vector) {
   byteBuffer.putDouble(v);}
Field vectorField = new StoredField(header + "_vector", byteBuffer.array());
doc.add(vectorField);
```

b) KeywordIndexerImpl.java

Η συγκεκριμένη κλάση είναι ,ομοίως, υπεύθυνη για την υλοποίηση του ευρετηρίου χρησιμοποιώντας τον KeywordAnalyzer(). Ο KeywordAnalyzer() δεν υποστηρίζει tokenization, lowercasing(κ.α.) και χρησιμοποιήθηκε καθώς θέλουμε η αναζήτηση να γίνεται με λέξεις κλειδιά(keywords).

Η υλοποίηση του κώδικα είναι πανομοιότυπη με αυτή του StandardIndexerImpl.java με τη διαφορά ότι γίνεται προεπεξεργασία του περιεχομένου του κάθε πεδίου πριν προστεθεί στο Document. Λόγω του ότι ο συγκεκριμένος analyzer δεν χρησιμοποιεί επεξεργασία των δεδομένων που δέχεται, έχει δημιουργηθεί η συνάρτηση preprocessText(), στο Helper.java, η οποία κάνει απαλοιφή των κενών (trim()), αντικαθιστά τα σημεία στίξης και ότι δεν είναι γράμμα ή αριθμός σε κενό(replaceAll("[^a-zA-Z0-9]"," ")) και μετατρέπει τα κεφαλαία γράμματα σε πεζά.

---

3. Αναζήτηση

Το σύστημα αναζήτησης που υλοποιήθηκε υποστηρίζει 3 τρόπους αναζήτησης:

1. Αναζήτηση σε διάφορα πεδία(χρησιμοποιώντας τον StandardAnalyzer())
2. Αναζήτηση λέξης κλειδιού (χρησιμοποιώντας τον KeywordAnalyzer())
3. Σημασιολογική αναζήτηση(χρησιμοποιώντας τον StandardAnalyzer() και υπάρχουσες διανυσματικές αναπαραστάσεις από ένα pretrained σύνολο)

SearchEngine.java

Αναζήτηση

Στη συγκεκριμένη κλάση υλοποιείται η διαδικασία της αναζήτησης, μέσω της μεθόδου search() που δέχεται ως όρισμα το κείμενο που επιθυμεί να ψάξει ο χρήστης(currentQuery), τη μέθοδο αναζήτησης που θα χρησιμοποιηθεί(currentField) και αν επιθυμεί να παρουσιαστούν τα αποτελέσματα ομαδοποιημένα(ορίζοντας μια Boolean μεταβλητή «Boolean isGrouped»).

Ο χρήστης πατώντας το κουμπί “Search” καλείται η παραπάνω συνάρτηση έχοντας καθοριστεί οι τιμές των ορισμάτων της και το πρώτο βήμα είναι να προστεθεί η ερώτηση που έκανε ο χρήστης στη λίστα που διατηρεί το ιστορικό αναζητήσεων.
history.getHistory(currentQuery);

Αμέσως μετά, καλείται η συνάρτηση selectSearcher() της κλάσης SearcherFactory(), η οποία δέχεται σαν όρισμα το τύπο αναζήτησης που επέλεξε ο χρήστης να πραγματοποιήσει και βάση αυτού επιλέγεται η κατάλληλη κλάση για να εκτελέσει τη διαδικασία. Στο τέλος επιστρέφεται η λίστα με τα αποτελέσματα στο LuceneGUI από όπου ξεκίνησε και η διαδικασία της αναζήτησης.

Page Handling

Η κλάση αυτή είναι επίσης υπεύθυνη για το ποιά αποτελέσματα πρέπει να εμφανιστούν σε κάθε σελίδα. Η λογική υλοποιείται με τη χρήση μιας μεταβλητής currentPage που κρατάει τον αριθμό της τρέχουσας σελίδας με τις εξής μεθόδους:

hasNextPage(), hasPreviousPage(): Οι μέθοδοι ελέγχουν απλά αν ο αριθμός της τρέχουσας σελίδας είναι μικρότερος από τον μέγιστο αριθμό σελίδων ή μεγαλύτερος από 1 αντίστοιχα.

