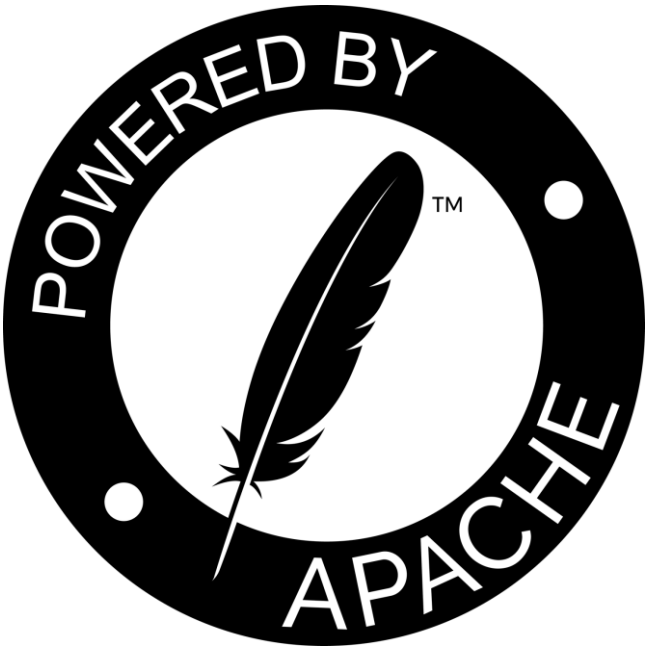


Μηχανή Αναζήτησης

ΣΕ ΑΡΘΡΑ ΤΗΣ WIKIPEDIA

Μιλτιάδης Βασιλειάδης 2944 | Ανάκτηση Πληροφορίας | Ιούνιος 2020
cs02944@uoi.gr | miltosv@outlook.com.gr

Περίληψη

Το παρόν αποτελεί τελική τεχνική αναφορά της εξαμηνιαίας εργασίας στο μάθημα της Ανάκτησης Πληροφορίας. Η φετινή εργασία ασχολείται με την αναζήτηση σε συλλογή από άρθρα που έχουν συλλεχθεί από την Wikipedia

https://drive.google.com/drive/folders/1iOdujNI-7nC9AcqQaMvOvpUmFxoQ4lVg?usp=sharing

Η ΣΥΛΛΟΓΗ ΤΩΝ ΔΕΔΟΜΕΝΩΝ

Το corpus δηλαδή τα έγγραφα που θα επεξεργαστούμε και θα αναζητεί το σύστημα προέρχονται από το Kaggle https://www.kaggle.com/jrobischon/wikipedia-movie-plots/data είναι αρκετά μεγάλο αφού έχει πληροφορία για πάνω από 35 χιλιάδες ταινίες. Είναι επίσης αρκετά βολικό γιατί όπως έγινε η συλλογή των δεδομένων το csv αρχείο ήδη παρέχει μια καλή επιλογή για τα fields των documents. Τα fields κρατάνε πληροφορία για τον Τίτλο, την Χρονιά, τους Ηθοποιούς, τον Σκηνοθέτη, την χώρα προέλευσης, το είδος (αν είναι πχ δραματική) και τέλος την πλοκή. Για λόγους επιδόσεων επέλεξα να περικόψω την συλλογή σε περίπου 2000 έγγραφα.

Parsing the Dataset

Υπάρχει η βιβλιοθήκη Appache Commons CSV που παρέχει ευκολία στο διάβασμα του dataset διαβαζόντας γραμμή γραμμή το αρχειο και το κάθε field (δηλαδή στήλη) ξεχωριστά.

Creating the Index

Δημιουργώ τον index writer που θα γράφει το ευρετήριο σε κατάλογο στον δίσκο (αντί για volatile ευρετήριο στην RAM).

Καλώ τις εντολές

```
IndexWriterConfig indxCnfg= new IndexWriterConfig(new
EnglishAnalyzer());
indxCnfg.setSimilarity(new BM25Similarity());

writer=new IndexWriter(dir,indxCnfg);
```

Αποφάσισα να πειραματιστώ με άλλου τύπου scoring και η lucene 8 εχει υλοιποιημένη κλάση για BM25 similarity, επίσης χρησιμοποίησα την EnglishAnalyzer για το stemming του κειμένου.

Από τα δεδομένα που κάνουμε extract δημιουργώ τα εξής πεδία

TextField("Title"),TextField("Origin"),TextField("Plot"),StringField("Year"),TextField("Cast"),TextField("Director"),StringField("Link"),TextField("Genre")

TextField γιατί περίεχουν κείμενο το οποίο πρέπει να περάσει από τον analyser. Οι λέξεις που περνάνε από τον analyser δέχονται και διαφορετικές εκδοχές επίσης γίνονται tokenized το οποίο είναι χρήσιμο για το cast αφου μπορεί να επιστρέψει διαφορετική ορθογραφία κάποιου ονόματος και πιάνει την εναλλαγή μεταξύ πεζών-κεφαλαίων.

