
Ανάκτηση Πληροφορίας

Εργασία 2025-2026

Αντιβάρχης Αντώνιος, ΑΜ: 22390289

Μαρκαντωνάτος Γεώργιος, ΑΜ: 22390138

[Colab](#) [GitHub](#)

2026-01-10

Περιεχόμενα

1. Συλλογή Δεδομένων	3
2. Προεπεξεργασία Κειμένου	4
2.1. Tokenization	4
2.2. Stemming	4
2.3. Stop-word removal	5
2.4. Αφαίρεση ειδικών χαρακτήρων	5
3. Indexing	5
4. Μηχανή Αναζήτησης	7
4.1. Boolean Retrieval	7
4.2. Vector Space Model	8
4.2.1. Βαθμολόγηση TF-IDF	8
4.3. Πιθανοτικό Μοντέλο Okapi BM25	8

1. Συλλογή Δεδομένων

Για τη μηχανή αναζήτησης χρησιμοποιείται το Reuters-21578 Corpus dataset. Το Reuters-21578 Corpus απαρτίζεται από έγγραφα που δημοσιεύτηκαν από το ειδησεογραφικό πρακτορείο [Reuters](#) το 1987. Το dataset συντάχθηκε από τον David D. Lewis, και έγινε διαθέσιμο για ερευνητικούς σκοπούς στα τέλη του 1980.

Τα έγγραφα του dataset είναι άρθρα χρηματοοικονομικού χαρακτήρα, με το corpus να αποτελεί ένα “στιγμιότυπο” της διεθνούς οικονομικής δραστηριότητας για το 1987. Κάθε έγγραφο περιέχει διάφορα πεδία προς κατηγοριοποίηση:

- `text` το κείμενο του άρθρου
- `topics` τα θέματα με τα οποία σχετίζεται το άρθρο
- `places` γεωγραφικές τοποθεσίες που αναφέρονται στο άρθρο
- `people` άτομα που αναφέρονται στο άρθρο
- `orgs` οργανισμοί που αναφέρονται στο άρθρο
- `exchanges` χρηματιστήρια που αναφέρονται στο άρθρο
- `date` η ημερομηνία δημοσίευσης του άρθρου
- `title` η επικεφαλίδα του άρθρου

Το Reuters-21578 Corpus dataset έχει διάφορες εκδοχές, για τις οποίες η κύρια διαφοροποίηση είναι το υποσύνολο των δεδομένων που επιλέγεται. Το de facto standard πλέον είναι η κατάτμηση ModApte, η οποία περιλαμβάνει έγγραφα με τουλάχιστον ένα θέμα στο πεδίο `topics`.

Το Reuters-21578 dataset έχει καθιερωθεί ως ένα από τα δημοφιλέστερα τεστ βαθμολόγησης επιδόσεων στο πλαίσιο έρευνας κατηγοριοποίησης κειμένου. Έτσι, αποτελεί μια κλασσική επιλογή dataset για τη μηχανή αναζήτησης μας.

2. Προεπεξεργασία Κειμένου

Ως προεπεξεργασία του κειμένου έγιναν οι εξής εργασίες:

- Tokenization
- Stemming
- Stop-word removal
- Αφαίρεση ειδικών χαρακτήρων

2.1. Tokenization

Tokenization είναι μία βασική διαδικασία στην επεξεργασία φυσικής γλώσσας κατά την οποία, κείμενο κατακερματίζεται σε, πιο διαχειρίσιμα για μηχανές, tokens–λεκτικές μονάδες. Ανάλογα με τον τύπο του tokenization, τα tokens μπορεί να είναι μεμονωμένοι χαρακτήρες, λέξεις ή και φράσεις. Για παράδειγμα, κάνοντας tokenization λέξεων, η φράση "Quis ut Deus?" θα γίνει ["Quis", "ut", "Deus"]. Καθώς το Reuters-21578 Corpus dataset αποτελείται από άρθρα γραμμένα στα Αγγλικά, μία γλώσσα με ξεκάθαρο διαχωρισμό των λέξεων, το tokenization σε επίπεδο λέξεων είναι κατάλληλο από τη στιγμή που δεν απαιτούμε μεγάλη λεπτομέρεια κατά την ανάλυση (όπως θα χρειαζόταν π.χ. για ορθογραφικό έλεγχο).

2.2. Stemming

Stemming είναι η διαδικασία του να μειώνουμε μία λέξη στη “ρίζα” της, δηλαδή να αφαιρούμε τυχόν προθέματα και παραθέματα. Για παράδειγμα, κάνοντας stemming για τη λέξη `most` θα έχουμε:

- `"mostly"` → `"most"`
- `"utmost"` → `"most"`
- `"foremost"` → `"most"` etc.