GetNextPage(), getPreviousPage(): Αυξάνει ή μειώνει αντίστοιχα την τιμή του currentPage και καλεί τη μέθοδο search() για να αναζητήσει τα αποτελέσματα της επόμενης σελίδας. Εάν υπάρχει επόμενη σελίδα, επιστρέφει τα αποτελέσματα, αλλιώς επιστρέφει μια άδεια λίστα.

a) StandardSearcherImpl.java

Ο πρώτος τρόπος αναζήτησης είναι χρησιμοποιώντας τον StandardAnalyzer(),όπου ο χρήστης έχει τη δυνατότητα να επιλέξει σε ποιο πεδίο επιθυμεί να πραγματοποιήσει την αναζήτηση του. Αν επιλεγεί , από τη SearcherFactory() κλάση να πραγματοποιηθεί η συγκεκριμένη αναζήτηση, τότε μέσω αυτής καλείται η συνάρτηση search() (αναφέρθηκαν παραπάνω τα ορίσματα της) της StandardSearcherImpl κλάσης. Τα βήματα που ακολουθήθηκαν είναι τα εξής:

- **Δημιουργία του QueryParser**
  ```
  QueryParser queryParser = new QueryParser(currentField, new StandardAnalyzer());
  ```
  Το όρισμα “currentField” είναι το πεδίο αναζήτησης που έχει επιλέξει ο χρήστης να ψάξει.

- **Διόρθωση ορθογραφικών λαθών και δημιουργία του Query**
  ```
  Query correctedQuery = queryParser.parse(correctedQueryString);
  ```

  Η διόρθωση ορθογραφικών γίνεται με την συνάρτηση:

addTypoCorrection(Query query, String currentField). Πρώτα, το ερώτημα μετατρέπεται σε συμβολοσειρά. Στη συνέχεια, η συμβολοσειρά διαιρείται σε λέξεις. Για κάθε λέξη, προστίθεται ο χαρακτήρας ~1 στο τέλος της, για να προσδιορίσει το επίπεδο των τυπογραφικών διορθώσεων που θα γίνουν.. Τέλος, οι τροποποιημένες λέξεις ενώνονται πάλι για να δημιουργηθεί η τελική συμβολοσειρά με τις διορθώσεις τυπογραφικών λαθών.

```
    public String addTypoCorrection(Query query, String currentField) throws
ParseException {
        String queryString = query.toString(currentField);
        String[] words = queryString.split(" ");
        String[] correctedWords = new String[words.length];

        for (int i = 0; i < words.length; i++) {
           correctedWords[i] = words[i] + "~2";  }
        String correctedQueryString = String.join(" ", correctedWords);
        return correctedQueryString;}
```

- **Άνοιγμα του ευρετηρίου**

- **Χρήση του αρχικοποιημένου IndexSearcher, που είναι υπεύθυνος για την αναζήτηση και εκτέλεση αναζήτησης με επιστροφή συγκεκριμένου αριθμού αποτελεσμάτων**

```
topDocs = standardIndexSearcher.search(query, numOut);
```

- **Υπολογισμός των scores κάθε αποτελέσματος και αποθήκευση στον πίνακα ScoreDoc[]**

```
ScoreDoc[] hits = topDocs.scoreDocs;
```

Η διαδικασία που περιγράφηκε παραπάνω αποτελεί υλοποίηση αναζήτησης **χωρίς** ομαδοποίηση των αποτελεσμάτων. Στη περίπτωση όπου ο χρήστης θέλει ομαδοποιημένα τα αποτελέσματα γίνεται κλήση της συνάρτησης groupingResults() που δέχεται σαν όρισμα το ερώτημα που θέτει ο χρήστης και επιστρέφει τη λίστα των αποτελεσμάτων. *topGroups = groupingResults(query);.*
Η συγκεκριμένη ομαδοποιεί τα αποτελέσματα σύμφωνα με το πεδίο 'Year' και δίνει συνολικά το πολύ 50 αποτελέσματα για κάθε ομάδα.