Το πεδίο Year είναι απλά string γιατί δεν χρειάζεται να περασει από τον analyser ο χρήστης θα αναζητεί απλά την χρονιά της ταινίας.

Τα fields εισάγονται σε ένα νέο document και το document εισάγεται στο ευρετήριο. Τα πεδία αυτά αποθηκεύονται μαζί με την πληροφορία τους (FIELD.STORE.YES) στο ευρετήριο για να μπορούμε να προβάλλουμε ως αποτελέσματα στο GUI. Στο τέλος του parsing κάνουμε close και τον indexWriter

SEARCHING

Η κλάση searcher δημιουργείται από τον controller και έχει αναφορά σε αυτόν για το πέρασμα αντικειμένων μεταξύ των κλάσεων για να υλοποιηθεί η αναζήτηση.

Η εφαρμογή υποστηρίζει την αναζήτηση και προβολή των δέκα καλύτερων αποτελεσμάτων βάση του όρου της αναζήτησης, αυτό επιτυγχάνεται με την χρήση των μεθόδων IndexSearcher.search() για να υπολογίσει τα πρώτα 10 καλύτερα και IndexSearcher.searchAfter() .

Οι ερωτήσεις έρχονται σε μορφή String και χρειάζεται να γίνουν parse σε Query χρησιμοποιώ την MultiFieldQueryParser δίνοντας ορίσματα τα fields και τον Αναλυτή

```
parser = new MultiFieldQueryParser({"Plot","Genre","Director","Cast","Title","Year","Origin"}, new EnglishAnalyzer());
```

Για την εκτέλεση της αναζήτησης καλείται από τον controller ή μέθοδος execute() της Searcher Class . Η μέθοδος παίρνει ως όρισμα έναν πίνακα που περιέχει μεταξύ άλλων και την συμβολοσειρά βάση της οποίας θα γίνει αναζήτηση. Η συμβολοσειρά αυτή περνάει από την Query Parser. Υποστηρίζονται οι λειτουργίες αναζήτησης βάση field πχ Title:"The Right Way" οι λογικοί τελεστές AND OR οι τελεστές + και – για inclusion exclusion αντιστοίχα καθώς και τα wildcards * και ? όπως περιγράφονται και στο Documentation της lucene.

https://lucene.apache.org/core/2_9_4/queryparsersyntax.html

Η execute παίρνει από τον controller την σελίδα των αποτελεσμάτων, αυτό καθορίζει ποια 10 εγγραφα θα εμφανίζονται αν η σελίδα = 0 τότε καλείται η search η οποία επιστρέφει μια συλλογή TopDocs που περιέχει τα 10 καλύτερα έγγραφα για την αναζήτηση αυτή. Αν η σελίδα είναι παραπάνω από 1 τότε γίναται αναζήτηση στο προηγουμενο TopDocs για να βρεθεί το τελευταίο ScoreDoc εκείνης της αναζήτησης, έπειτα ή searchAfter ψάχνει και επιστρέφει τα επόμενα 10 ScoreDocs για την

αναζήτηση. Τα αποτελέσματα της αναζήτησης επιστρέφονται με σειρά των score από το μεγαλύτερο προς το μικρότερο.

Αμφότερες οι 2 αναζητήσεις μετά την εκτέλεση τους, κάνουν set αυτό το TopDocs αντικείμενο στον Controller για να προβληθούν στο front end.

