**Η μηχανή αναζήτησης paperWorld**

Στέφανος Γκερς-Κουτσογιάννης, 5046

Ηλίας Γρατσιάς, 4941

Ανάκτηση Πληροφορίας, Εαρινό Εξάμηνο 2024

**Πίνακας Περιεχομένων**

1. Εισαγωγή
2. Συλλογή
3. Ανάλυση κειμένου και κατασκευή ευρετηρίου
4. Αναζήτηση
5. Παρουσίαση αποτελεσμάτων

**Κεφάλαιο 1:** Εισαγωγή

Η μηχανή αναζήτησης paperWorld, είναι μια μηχανή αναζήτησης που επιτρέπει στους χρήστες την ανάκτηση επιστημονικών άρθρων που δημοσιεύτηκαν στο συνέδριο NeurIPS(Neural Information Processing Systems) την χρονική περίοδο 1987-2019. Στόχος της μηχανής αναζήτησης είναι η γρήγορη και εύκολη ανάκτηση επιστημονικών άρθρων ,που συμπίπτουν με το ερώτημα που θέτει ο χρήστης.

**Κεφάλαιο 2:** Συλλογή

Την συλλογή μπορεί να κατεβάσει ο καθένας από την σελίδα της Kaggle(<https://www.kaggle.com/datasets/rowhitswami/nips-papers-1987-2019-updated>). Στην δυλλογή αυτή υπάρχουν δύο αρχεία, το papers.csv και το authors.csv. Εμείς κάναμε συνένωση των δύο αρχείων στο πεδίο source\_id, και θα χρησιμοποιήσουμε αυτό το αρχείο ως βάση της συλλογής μας.

Στο αρχείο που προκύπτει από την συνένωση αυτή, μετά από προ-επεξεργασία των δεδομένων, κρατήσαμε τα πεδία:

* year: η χρονιά δημοσίευσης του επιστημονικού άρθρου
* title: ο τίτλος του επιστημονικού άρθρου
* abstract: η περίληψη του άρθρου
* full\_text: όλο το επιστημονικό άρθρο
* authors: οι συγγραφείς του εγγράφου

Μετά την προ-επεξεργασία, η συλλογή μας αποτελείται από 6008 έγγραφα.

Τα βήματα για την τελική μορφή του εγγράφου φαίνονται στο συνημμένο notebook(src/main/resources/IR\_preprocessing.html).

**Κεφάλαιο 3:** Ανάλυση κειμένου και κατασκευή ευρετηρίου

Για την ανάλυση κειμένου, θα χρησιμοποιήσουμε τον StandardAnalyzer, ο οποίος μετατρέπει το έγγραφο σε μια σειρά από tokens, κάνει απαλοιφή stopwords και μετατροπή των token σε lowercase.

Για την κατασκευή του ευρετηρίου, δημιουργούμε ένα FSDirectory, μια δομή δεδομένων στην οποία θα αποθηκευτεί το ευρετήριο μας. Η δομή FSDirectory, αποθηκεύει το ευρετήριο μας στο δίσκο.

Αφού το ανοίξουμε και κρατήσουμε την αναφορά, αρχικοποιούμε ένα αντικείμενο IndexWriterConfig, το οποίο δέχεται ως όρισμα έναν Αnalyzer , στην δική μας περίπτωση τον StandardAnalyzer. Χρειαζόμαστε το αντικείμενο IndexWritterConfig, έτσι ώστε να μπορούμε να το περάσουμε σαν όρισμα σε έναν IndexWriter, μαζί με την αναφορά στη δομή FSDirectory, έτσι ώστε τελικά να μπορούμε να προσθέσουμε στοιχεία στο ευρετήριο.

Για να γράψουμε στο ευρετήριο, για κάθε αρχείο της συλλογής μας δημιουργούμε μια δομή Document, η οποία απαρτίζεται από Field διάφορων ειδών. Η δομή Document παίζει τον ρόλο του container, όπου μέσα σε αυτό υπάρχουν πεδία, στη δική μας περίπτωση πεδία τύπου TextField, τα οποία και αποθηκεύουμε.

Αφού μετατρέψουμε κάθε αρχείο της συλλογής μας σε Document με τα αντίστοιχα Fields και τα προσθέσουμε στο ευρετήριο μας, κλείνουμε το ευρετήριο μας.

Η ευρετηριοποίηση της συλλογής μας και κατά συνέπεια και η δημιουργία Documents, γίνεται μέσω της κλάσης Indexer. Αυτή, έχει τη μέθοδο index(), η οποία φτιάχνει το ευρετήριο. Αυτό γίνεται εκτός εφαρμογής, δηλαδή φτιάχνουμε πρώτα το ευρετήριο και μετά χρησιμοποιούμε την εφαρμογή.