```
Sort groupSort = new Sort(new SortField(LuceneConstants.GROUP,
SortField.Type.STRING));
groupingSearch.setGroupDocsLimit(50);
groupingSearch.setGroupSort(groupSort);
TopGroups<BytesRef> topGroups =
groupingSearch.search(standardIndexSearcher, query, 0, 50);
```

b) KeywordSearcherImpl.java

Ο δεύτερος τρόπος αναζήτησης είναι χρησιμοποιώντας τον KeywordAnayzer() όπου ο χρήστης αναζητά σε όλα τα πεδία λέξεις-κλειδιά(keywords). Αν επιλεγεί , από τη SearcherFactory() κλάση να πραγματοποιηθεί η συγκεκριμένη αναζήτηση, τότε μέσω αυτής καλείται η συνάρτηση search() (αναφέρθηκαν παραπάνω τα ορίσματα της) της KeywordSearcherImpl κλάσης. Τα βήματα που ακολουθήθηκαν είναι τα ίδια με τον παραπάνω Analyzer με τη μόνη διαφορά ότι χρησιμοποιείται ως QueryParser ο MultiFieldQueryParser() για να πραγματοποιηθεί αναζήτηση σε όλα τα πεδία.

```
QueryParser queryParser = new
MultiFieldQueryParser(Helper.getFieldNames(keywordIndexReader), new
KeywordAnalyzer());
```

c) SemanticSearcherImpl.java

- **Φόρτωση του μοντέλου Word2Vec:** Ανοίγει το αρχείο μοντέλου Word2Vec και το φορτώνει στο πρόγραμμα. Αυτό το μοντέλο χρησιμοποιείται για την αναπαράσταση των λέξεων ως διανύσματα.
- **Υπολογισμός του διανύσματος του ερωτήματος:** Το ερώτημα(query) μετατρέπεται σε ένα διάνυσμα χρησιμοποιώντας το μοντέλο Word2Vec. Αυτό το διάνυσμα αναπαριστά το σημασιολογικό περιεχόμενο του ερωτήματος.

  Ο υπολογισμός του διανύσματος έγινε με την χρήση της συνάρτησης computeQueryVectorh.

```
private double[] computeQueryVector(String query) {
                String[] words = query.split("\\s+");
                double[] vector = new double[word2Vec.getLayerSize()];
                for (String word : words) {
                   double[] wordVector = word2Vec.getWordVector(word);
                   if (wordVector != null) {
                      for (int i = 0; i < vector.length; i++) {
                         vector[i] += wordVector[i];
                      }}}
                for (int i = 0; i < vector.length; i++) {
                   vector[i] /= words.length;}
                return vector;}
```

Η συνάρτηση αρχικά, διαιρεί το ερώτημα σε λέξεις και δημιουργεί ένα διάνυσμα με μηδενικές τιμές. Στη συνέχεια, ανακτά τα διανύσματα των λέξεων από το pretrained μοντέλο Word2Vec και τα προσθέτει στο συνολικό

διάνυσμα του ερωτήματος. Τέλος, κανονικοποιεί το διάνυσμα διαιρώντας το με τον αριθμό των λέξεων. Η συνάρτηση επιστρέφει το διάνυσμα του ερωτήματος ως αποτέλεσμα.

