Blue text on a black background

Description automatically generated

***Σχολή Μηχανικών***

***Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών***

***Ανάκτηση Πληροφορίας***

***Δημιουργία μηχανής αναζήτησης***

*ΕΥΦΡΟΣΥΝΗ ΒΑΡΣΟΥ 21390021*

*ΑΓΓΕΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ ΜΕΝΤΖΕΛΟΣ 21390132*

[Github Project Link](https://github.com/frottori/IR_Search_Engine/blob/main/Project/Project.ipynb)

Βήμα 1. Συλλογή δεδομένων

## Περιγραφή συνόλου δεδομένων

Το σύνολο δεδομένων είναι επιλεγμένα άρθρα που επιλέξαμε από το Wikipedia. Μερικά από αυτά επικεντρώνονται σε θεματολογίες που αφορούν την επιστήμη, την τεχνολογία και άλλα σχετικά πεδία. Άλλα άρθρα επικεντρώνονται σε άλλα τελείως διαφορετικά θέματα όπως την τέχνη, ιστορία κτλ. Επιλέξαμε άρθρα με παρόμοια θεματολογία σχετικά με την επιστήμη για να δούμε αν με πολλές ομοιότητες (π.χ. λεξιλόγιο) η μηχανή αναζήτησης δίνει αποδοτικά αποτελέσματα. Ωστόσο, επιλέξαμε και άρθρα που δεν έχουν κάποια διακριτή συσχέτισή για να δούμε και αν η μηχανή αναζήτησης σε τελείως διαφορετικά θέματα βρει και εκεί κάποιες ομοιότητες που δεν περιμέναμε. Γενικά, επιλέξαμε μια ποικιλία από άρθρα για να μπορούμε να εξετάσουμε αποτελεσματικά την μηχανή αναζήτησης σε σχέση με το ερώτημα του χρήστη.

Περιγραφή της μεθοδολογίας συλλογής

Για την συλλογή των δεδομένων, υλοποιήσαμε έναν web crawler με BeautifulSoup. Ειδικότερα, δημιουργήσαμε μια λίστα από άρθρα από την Wikipedia και για κάθε URL που αντιστοιχεί σε κάθε από αυτά καλούμε την συνάρτηση crawl\_wikipedia(). H συνάρτηση αυτή τώρα, είναι υπεύθυνη για την εξαγωγή δεδομένων (τίτλου και περιεχομένου) από την κάθε HTML σελίδα. Ειδικότερα, στέλνει ένα HTTP request στην διεύθυνση URL και με την βοήθεια της βιβλιοθήκης BeautifulSoup βρίσκει το h1 tag που αντιστοιχεί στον τίτλο του άρθρου καθώς και όλα τα p tags που αντιστοιχούν στο περιεχόμενο του άρθρου. Με αυτόν τον τρόπο, δημιουργήσαμε μια δομή dictionary που κάθε τίτλος άρθρου αντιστοιχεί σε κάποιο περιεχόμενο και έτσι το αποθηκεύουμε σε ένα .json και .csv (wiki\_data) με αυτήν την μορφή.

Βήμα 2. Προεπεξεργασία κειμένου (Text Processing)

Για την προεπεξεργασία κειμένου, ακολουθήσαμε μια σειρά από ενέργειες για να «καθαρίσουμε» τα δεδομένα. Τα άρθρα του Wikipedia όπως γνωρίζουμε είναι γεμάτο από πηγές στο κείμενο τους που συμβολίζονται με δύο αγκύλες και ανάμεσα τους έναν αριθμό που αντιστοιχεί σε μια παραπομπή στο τέλος του άρθρου ή σε link άλλου άρθρου (π.χ. [1]). Οι πηγές όμως δεν μας ενδιαφέρουν για την μηχανή αναζήτησης καθώς δεν παρέχουν κάποια πληροφορία στο περιεχόμενο. Για αυτόν τον λόγο και τις αφαιρούμε με το αντίστοιχο regex. Στην συνέχεια, αφαιρούμε και τα σημεία στίξης καθώς για τον ίδιο λόγο δεν προσφέρουν κάποια πληροφορία στην μηχανή αναζήτησης.

Ως κύρια τεχνική προεπεξεργασίας επιλέξαμε tokenization με stemming. Δηλαδή, για κάθε άρθρο χωρίσαμε ανά λέξη το κείμενό τους (token) και με την χρήση stemming κρατάμε μόνο την ρίζα της λέξης (π.χ. running → run). Με αυτόν τον τρόπο, όλες οι ομόρριζες λέξεις ομαδοποιούνται στην μία ρίζα καθιστώντας το νόημα του κειμένου πιο συγκεκριμένο/ειδικό. Αυτό καθιστά το κείμενο και συνεπώς το σύνολο δεδομένων λιγότερο πολύπλοκο και βοηθάει την αποδοτικότητα της μηχανής αναζήτησης.