**Κεφάλαιο 4:** Αναζήτηση

Για να γίνει οποιουδήποτε είδος αναζήτησης, η Lucene χρειάζεται να ανοίξει το ευρετήριο που έχει προηγουμένως δημιουργηθεί, και να δημιουργήσει ένα αντικείμενο IndexSearcher, μέσω του οποίου θα γίνει η αναζήτηση. Ο indexSearcher, παίρνει ως όρισμα το Directory, στο οποίο έχουμε δημιουργήσει το ευρετήριο μας.

Η βασική μονάδα της Lucene για την διατύπωση ερωτήσεων είναι η κλάση Query, η οποία προσφέρει πολλές υποκλάσεις, με ιδιαίτερα χαρακτηριστικά, όπως οι TermQuery, WildCardQuery, PhraseQuery, FuzzyQuery και άλλες.

Η Lucene όμως προσφέρει και ένα ιδιαίτερα χρήσιμο εργαλείο για την αυτοματοποίηση της δημηουργίας κατάλληλων ερωτήσεων, μέσω της κλάσης QueryParser. Αυτή η κλάση, δέχεται σαν ορίσματα το πεδίο για το οποίο θα δημιουργήσει το αντίστοιχο Query και έναν Analyzer για την ανάλυση της ερώτησης σε tokens. Η μέθοδος parse(), δέχεται σαν όρισμα κείμενο που θέλουμε να αναζητήσουμε και δημιουργεί αυτόματα το κατάλληλο Query για το πεδίο αυτό.

Είναι σημαντικό, όταν χρησιμοποιούμε το QueryParser για την δημιουργία Queries, σαν Analyzer να βάζουμε τον ίδιο Analyzer που χρησιμοποιήθηκε στην ευρετηριοποίηση της συλλογής, έτσι ώστε να έχουμε συνέπεια στα δεδομένα μας, δηλαδή τα tokens της ερώτησης μας να αντιστοιχούν σε tokens που είναι αποθηκευμένα στο ευρετήριο μας.

Η υλοποίηση μας υποστηρίζει αναζήτηση μέσω QueryParser για όλα τα πεδία, με την εξαίρεση πως για όλα τα πεδία εκτός του full\_text, ο χρήστης πρέπει να δηλώσει ρητά ποιο πεδίο θέλει να ψάξει. Οι επιλογές είναι:

* “year:text”: ψάχνει για text στο πεδίο year τον όρο/φράση text
* “title:text”: ψάχνει στο πεδίο title τον όρο/φράση text
* “abstract:text”: ψάχνει στο πεδίο abstract τον όρο/φράση text
* “name:text”: ψάχνει στο πεδίο authors τον όρο/φράση text
* “text”: ψάχνει στο πεδίο full\_text τον όρο/φράση text

Η ευελειξία που μας προσφέρει ο QueryParser, είναι πως δε χρειάζεται κάθε φορά να δηλώνουμε τι ερώτηση θέλουμε να κάνουμε και απλά μας την δημιουργεί αυτός.

Αυτό πρακτικά σημαίνει ότι, η μηχανή αναζήτησης μας υποστηρίζει όλους(ή αν όχι όλους τους περισσότερους) βασικούς τύπους ερωτήσεων της Lucene.

Παραδείγματος χαριν, η έκφραση «title:K-me\*», αφού την πάρει ο QueryParser, θα δημιουργήσει ένα WildcardQuery, το οποίο τελικά θα μας επιστρέψει όλα τα έγγραφα, που έχουν στον τίτλο τους το «Κ-me» και ακολουθείται από οσουσδήποτε χαρακτήρες, ενώ η έκφραση «title:Learning OR Boosting», αφού την πάρει ο QueryParser, θα μας επιστρέψει τα έγγραφα στα οποία στον τίτλο υπάρχει μια από τις δύο λέξεις.

Ο IndexSearcher, μέσω της μεθόδου search(), με ορίσματα το ερώτημα που δημιουργεί ο QueryParser, τον αριθμό αποτελεσμάτων που θα ανακτηθούν και ένα αντικείμενο Sort, υπεύθυνο για την διάταξη των εγγράφων , επιστρέφει μια λίστα TopDocs, η οποία περιέχει αναφορές στα Documents που έχουν το μεγαλύτερο score, με βάση τον τρόπο scoring που βαθμολογεί η Lucene τα έγγραφα.

Για κάθε ένα από αυτά τα έγγραφα, δημιουργούμε ένα αντικείμενο Result, μια κλάση ορισμένη από εμάς, η οποία αντιπροσωπεύει τα έγγραφα που ανακτήθηκαν. Όλα αυτά τα έγγραφα όταν μετασχηματιστούν σε Results, επιστρέφονται σε μια λίστα στο χρήση που κάλεσε την μέθοδο search.

