
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

# *Ανάκτηση Πληροφορίας - Εργασία Εργαστηρίου*
# *Χειμερινό Εξάμηνο 2024-2025*

---

## *Τεκμηρίωση Εργασίας - Εργαστήριο*

### *Δημιουργία μηχανής αναζήτησης*

### *Text Processing, Query Processing, Ranking, Evaluation*

***Επιμέλεια:***

- *Γουβιανάκης Ιωάννης - 21390044*
- *Κοντούλης Δημήτριος - 21390095*

Github Link: https://github.com/DimiKont/IR_Project

---

## Βήμα 1. Συλλογή Δεδομένων

Όσον αφορά την συλλογή των δεδομένων, αποφασίσαμε να βρούμε έτοιμα έγγραφα στο διαδίκτυο. Συγκεκριμένα, ψάξαμε στην ιστοσελίδα του **Kaggle** για έγγραφα τα οποία περιέχουν πληροφορίες σχετικά με διάφορους ανθρώπους και καταλήξαμε στο εξής: **People Wiki Data.** Κατεβάσαμε το .csv αρχείο του οποίου τα περιεχόμενα είχαν την μορφή **URI, name, text.** Τα πεδία αυτά περιγράφονται ως εξής:

- **URI:** Ο σύνδεσμος στον οποίο βρίσκονται περαιτέρω πληροφορίες για το εκάστοτε άτομο.
- **name:** Το ονοματεπώνυμο του ατόμου
- **text:** μία περιγραφή σχετικά με το άτομο, όπως π.χ ημερομηνία/τόπος γέννησης κ.λπ.

Είναι σημαντικό σε αυτό το σημείο να γίνει η αναφορά μερικών προβλημάτων που προέκυψαν έπειτα από την λήψη και πριν από την επεξεργασία των δεδομένων που αναφέρθηκαν παραπάνω. Αφού κάναμε λήψη του αρχείου, ξεκινήσαμε την ανάπτυξη του προγράμματος. Όσο πραγματοποιούσαμε ελέγχους στο πρόγραμμα, κοιτώντας τα αποτελέσματα παρατηρήσαμε ότι υπάρχουν μερικά **URI** των τα οποία περιέχουν περίεργους χαρακτήρες. Έπειτα από περαιτέρω μελέτη, διαπιστώσαμε ότι στους συνδέσμους αυτούς των **URI** υπήρχαν αριθμοι δεκαεξαδικής μορφής. Οι αριθμοί αυτοί ήταν ειδικοί χαρακτήρες οι οποίοι εμφανίζονταν σε ονόματα, όπως για παράδειγμα ο Ισπανικός χαρακτήρας **á.** Για την επεξεργασία αυτών έγινε ανάπτυξη του προγράμματος **clear_impure.py,** το οποίο είναι ένα απλό Python script ικανό να φιλτράρει τους χαρακτήρες αυτούς. Το περιεχόμενο του βρίσκεται εντός του **GitHub Repository.**

## Βήμα 2. Προεπεξεργασία κειμένου (Text Preprocessing)

Ως προεπεξεργασία κειμένου, αναφερόμαστε στην εφαρμογή μεθόδων όπως **tokenization**, **stemming, αφαίρεση ειδικών χαρακτήρων** και άλλων, έτσι ώστε σε μεταγενέστερα στάδια να μπορεί η μηχανή να εξάγει την απαραίτητη πληροφορία. Ας δούμε λοιπόν τις μεθόδους οι οποίες πραγματοποιήθηκαν για να γίνει επιτυχής προεπεξεργασία του κειμένου.