Ο κύριος λόγος που επιλέξαμε την τεχνική stemming αντί της lemmatization είναι η ταχύτητα και λιγότερης πολυπλοκότητας. Το stemming κυρίως χρησιμοποιείται σε μηχανές αναζητήσεις καθώς αν και έχουν πιο απλούς ευριστικούς (heuristic) κανόνες και δεν έχουν τόση ακρίβεια, είναι άκρως πιο αποδοτικοί σε μεγάλα dataset από ότι το lemmatization που σε σύγκριση είναι αργό. Το lemmatization κυρίως χρησιμοποιείται για περισσότερη ακρίβεια στην ανάλυση του κειμένου και όταν θέλουμε να κρατήσουμε την συνοχή του κειμένου για ειδικότερη ανάλυση των εννοιών του κειμένου (π.χ. σε ιατρικά άρθρα όταν το νόημα του κείμενου έχει μεγαλύτερη σημασία). Τέλος, αφαιρέσαμε και όλα τα stop words δηλαδή σύντομες μικρές συχνές λέξεις που συνήθως χρησιμοποιούνται λόγω γραμματικής (π.χ. the, and, but κτλ.). Αυτές οι λέξεις γενικά δεν προσφέρουν κάποιο περιεχόμενο στο νόημα του κειμένου και επιπλέον είναι σε αρκετά μεγάλη συχνότητα. Επομένως, η αφαίρεσή τους καθιστά ακόμα πιο αποτελεσματική την αποδοτικότητα της μηχανής αναζήτησης.

Βήμα 3. Ευρετήριο (Indexing)

Εφόσον, καθαρίσαμε τα δεδομένα τώρα μέσω του stemming έχοντας πλέον πολλές ρίζες βασικών λέξεων μπορούμε να φτιάξουμε το αντεστραμμένο ευρετήριο (inverted index). Το inverted index είναι ένας πίνακας που αντιστοιχίζει λέξεις σε λίστες εγγράφων στα οποία εμφανίζονται αυτές οι λέξεις. Δηλαδή, στις στήλες αντιστοιχούν όλες οι συνολικά μοναδικές λέξεις από όλα τα άρθρα και κάθε γραμμή αποτελείται από το σύνολο των άρθρων με τιμή το πλήθος της κάθε λέξης. Η δομή δεδομένων που διατηρούμε το inverted index είναι ένα dictionary που αποτελείται από άλλα dictionaries. Το εξωτερικό dictionary αντιστοιχίζει τους τίτλους σε dictionaries των λέξεων και αυτά έχουν τιμή το πλήθος των tokens. Συνεπώς, με το ευρετήριο θα μπορέσουμε στην συνέχεια να υλοποιήσουμε την μηχανή αναζήτης καθώς καθιστά την ανάκτηση της πληροφορίας του dataset σημαντικά πιο γρήγορη.

Βήμα 4. Μηχανή αναζήτησης (Search Engine)

Επεξεργασία ερωτήματος (Query Processing)

Πλέον, έχοντας δημιουργήσει το αντεστραμμένο ευρετήριο από το προηγούμενο βήμα μπορούμε να βρούμε για κάποιο δόθεν query από τον χρήστη σε ποιο άρθρο ανήκει. Ειδικότερα, ο χρήστης πληκτρολογεί ένα query με χρήστη Boolean τελεστών (AND, OR, NOT) για παράδειγμα ‘term1 AND term2’ τα οποία γίνονται tokens για κάθε λέξη και operator που χρησιμοποιείται. Συνεπώς, διασχίζοντας όλα τα tokens (περιλαμβανόμενων και τον τελεστών) όποιο είναι λέξη βρίσκουμε τα άρθρα στο οποίο ανήκει ενώ αν είναι τελεστής κάνουμε την αντίστοιχή Boolean πράξη με τα αντίστοιχα άρθρα των κάθε term. Με αυτόν τον τρόπο, όταν χρησιμοποιούμε τον τελεστή AND επιστρέφουμε τα άρθρα που περιέχουν και τις δύο λέξεις, όταν χρησιμοποιούμε τον τελεστή OR επιστρέφουμε τα άρθρα που περιέχουν τουλάχιστον μία από τις δύο λέξεις και όταν χρησιμοποιούμε τον τελεστή NOT όλα τα άρθρα που περιέχουν την δοσμένη λέξη.