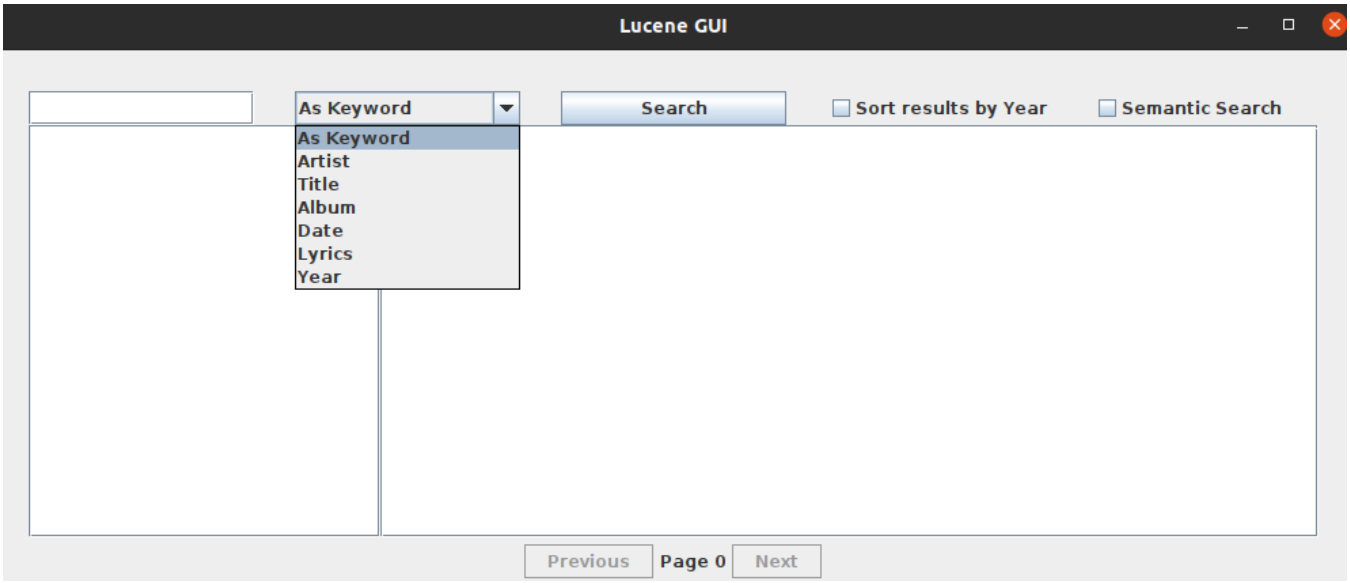
- **Επαναταξινομιση εγγράφων βάση της ομοιότητας συνημιτόνου**: Τα έγγραφα ταξινομούνται εκ νέου βάσει της ομοιότητας συνημιτόνου (*cosine similarity*) μεταξύ του διανύσματος του ερωτήματος και του διανύσματος του κάθε έγγραφου. Αυτός ο υπολογισμός γίνεται με χρήση της συνάρτησης cosineSimilarity.

  Η συνάρτηση, υπολογίζει το εσωτερικό γινόμενο (dot product) των δύο διανυσμάτων και τις τετραγωνικές νόρμες (euclidean norm) κάθε διανύσματος. Στη συνέχεια, τα διαιρεί για να υπολογίσει την ομοιότητα συνημιτόνου, η οποία κυμαίνεται από -1 έως 1 και όσο πιο κοντά στο 1 είναι η τιμή της, τόσο πιο παρόμοια είναι τα διανύσματα.