- **Tokenization:** διαδικασία κατά την οποία το δοσμένο κείμενο χωρίζεται σε ξεχωριστούς μικρότερους όρους, όπως λέξεις.
- **Special Character Removal:** Οι ειδικοί χαρακτήρες δεν συνεισφέρουν ιδιαίτερα στο νόημα του κειμένου, επομένως μπορούν να αφαιρεθούν. Ως ειδικούς χαρακτήρες εννοούμε το θαυμαστικό (!), το ερωτηματικό (;) και άλλα.
- **Stop-word Removal:** Λέξεις όπως το “is”, “the”, “are”, “a”, αποτελούν ένα ιδιαίτερα μεγάλο κομμάτι ενός κειμένου, και για τους σκοπούς της λειτουργία της μηχανής αναζήτησης δεν έχουν μεγάλη σημασία.
- **Stemming/Lemmatization:** Ένα από τα αποτελέσματα εφαρμογής ενός τέτοιου αλγορίθμου επί ενός συνόλου λέξεων, ουσιαστικά αφαιρεί το πρόθημα (prefix) και το επίθημα (suffix) των λέξεων αυτών. Με αυτό τον τρόπο, διαφορετικές μορφές της ίδιας λέξης θεωρούνται το ίδιο. Όπως για παράδειγμα οι λέξεις **creative, creating, created,** όλες προέρχονται από την ίδια ρίζα **create.**

## Βήμα 3. Ευρετήριο (Indexing)

**A)** Στο υποερώτημα αυτό ζητείται η δημιουργία ενός ανεστραμμένου ευρετηρίου (inverted index) ώστε να γίνει αντιστοίχιση των όρων στα έγγραφα. Κάθε έγγραφο, στην δική μας περίπτωση αναφερόμαστε σε κάθε **URI,** περιέχει ένα σώμα κειμένου. Αυτό το σώμα κειμένου, το έχουμε χωρίσει σε ξεχωριστές λέξεις με την υλοποίηση του **Βήματος 2,** στο οποίο πραγματοποιήθηκε προεπεξεργασία του κάθε κειμένου. Σκοπός μας λοιπόν τώρα είναι βρούμε σε ποιά έγγραφα εμφανίζεται το σύνολο των λέξεων που έχει προκύψει, καθώς και το πλήθος αυτών στο εκάστοτε έγγραφο. Για να επιτευχθεί αυτό, πρέπει να γίνει προσπέλαση στο σύνολο των χωρισμένων λέξεων κάθε εγγράφου, και να προσμετρηθεί το πλήθος (count) όλων των λέξεων σε κάθε έγγραφο. Με την εκτέλεση των προαναφερθέντων θα δημιουργηθεί ένα dataFrame, στου οποίου τις στήλες θα υπάρχουν όλες οι λέξεις που βρίσκονται σε όλα τα έγγραφα και στις σειρές όλα τα έγγραφα του dataSet[1]. Παρακάτω φαίνεται ένα screenshot στο οποίο φαίνεται η μορφή του dataset όπως περιγράφηκε παραπάνω.

Out[42]:

| | aaron | lacrate | american | music | producer | recording | artist | dj | fashion | designer | ... | investigationzuenir | janit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| http://dbpedia.org/resource/Aaron_LaCrate | 5 | 12 | 1 | 7 | 1 | 2 | 2 | 3 | 5 | 2 | ... | 0 | |
| http://dbpedia.org/resource/Abdel_Fattah_el-Sisi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Abdul_Salam_Azimi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Abdulalim_A._Shabazz | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Abu_Mustafa_al-Sheibani | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| http://dbpedia.org/resource/Yolande_Thibeault | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Yoshi_Wada | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Yoshihisa_Naruse | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Zahid_Al-Sheikh | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| http://dbpedia.org/resource/Zuenir_Ventura | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | |

1000 rows × 31294 columns

***Εικόνα 1.*** *Μορφή ανεστραμμένου ευρετηρίου (Inverted Index)*

**B)** Έπειτα, για να μπορέσουμε να χρησιμοποιήσουμε πιο ελεύθερα την ανεστραμμένη δομή δεδομένων ευρετηρίου θα εφαρμόσουμε μια δομή δεδομένων για να αποθηκεύσουμε το ευρετήριο σε μια εύκολη και χρήσιμη μορφή. Γι αυτό τον λόγο και το ευρετήριο αποθηκεύτηκε σε ένα αρχείο τύπου csv. Για την αποθήκευση του dataframe, χρειάστηκε απλώς η εκτέλεση της εντολής **.to_csv()** επί του dataframe, δηλαδή **df.to_csv(),** δίνοντας ως παράμετρο το **filepath** στο οποίο θέλουμε να αποθηκεύσουμε το αρχείο.