Κατάταξη αποτελεσμάτων (Ranking)

Για την κατάταξη των αποτελεσμάτων χρησιμοποιήσαμε 3 αλγόριθμους ανάκτησης τον TF-IDF, Vector Space Model και τον Okapi BM25. Υλοποιήσαμε μια απλή διεπαφή χρήστη στη γραμμή εντολών όπου ο χρήστης αρχικά διαλέγει τον αλγόριθμο που θέλει να χρησιμοποιήσει και στην συνέχεια πληκτρολογεί το query και επιστρέφονται σε ταξινομημένη σειρά τα best matched articles ανάλογα με το score που έχει υπολογίσει ο κάθε αλγόριθμος. Ο αλγόριθμος TF-IDF υλοποιεί την αναζήτηση υπολογίζοντας την συχνότητα εμφάνισης της σε ένα άρθρο (TF) και αντίστοιχα την σπάνια εμφάνιση της λέξης στο σύνολο των άρθρων (IDF) και συνδυάζοντας αυτές τις δύο τιμές βρίσκει την σημασία της δοσμένης λέξης για το σύνολο των δεδομένων. Ο αλγόριθμος Vector Space Model (VSM) λειτουργεί με παρόμοιο τρόπο με τον TF-IDF υπολογίζοντας τα score με τον ίδιο τρόπο αλλά με διαφορά ότι αναπαριστά τα άρθρα σε μορφή διανύσματος. Στην συνέχεια, με βάση το κάθε διάνυσμα που αναπαριστά ένα άρθρο ο αλγόριθμος τα συγκρίνει υπολογίζοντας το cosine similarity μεταξύ τους. O ΒΜ25 αντίστοιχα υλοποιεί την αναζήτηση υπολογίζοντας την πιθανότητα ύπαρξης μιας λέξης στο άρθρο ενσωματώνοντας και αυτός το TF, IDF στον τύπο υπολογισμού.

Βήμα 5. Αξιολόγηση συστήματος

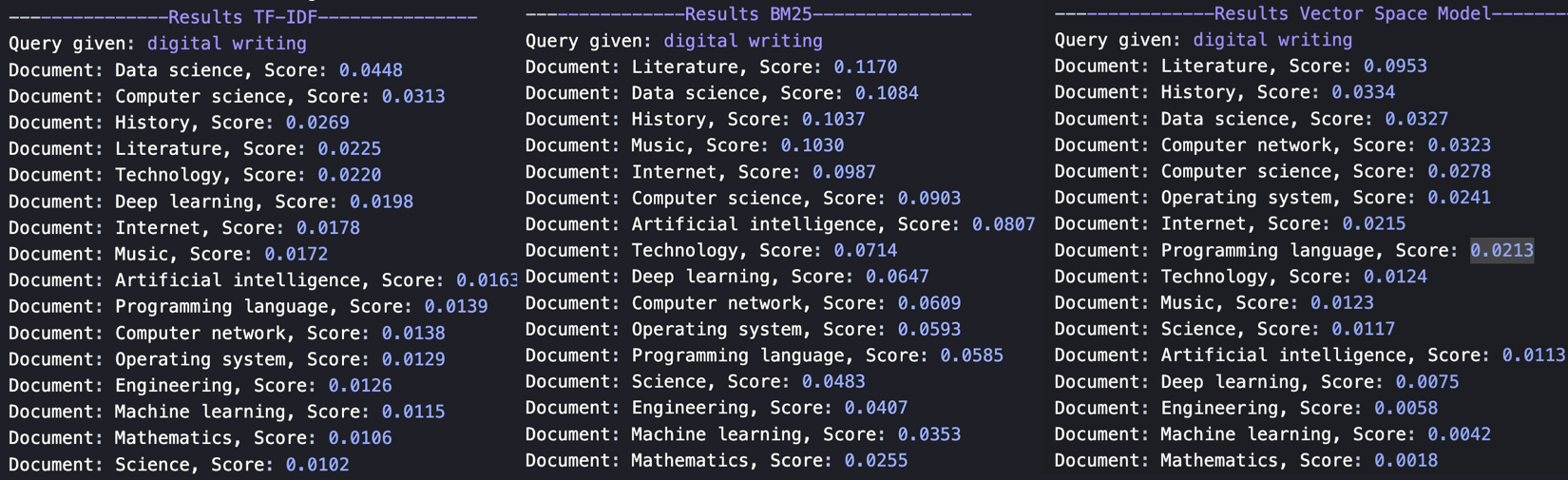
Για την αξιολόγηση του συστήματος χρησιμοποιήσαμε τις μετρικές Precision, Recall, F1-score και το MAP (Mean Average Precision). Το Precision υπολογίζει πόσο αξιόπιστες είναι οι θετικές προβλέψεις (true positives) του μοντέλου, δηλαδή με υψηλό precision οι περισσότερες προβλέψεις του αλγόριθμου ανάκτησης είναι σωστές. Το Recall αντίστοιχα, υπολογίζει την ικανότητα του αλγόριθμου να αναγνωρίσει τα θετικά δείγματα. Το F1-score υπολογίζει τον μέσο όρο του precision και του recall αξιολογώντας την συνολική απόδοση του αλγόριθμου. Τέλος, το MAP υπολογίζει τον μέσο όρο των precision των άρθρων για κάθε αναζήτηση και τις αθροίζει για να βρει τον συνολικό μέσο όρο για όλες τις αναζητήσεις του χρήστη.