Η αναζήτηση μέσω των μηχανισμών της Lucene, επιτυγχάνεται μέσω της κλάσης Searcher, ενώ η δημιουργία κατάλληλων ερωτήσεων επιτυγχάνεται μέσω της κλάσης QueryBuilder.

**Κεφάλαιο 5: Παρουσίαση αποτελεσμάτων**

Για την παρουσίαση αποτελεσμάτων, χρησιμοποιήσαμε την βιβλιοθήκη JavaFX, για την δημιουργία ενός απλού desktop app. Αυτό αποτελείται από δύο βασικές σελίδες, μια αρχική σελίδα που εμφανίζεται στο χρήστη όταν θα ανοίξει την εφαρμογή, και μια δεύτερη σελίδα με links στα αρχεία που επιστρέφονται ως αποτελέσματα της αναζήτησης.

Στην Εικόνα 1, φαίνεται η αρχική σελίδα της εφαρμογής μας. Σε αυτή, υπάρχει ένα TextField, το οποίο χρησιμοποιούμε σαν Search Bar για την διατύπωση ερωτήσεων.

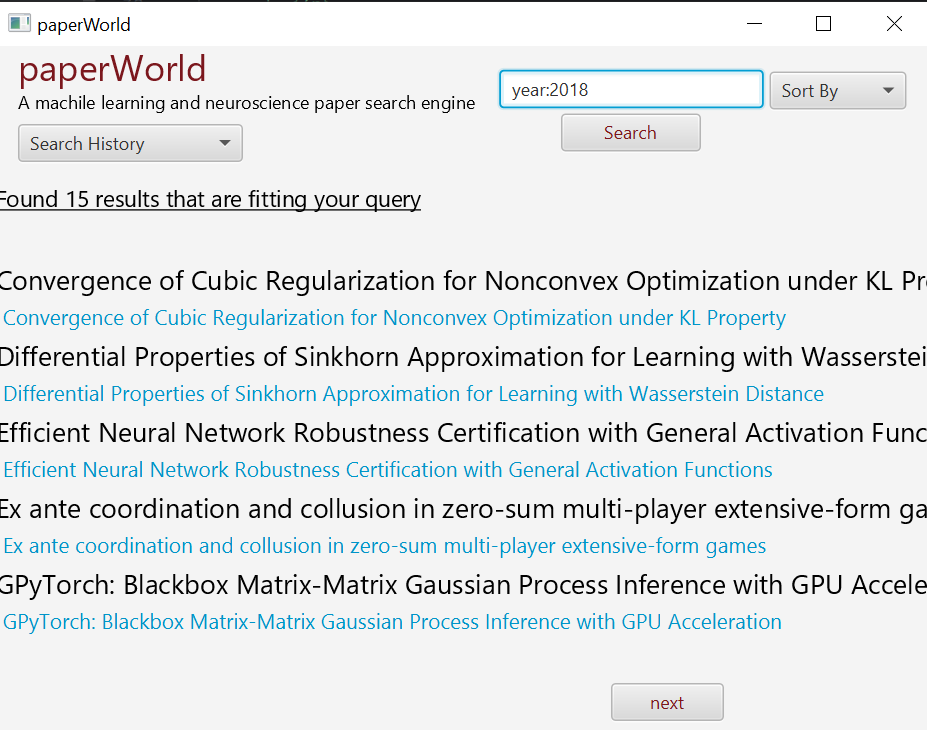
Στην Εικόνα 2, απεικονίζεται η σελίδα με τα αποτελέσματα. Σε αυτή, βλέπουμε τα αποτελέσματα της αναζήτησης ανά 5, τον τίτλο τους και από κάτω ένα link, το οποίο αν πατήσει ο χρήστης, του ανοίγει ένα παράθυρο με τις πληροφορίες του αρχείου.

Δίνεται επίσης η δυνατότητα εναλλαγής σελίδων μεταξύ των αποτελεσμάτων, μέσω των κουμπιών previous και next και το σύστημα κρατάει ιστορικό αναζήτησης. Η επιλογή Sort By δεν λειτουργεί. Αν λειτουγούσε, δεν είχε σκοπό την ταξινόμηση των αρχείων με βάση τα αποτελέσματα, αλλά περνάει ως παράμετρος στην μέθοδο search() της Lucene, και η Lucene απλά επιστρέφει τα έγγραφα με βάση τη διάταξη που έχουμε περάσει σαν όρισμα.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, ιστοσελίδα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 1: Landing page της εφαρμογής



Εικόνα 2: Η σελίδα αποτελεσμάτων

Η διεπαφή χρήστη φτιάχτηκε χρησιμοποιόντας την εφαρμογή SceneBuilder, μια εφαρμογή για την ανάπτυξη γραφικών περιβάλλοντων για desktop apps.