[1] Κανονικά η μορφή που επιστρέφεται για το dataset μετά από την εκτέλεση του κώδικα είναι στις σειρές οι λέξεις και στις στήλες τα έγγραφα, αλλά με την χρήση της ιδιότητας T (Transpose), μπορούμε να αναστρέψουμε τον πίνακα. Αυτό έγινε για λόγους ευαναγνωσίας κυρίως.

## Βήμα 4. Μηχανή αναζήτησης (Search Engine)

**A)** Στο υποερώτημα αυτό θα αξιοποιήσουμε το **ανεστραμμένο ευρετήριο (inverted index)** το οποίο υλοποιήσαμε προηγουμένως στο **Βήμα 3**, για την δημιουργία ενός module ικανού να επεξεργαστεί **ερωτήματα Boolean (Boolean Queries).** Τα ερωτήματα αυτά κατασκευάζονται από λέξεις τις οποίες θέλουμε να ψάξουμε μέσω του ανεστραμμένου ευρετηρίου, καθώς και λέξεις που αποτελούν **λογικούς τελεστές** όπως **AND, OR** και **NOT.** Οι λογικοί τελεστές εισάγονται μεταξύ μεμονωμένων λέξεων. Για παράδειγμα, ένα τέτοιο query μπορεί να είναι **musician AND jazz AND american.** Ανάλογα με το ποιοί λογικοί τελεστές εισάγονται, πραγματοποιούνται και διαφορές πράξεις μεταξύ των συνόλων των όρων. Δηλαδή αν βάλουμε **musician AND jazz**, τότε θα πρέπει να μας επιστρέψει το σύνολο των εγγράφων στα οποία βρίσκονται και οι δύο όροι μαζί. Αν για παράδειγμα υπήρχε το λογικό **ή (OR)** ενδιάμεσα, τότε θα μας επέστρεφε τα έγγραφα στα οποία εμφανίζεται είτε ο ένας είτε ο άλλος όρος. Αντίστοιχα με τον λογικό τελεστής άρνησης **NOT,** θα μας επέστρεφε τα έγγραφα στα οποία δεν υπάρχει ο όρος του οποίου προηγείται ο τελεστής. Για παράδειγμα το ερώτημα **musician NOT american,** θα είχε ως αποτέλεσμα την επιστροφή των εγγράφων στα οποία περιέχεται ο όρος **musician** αλλά όχι και ο όρος **american.** Αυτή είναι η γενικότερη λογική της λειτουργίας ενός επεξεργαστή ερωτημάτων ικανό να χειρίζεται **ερωτήματα Boolean**.