CONTROLLER AND FRONT-END

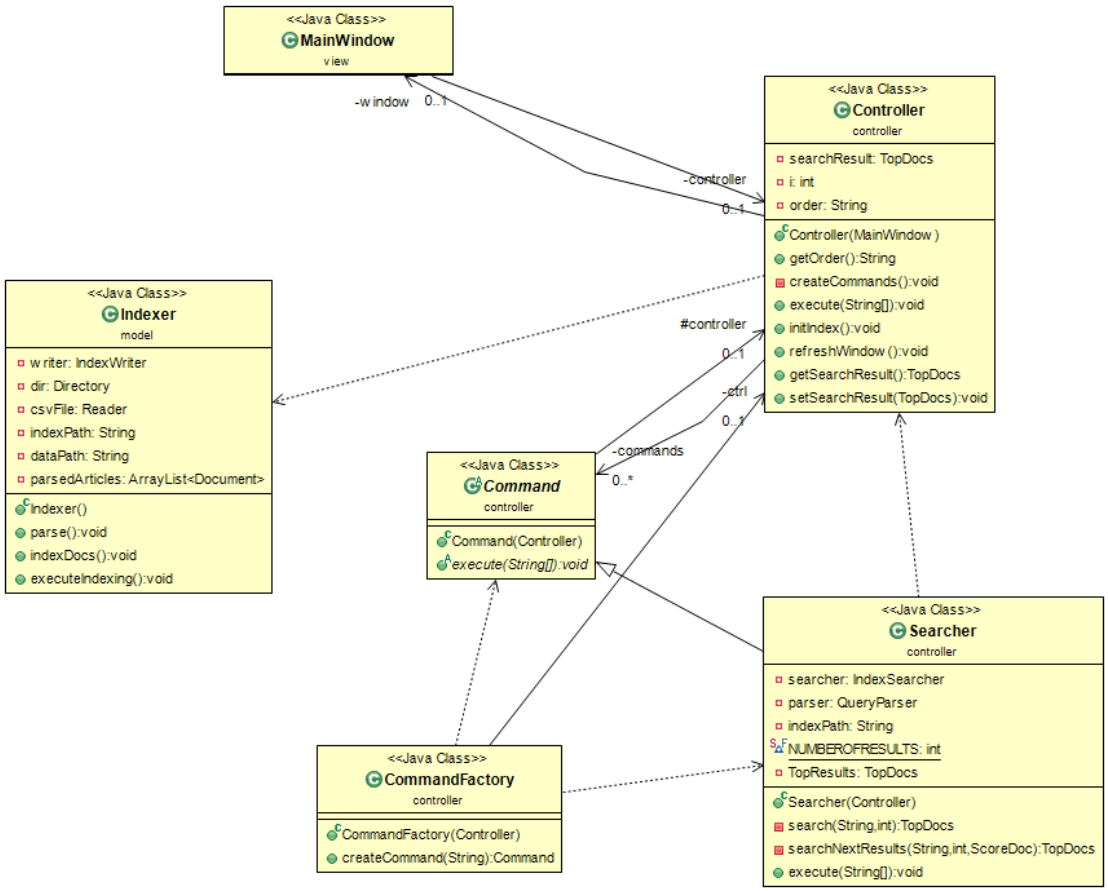
Η διεπαφή της εφαρμογής είναι αρκετά απλή στην εμφάνιση, αποτελείται από μία περιοχή όπου ο χρήστης βάζει την ερώτηση, ένα πλαίσιο όπου θα εμφανίζονται τα αποτελέσματα καθώς και τα κουμπιά για έναρξη της αναζήτησης και για την μετάβαση στα επόμενα 10 καλύτερα αποτελέσματα. Επίσης προστέθηκαν κουμπιά για τους τελεστές για την αναζήτηση βάση πεδίου, για να διευκολύνει ίσως τον χρήστη στην σύνταξη της ερώτησης.

Το front-end επίσης κρατάει πληροφορία για τις προηγούμενες αναζητήσεις. Μετά την επιστροφή αποτελεσμάτων αναζήτησης από τον controller οι λέξεις των αποτελεσμάτων που ταιριάζουν στο ερώτημα επισημαίνονται με κόκκινο χρώμα.

Ο controller επιτελεί κλάση που συνδέει το front end με το back end. Η σημαντικότερη λειτουργία του είναι η δημιουργία του τελικού κειμένου που εμφανίζεται στο front end.

Από το TopDocs που επιστρέφεται από την Searcher ο controller διατρέχει την λίστα με τα ScoreDocs και προσθέτει στο string των αποτελεσμάτων το ID του εγγράφου, το ΣΚΟΡ του και χρησιμοποιώντας έναν indexReader δημιουργεί μια μικρή περίληψη του εγγραφου χρησιμοποιώντας τον Τίτλο της Ταινίας, το έτος που κυκλοφόρησε το link στο άρθρο της Wikipedia και την περίληψη της πλοκής. Τέλος επιστρέφει τα αποτελέσματα στο front end.

UML DIAGRAM


<<Java Class>>
MainWindow
view
-window
0..1
<<Java Class>>
Controller
controller
searchResult: TopDocs
i: int
order: String
Controller(MainWindow)
getOrder():String
createCommands():void
execute(String[]):void
initIndex():void
refreshWindow():void
getSearchResult():TopDocs
setSearchResult(TopDocs):void
-controller
0..1
<<Java Class>>
Indexer
model
writer: IndexWriter
dir: Directory
csvFile: Reader
indexPath: String
dataPath: String
parsedArticles: ArrayList<Document>
Indexer()
parse():void
indexDocs():void
executeIndexing():void
#controller
0..1
-ctrl
0..1
<<Java Class>>
Command
controller
Command(Controller)
execute(String[]):void
-commands
0..*
<<Java Class>>
Searcher
controller
searcher: IndexSearcher
parser: QueryParser
indexPath: String
NUMBEROFRESULTS: int
TopResults: TopDocs
Searcher(Controller)
search(String,int):TopDocs
searchNextResults(String,int,ScoreDoc):TopDocs
execute(String[]):void
<<Java Class>>
CommandFactory
controller
CommandFactory(Controller)
createCommand(String):Command