Αυτή η “κανονικοποίηση” κάνει τη λεκτική ανάλυση πιο αποδοτική, όμως υπάρχουν πιθανά μειονεκτήματα όπως η απώλεια ακρίβειας ή να γίνει το κείμενο πιο δυσανάγνωστο.

Υπάρχουν διάφοροι αλγόριθμοι stemming. Στη μηχανή αναζήτησης χρησιμοποιείται ο [Stemmer του Porter](#) καθώς είναι ένας από τους δημοφιλέστερους αλγορίθμους stemming, είναι γρήγορος και παρέχεται από την βιβλιοθήκη nltk.

2.3. Stop-word removal

Stop-word removal είναι η διαδικασία αφαίρεσης λέξεων χαμηλής “σημασιολογικής αξίας” από το κείμενο. Τέτοιες λέξεις είναι άρθρα, προθέσεις, αντωνυμίες etc. Για παράδειγμα, η φράση “One ring to rule them all” μετά από stop-word removal θα γίνει “ring rule”. Το όφελος είναι η βελτίωση της ακρίβειας και της απόδοσης εφαρμογών επεξεργασίας φυσικής γλώσσας, ειδικά όταν ο όγκος του κειμένου προς επεξεργασία είναι μεγάλος. Για τη μηχανή αναζήτησης, γίνεται αφαίρεση των stop-words που εμπεριέχονται στο σύνολο των αγγλικών stop-words που παρέχεται από τη βιβλιοθήκη nltk.

2.4. Αφαίρεση ειδικών χαρακτήρων

Η αφαίρεση ειδικών χαρακτήρων είναι ακριβώς αυτό που λέει το όνομα. Ομολογουμένως, ο όρος “ειδικοί χαρακτήρες” δεν είναι ιδιαίτερα συγκεκριμένος. Μπορεί να σημαίνει μόνο το σύνολο ASCII, μόνο γράμματα (π.χ. και το ελληνικό αλφάβητο που απαιτεί κωδικοποίηση UTF-8) etc. Για τη μηχανή αναζήτησης, “ειδικούς χαρακτήρες” θεωρούμε “μη αλφαριθμητικά”, κοινώς οποιονδήποτε χαρακτήρα κάνει τη μέθοδο `isalnum()` να επιστρέψει `False` για το εκάστοτε string. Η αφαίρεση ειδικών χαρακτήρων γίνεται κυρίως για ευκολία. Προφανώς δε μπορούμε να ελέγξουμε το κείμενο του Reuters-21578 Corpus “χειροκίνητα”, οπότε αντί να υλοποιήσουμε κώδικα για τη διαχείριση των ειδικών χαρακτήρων, απλά τους αφαιρούμε. Μειώνουμε έτσι και φόρτο για το πρόγραμμά μας.

3. Indexing

Για το ευρετήριο χρησιμοποιούμε τη δομή δεδομένων inverted index. Εντός της δομής αυτής, το ευρετήριο οργανώνεται βάσει όρων, και κάθε όρος “δείχνει” σε μία λίστα εγγράφων που περιέχουν τον όρο αυτό. Ουσιαστικά, το inverted index μοιάζει με [hash table](#) από την άποψη πως έχουμε συνδυασμό “κλειδιού - στοιχείου”, όπου το κλειδί είναι ο όρος και το στοιχείο είναι λίστα από έγγραφα.

Η υλοποίηση inverted index για τη μηχανή αναζήτησης βασίζεται στο `dict()` container της python, που αποθηκεύει πληροφορία με το προαναφερθέν μοτίβο “κλειδιού - στοιχείου”. Παράδειγμα dictionary:

```
car_dict = {
    "model": "R8",
    "make": "Audi",
    "year": 2006
}
```

Κάθε έγγραφο είναι ένα dictionary που περιέχει ένα *id* και μία λίστα από όρους. Συγκεκριμένα: `documents: Dict[str, List[str]]`

Έτσι, έχουμε την ακόλουθη επανάληψη για να επεξεργαζόμαστε τα στοιχεία κάθε dictionary:

```
for doc_id, terms in documents.items():
```

- Καταγράφουμε πόσους όρους έχει το έγγραφο (και κατ' επέκταση τους συνολικούς όρους του dataset):

```
doc_length = len(terms)
self.doc_lengths[doc_id] = doc_length
total_length += doc_length
```

- Καταγράφουμε τη συχνότητα των όρων:

```
term_counts = Counter(terms)
self.doc_terms[doc_id] = term_counts
```

- Τέλος, κατασκευάζουμε το ευρετήριο, καταχωρούμε δηλαδή σε λίστα τη θέση του εκάστοτε όρου και το αντίστοιχο *id* του εγγράφου:

```
for pos, term in enumerate(terms):
    self.index[term].append((doc_id, pos))
```