Παρακάτω παρουσιάζονται μερικές εικόνες οι οποίες περιέχουν την επεξεργασία μερικών boolean ερωτημάτων, όπως αυτών που αναφέρθηκαν προηγουμένως. Για κάθε ερώτημα, εμφανίζεται ο αριθμός των εγγράφων που επιστράφηκαν, καθώς και το αντίστοιχο **URI** σε μορφή συνδέσμου.

```
Tokenized query:  ['musician', 'AND', 'jazz']
Number of matching URIs:  18
Matching URI: http://dbpedia.org/resource/Asahito_Nanjo
Matching URI: http://dbpedia.org/resource/Bob_Havens
Matching URI: http://dbpedia.org/resource/Shamek_Farrah
Matching URI: http://dbpedia.org/resource/Jamie_Shields_(musician)
Matching URI: http://dbpedia.org/resource/Gregg_Karukas
Matching URI: http://dbpedia.org/resource/Gwyn_Jay_Allen
Matching URI: http://dbpedia.org/resource/John_Edmondson_(musician)
Matching URI: http://dbpedia.org/resource/Vanessa_Ament
Matching URI: http://dbpedia.org/resource/Attila_Pacsay
Matching URI: http://dbpedia.org/resource/Tony_Garnier_(musician)
Matching URI: http://dbpedia.org/resource/Chris_Karan
Matching URI: http://dbpedia.org/resource/Dudley_Riggs
Matching URI: http://dbpedia.org/resource/Martin_Iveson
Matching URI: http://dbpedia.org/resource/Oren_Bloedow
Matching URI: http://dbpedia.org/resource/Mikko_Innanen_(musician)
Matching URI: http://dbpedia.org/resource/Dylan_Cramer
Matching URI: http://dbpedia.org/resource/Kasia_Kowalska
Matching URI: http://dbpedia.org/resource/Sam_Stephenson_(writer)
```

```
Tokenized query:  ['musician', 'AND', 'jazz', 'AND', 'american']
Number of matching URIs:  6
Matching URI: http://dbpedia.org/resource/Bob_Havens
Matching URI: http://dbpedia.org/resource/Gwyn_Jay_Allen
Matching URI: http://dbpedia.org/resource/John_Edmondson_(musician)
Matching URI: http://dbpedia.org/resource/Vanessa_Ament
Matching URI: http://dbpedia.org/resource/Tony_Garnier_(musician)
Matching URI: http://dbpedia.org/resource/Oren_Bloedow
```

```
Tokenized query:  ['musician', 'AND', 'jazz', 'NOT', 'american']
Number of matching URIs:  12
Matching URI: http://dbpedia.org/resource/Asahito_Nanjo
Matching URI: http://dbpedia.org/resource/Shamek_Farrah
Matching URI: http://dbpedia.org/resource/Jamie_Shields_(musician)
Matching URI: http://dbpedia.org/resource/Gregg_Karukas
Matching URI: http://dbpedia.org/resource/Attila_Pacsay
Matching URI: http://dbpedia.org/resource/Chris_Karan
Matching URI: http://dbpedia.org/resource/Dudley_Riggs
Matching URI: http://dbpedia.org/resource/Martin_Iveson
Matching URI: http://dbpedia.org/resource/Mikko_Innanen_(musician)
Matching URI: http://dbpedia.org/resource/Dylan_Cramer
Matching URI: http://dbpedia.org/resource/Kasia_Kowalska
Matching URI: http://dbpedia.org/resource/Sam_Stephenson_(writer)
```

```
Tokenized query:  ['musician', 'AND', 'jazz', 'AND', 'american', 'OR', 'french']
Number of matching URIs:  58
Matching URI: http://dbpedia.org/resource/Adil_Rami
Matching URI: http://dbpedia.org/resource/Nick_Baines_(bishop)
Matching URI: http://dbpedia.org/resource/Sophie_Crumb
Matching URI: http://dbpedia.org/resource/Gianluca_Attanasio
Matching URI: http://dbpedia.org/resource/Patrice_Yengo
Matching URI: http://dbpedia.org/resource/Charles_Lafortune
Matching URI: http://dbpedia.org/resource/Robert_Petit
Matching URI: http://dbpedia.org/resource/Adel_Sellimi
Matching URI: http://dbpedia.org/resource/Vasco_Uva
Matching URI: http://dbpedia.org/resource/Joseph_Nation
Matching URI: http://dbpedia.org/resource/Mikko_Innanen_(musician)
Matching URI: http://dbpedia.org/resource/Trevor_Ferguson
Matching URI: http://dbpedia.org/resource/Elaine_Princi
Matching URI: http://dbpedia.org/resource/Isaac_Chueke
Matching URI: http://dbpedia.org/resource/Guy_Sorman
Matching URI: http://dbpedia.org/resource/Ventimiglia_Giovanni
Matching URI: http://dbpedia.org/resource/Jimmy_Lewis_(surfer)
Matching URI: http://dbpedia.org/resource/Oren_Bloedow
Matching URI: http://dbpedia.org/resource/Lucien_Bourjeily
Matching URI: http://dbpedia.org/resource/Tana_Umaga
Matching URI: http://dbpedia.org/resource/Bob_Flowerdew
Matching URI: http://dbpedia.org/resource/Anthony_Cekada
Matching URI: http://dbpedia.org/resource/Bob_Havens
...
Matching URI: http://dbpedia.org/resource/Ahmed_Najib_Chebbi
Matching URI: http://dbpedia.org/resource/Roger_Godement
Matching URI: http://dbpedia.org/resource/Aled_de_Malmanche
Matching URI: http://dbpedia.org/resource/James_Grieve_(translator/author)
```