```
return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
```

---

4 Παρουσίαση Αποτελεσμάτων

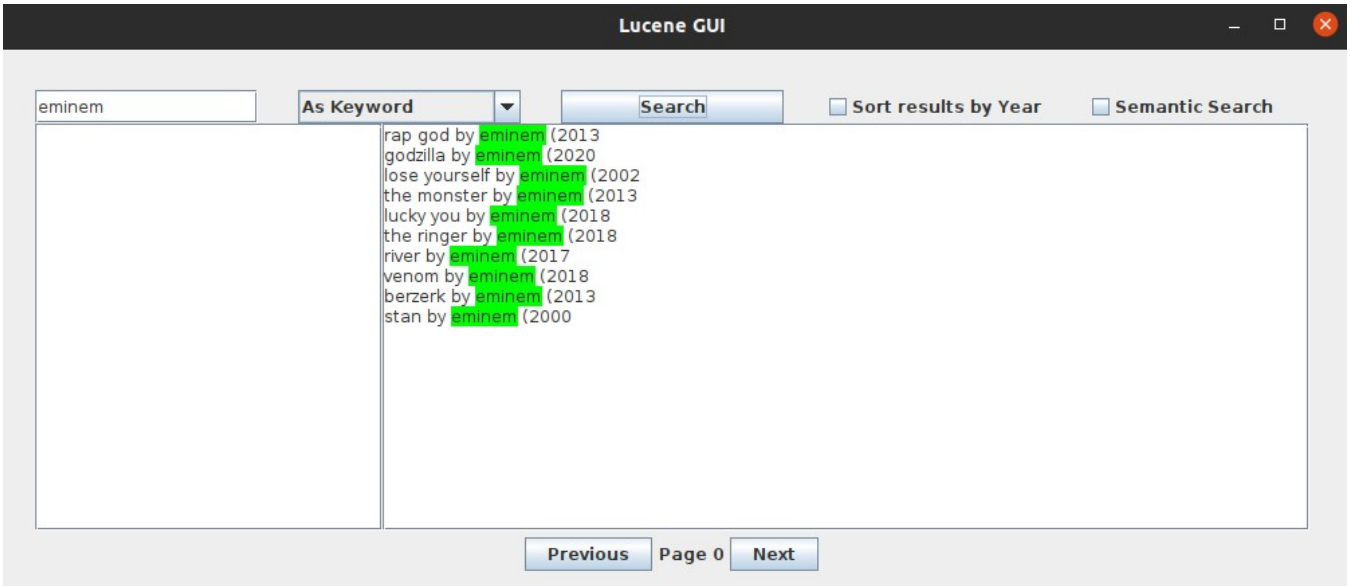


Η κλάση LuceneGUI υλοποιεί ένα γραφικό περιβάλλον χρήστη για την αλληλεπίδραση με έναν Lucene Search Engine. Οταν η αναζήτηση γίνεται χωρίς grouping εμφανίζονται το πολύ 10 αποτελέσματα ανά σελίδα.

Ο χρήστης μπορεί να επιλέξει αναζήτηση στα πεδία Artist, Title, Album, Date, Lyrics, Year ή να αναζητήσει μια φράση-κλειδί(“As Keyword”) σε όλα τα πεδία.

Αναζήτηση ως Keyword