Στον κώδικα τώρα, για κάθε αλγόριθμο διατηρούμε σε ένα dictionary τις αναζητήσεις του χρήστη και για κάθε αναζήτηση ποια άρθρα επιστράφηκαν (tf\_results, vsm\_results, bm25\_results). Με αυτόν τον τρόπο, όταν ο χρήστης κάνει αρκετές αναζητήσεις και στην συνέχεια επιλέξει την επιλογή της αξιολόγησης επιστρέφονται οι τιμές των μετρικών για κάθε αλγόριθμο (εφόσον έχουν υπάρξει αναζητήσεις σε αυτούς).

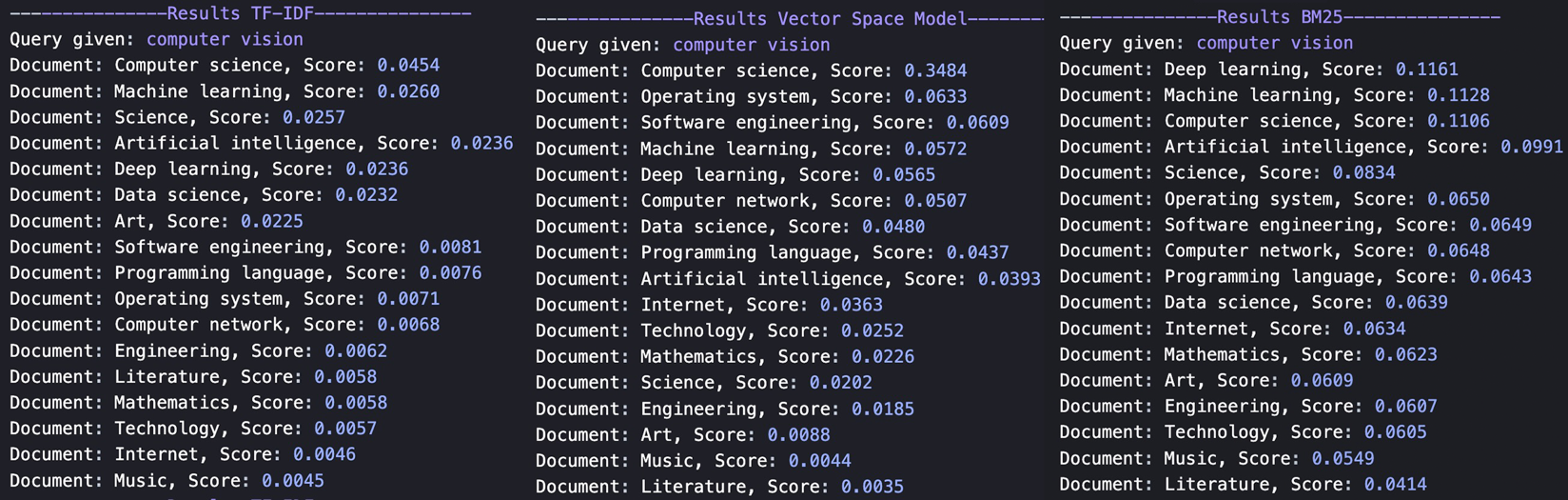
Τρέχοντας τους διάφορους αλγόριθμους για διαφορετικές αναζητήσεις βλέπουμε ότι ο TF-IDF σε πολλές αναζητήσεις δεν είναι τόσο αποδοτικός όσο το VSM ή ο BM25 καθώς δεν λαμβάνει υπόψιν την σύνδεση κάποιων λέξεων. Όταν η αναζήτηση περιλαμβάνει μία λέξη είναι πολύ καλύτερος και συμπίπτει με τα αποτελέσματα των άλλων αλγορίθμων. Ο VSM και ο BM25 συμπίπτουν στα περισσότερα αποτελέσματα που επιστρέφουν αλλά επειδή ο VSM χρησιμοποιεί διανύσματα σε μεγάλα σύνολα δεδομένων δεν είναι τόσο αξιόπιστος όσο το BM25. Συνεπώς, η απόδοση στην μηχανή αναζήτησης λαμβάνοντας υπόψιν και τις μετρικές είναι καλύτερη όταν χρησιμοποιούμε τον αλγόριθμο BM25

Ενδεικτικές Δοκιμές

* digital writing



* computer vision



* machine learning