Η μέθοδος `enumerate()` αναθέτει δείκτες στα στοιχεία της δοθείσας λίστας για εύκολη προσπέλαση. Παράδειγμα:

```
s = ["foo", "bar", "baz"]
print(list(enumerate(s)))
```

Output: `[(0, 'foo'), (1, 'bar'), (2, 'baz')]`

Το inverted index class έχει getter methods για να επιστρέφουμε τα εξής:

- Έγγραφα που περιέχουν συγκεκριμένο όρο

```
def get_docs_containing(self, term: str) -> Set[str]:  
    return set(doc_id for doc_id, _ in self.index.get(term, []))
```

- Τη συχνότητα ενός όρου

```
def get_term_frequency(self, term: str, doc_id: str) -> int:  
    return self.doc_terms[doc_id].get(term, 0)
```

- Τη συχνότητα εγγράφων για έναν όρο, δηλαδή σε πόσα έγγραφα εμπεριέχεται ο όρος αυτός

```
def get_doc_frequency(self, term: str) -> int:  
    return len(set(doc_id for doc_id, _ in self.index.get(term, [])))
```

4. Μηχανή Αναζήτησης

Η υλοποίηση της μηχανής αναζήτησης κάνει χρήση των τριών προτεινόμενων μοντέλων της εκφώνησης:

- Boolean Retrieval
- Vector Space Model
- Okapi BM25

4.1. Boolean Retrieval

Αυτό το μοντέλο ανάκτησης βασίζεται στις γνωστές λογικές πράξεις Boole.

Στη μηχανή αναζήτησης υλοποιούνται οι εξής:

- **AND** – επιστρέφονται έγγραφα που περιέχουν **όλους** τους όρους από το query του χρήστη
- **OR** – επιστρέφονται έγγραφα που περιέχουν **τουλάχιστον έναν** από τους όρους από το query του χρήστη
- **NOT** – επιστρέφονται έγγραφα που **δεν** περιέχουν όρους από το query του χρήστη

Το μοντέλο Boolean Retrieval λειτουργεί ταιριάζοντας ακριβώς τα έγγραφα βάσει του query του χρήστη και της λογικής πράξης που επιλέχθηκε.

Συνεπώς, δεν υπάρχει νόημα κατάταξης των αποτελεσμάτων, το έγγραφο είτε επιστρέφεται είτε όχι.

4.2. Vector Space Model

Στο αυτό το μοντέλο, τα έγγραφα και τα queries αναπαριστώνται ως διανύσματα σε πολυδιάστατο χώρο. Κάθε διάσταση αντιστοιχεί σε συγκεκριμένο όρο από το dataset.

- Κάθε έγγραφο / query γίνεται N-διάστατο διάνυσμα, όπου N είναι ο αριθμός ξεχωριστών όρων στο dataset.
- Κάθε διανυσματική συνιστώσα αναπαριστά το βάρος (σημασία) του συγκεκριμένου όρου εντός του εγγράφου / query.

4.2.1. Βαθμολόγηση TF-IDF

Η φόρμουλα TF-IDF (Term Frequency-Inverse Document Frequency) χρησιμοποιείται για τον υπολογισμό του βάρους κάθε όρου εντός ενός εγγράφου.

- Term Frequency (TF) – πόσο συχνά εμφανίζεται ένας όρος στο έγγραφο:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- Cosine Similarity

Προκειμένου να υπολογίσουμε την ομοιότητα μεταξύ ενός διανύσματος query και ενός διανύσματος εγγράφου, χρησιμοποιούμε τον εξής τύπο:

$$\cos(a, b) = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2 \sum_{i=1}^n b_i^2}}$$

Τα βάρη διακυμαίνονται από 0 έως 1, με την ομοιότητα να αυξάνεται όσο το βάρος πλησιάζει το 1.

4.3. Πιθανοτικό Μοντέλο Okapi BM25

Το μοντέλο Okapi BM25 (Best Matching 25) είναι μία βελτιωμένη εκδοχή του TF-IDF. Λειτουργεί υπολογίζοντας την ομοιότητα μεταξύ ενός query q και ενός εγγράφου d , χρησιμοποιώντας Term Frequency (TF), Inverse Document Frequency (IDF) και Document Length Normalization (κανονικοποιεί την ομοιότητα για μεγαλύτερα έγγραφα έτσι ώστε να μη συσσωρεύονται στη κορυφή της κατάταξης).

- Κανονικοποίηση Term Frequency (TF)

Το Term Frequency κανονικοποιείται με την ακόλουθη φόρμουλα, με αποτέλεσμα, από ένα σημείο και μετά, οι επιπρόσθετες εμφανίσεις ενός όρου να συνεισφέρουν όλο και λιγότερο στην ομοιότητα του εγγράφου:

$$TF_{t,d} = \frac{freq(t, d)}{freq(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$