Αναζητούμε την λέξη αυτούσια χωρίς να υπολογίσουμε ορθογραφικά ή άλλες τροποποιήσεις.

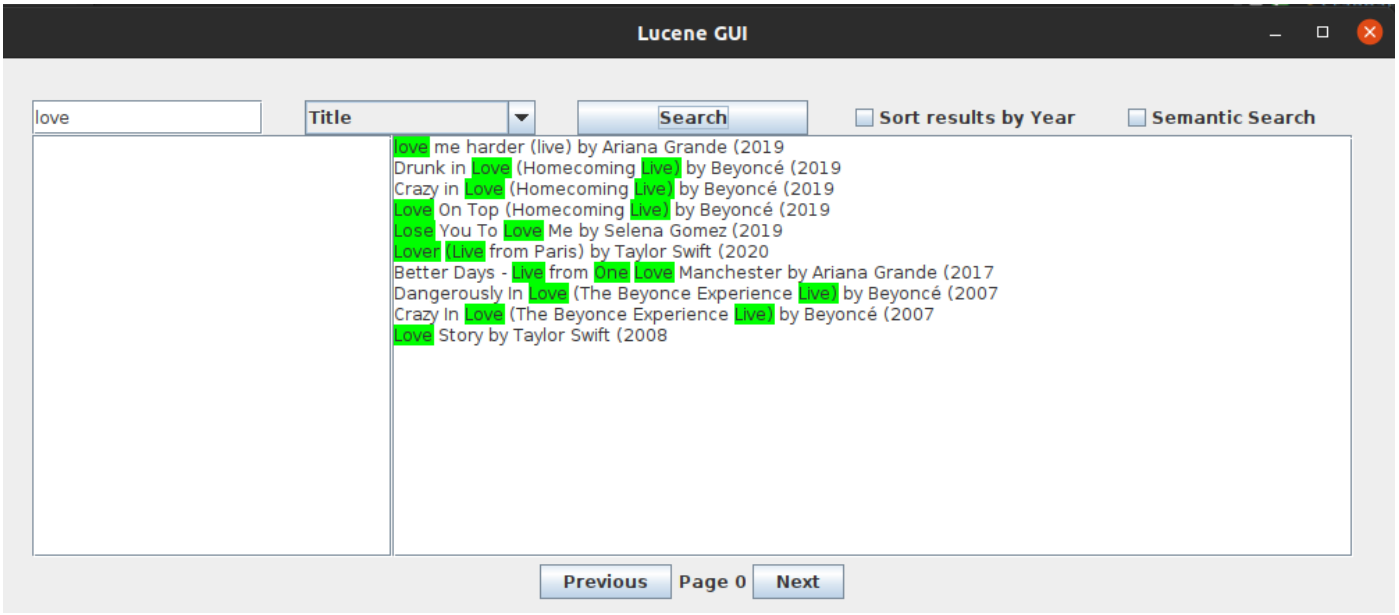


Highlighting

Η μέθοδος displayResults(List<Document> results) εμφανίζει τα αποτελέσματα αναζήτησης και πραγματοποιεί το highlighting των όρων αναζήτησης στο κείμενο των αποτελεσμάτων.

Η διαδικασία του highlighting στο πεδίο κειμένου των αποτελεσμάτων περιλαμβάνει τα εξής βήματα:

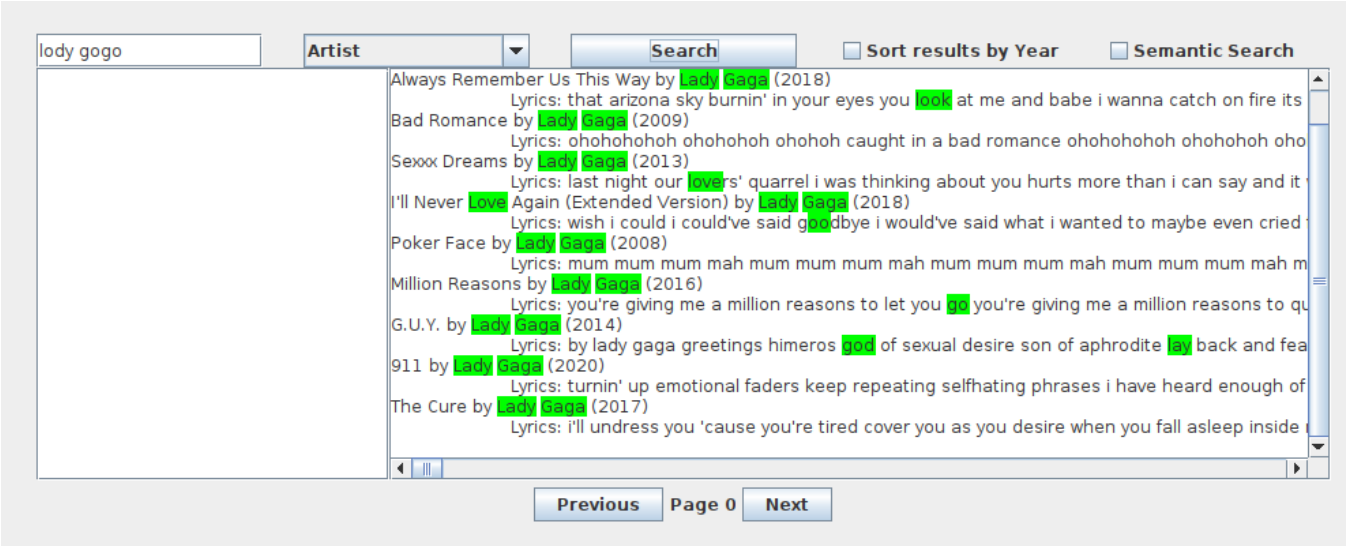
1. Παίρνουμε τον όρο αναζήτησης από το ερώτημα αναζήτησης.
2. Χωρίζουμε το κείμενο των αποτελεσμάτων σε λέξεις.
3. Ελέγχουμε κάθε λέξη για αντιστοιχία με τον όρο αναζήτησης χρησιμοποιώντας την απόσταση επεξεργασίας(edit distance), με την βοήθεια της μεθόδου getEditDistance(String word1, String word2).
4. Αν βρεθεί αντιστοιχία, προσθέτουμε highlight στη θέση της λέξης στο κείμενο των αποτελεσμάτων.



Διόρθωση τυπογραφικών λαθών

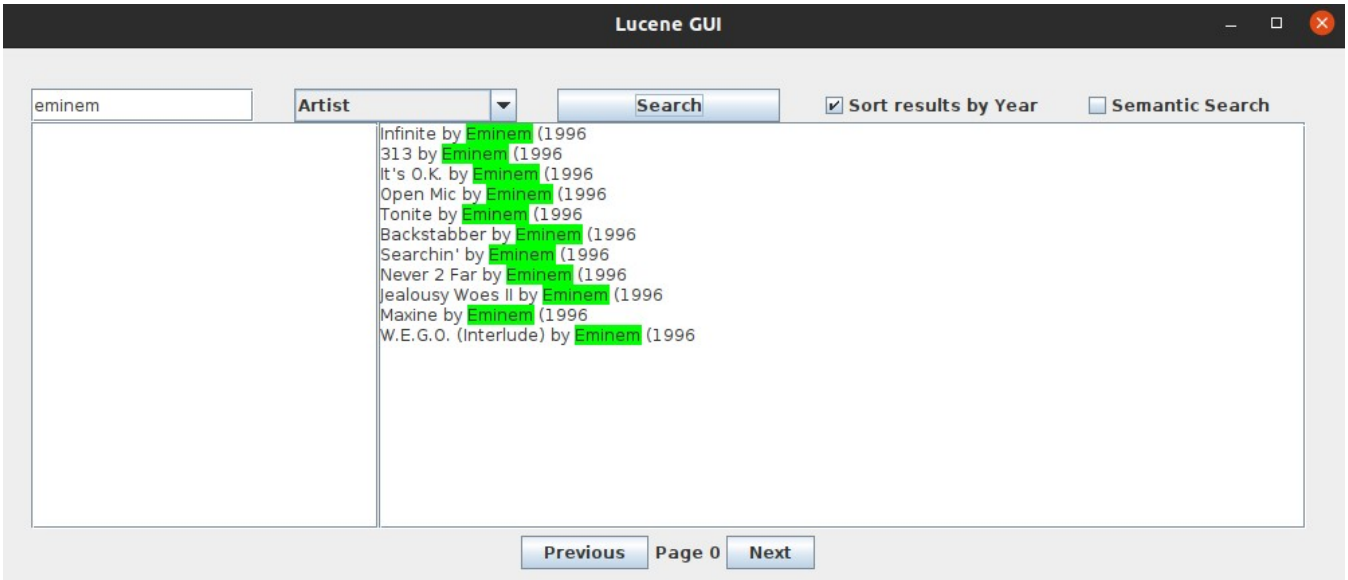
Ο αλγόριθμος είναι σε θέση να αναζητήσει και να επιστρέψει αποτελέσματα που περιέχουν λέξεις που μοιάζουν με την αρχική λέξη αλλά έχουν κάποιο επίπεδο τυπογραφικού λάθους.

1. “lody gogo”, αντι για “Lady Gaga” στο πεδίο “Artist”

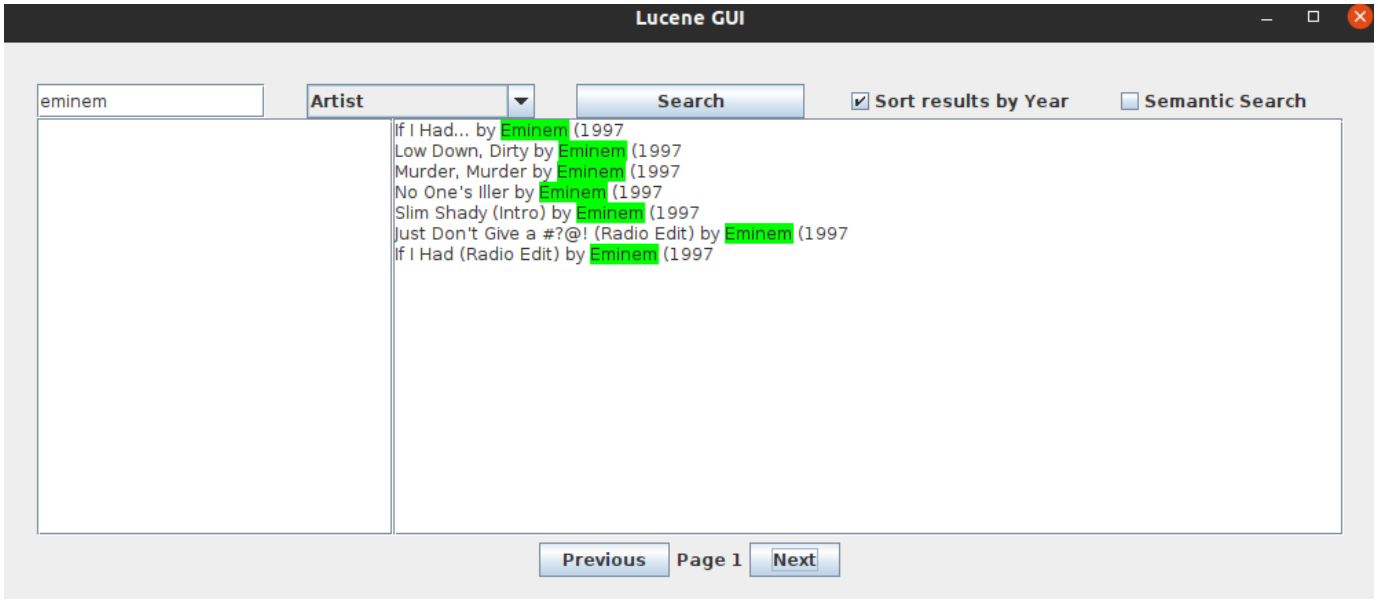


Grouping

Δίνεται η δυνατότητα grouping με βάση το πεδίο “Year”. Με αυτον τον τρόπο σε κάθε σελίδα θα υπάρχει ένα μοναδικό group(που μοιράζεται δηλαδή κοινό “Year”).



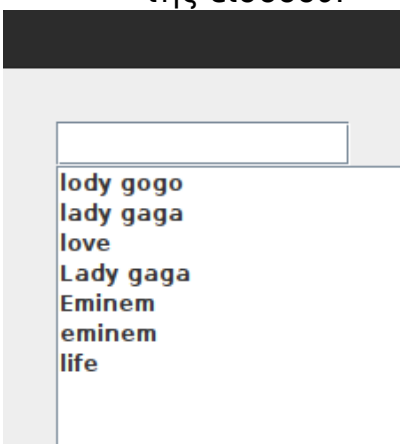
Page 0

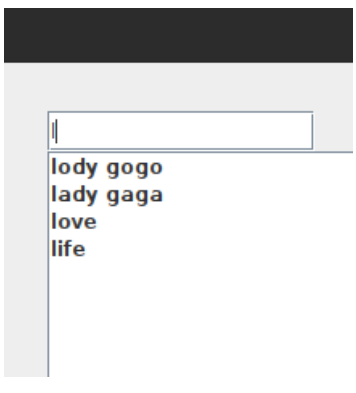


Page 1

Ενναλακτικά ερωτήματα

Παρακάτω φαίνεται ενα παράδειγμα του τρόπου λειτουργίας της πρότασης εναλλακτικών ερωτημάτων. Αρχικά φαίνεται όλο το ιστορικό αναζήτης και έπειτα αναλογά με την είσοδο του χρήση μειώνει τις προτάσεις του βάση της εισόδου.





Σημασιολογική αναζήτηση

Αναζήτηση με βάση το νόημα της λέξης.