**Β)** Σε αυτό το κομμάτι, ζητείται να κατατάξουμε τα αποτελέσματα με διάφορους διαφορετικούς αλγόριθμους κατάταξης. Για αρχικό αλγόριθμό θα χρησιμοποιήσουμε το module που φτιάχτηκε στο **υποερώτημα Α** αυτού του βήματος και θα προσθέσουμε την δυνατότητα ο αλγόριθμος να μετράει το πόσες φορές εμφανίζονται μέρη του ζητούμενου σε κάθε έγγραφο το οποίο υπάρχει στο αποτέλεσμα του module. Έπειτα θα χρησιμοποιήσουμε άλλους 3 αλγορίθμους για να κάνουμε κατάταξη προσθέτοντας και την επιλογή στον χρήστη να επιλέξει ποιον από τους 3 θέλει να χρησιμοποιήσει. Αρχικά χρησιμοποιούμε τον αλγόριθμο boolean retrieval ο οποίος έχει την δυνατότητα να πάρει πιο σύνθετα ζητούμενα. Ο αλγόριθμος vector space model ο οποίος λειτουργεί με τις σχέσεις των ζητούμενων και των δεδομένων και τέλος ο αλγόριθμος probabilistic retrieval model το οποίο λειτουργεί με την πιθανότητα το ζητούμενο να είναι στα δεδομένα.

## Βήμα 5. Αξιολόγηση Συστήματος

Σε αυτό το κομμάτι , ζητείται να αξιολογήσουμε την μηχανή αναζήτησης. Για να αξιολογήσουμε την μηχανή θα χρησιμοποιήσουμε συγκεκριμένους μαθηματικούς τύπους όπως την ακρίβεια, την ανάκληση και το F1-score. Αρχικά , για να μπορέσουμε να χρησιμοποιήσουμε τους μαθηματικούς τύπους θα χρειαστεί να δημιουργήσουμε ένα σύνολο δεδομένων τα οποία θα χρησιμοποιηθούν ως τα απόλυτα σωστά και τα ζητούμενα της κάθε αναζήτησης. Χρησιμοποιώντας αυτά τα δεδομένα καθώς και τα αποτελέσματα της μηχανής προς αξιολόγηση , μπορούμε να καταλάβουμε πόσο ικανή είναι η μηχανή στην σωστή αναζήτηση.

Χρησιμοποιώντας μελέτες περιπτώσεων από ερωτήματα χρηστών μπορέσαμε να βρούμε την **ακρίβεια (precision)**, ανάκληση **(recall)** και F1- score για κάθε μία από τις μεθόδους κατάταξης των αποτελεσμάτων **(ranking)**.

```
Evaluation Metrics:
Precision: 0.186
Recall: 1.000
F1-Score: 0.313
probabilistic_retrieval
musician AND jazz
```

```
Evaluation Metrics:
Precision: 0.186
Recall: 1.000
F1-Score: 0.313
vsm_retrieval
musician AND jazz
```

```
Evaluation Metrics:
Precision: 1.000
Recall: 1.000
F1-Score: 1.000
boolean_retrieval
musician AND jazz
```

```
Evaluation Metrics:
Precision: 0.752
Recall: 1.000
F1-Score: 0.859
probabilistic_retrieval
musician NOT pop
```

```
Evaluation Metrics:
Precision: 0.752
Recall: 1.000
F1-Score: 0.859
vsm_retrieval
musician NOT pop
```

```
Evaluation Metrics:
Precision: 1.000
Recall: 1.000
F1-Score: 1.000
boolean_retrieval
musician NOT pop
```

```
Evaluation Metrics:
Precision: 1.000
Recall: 1.000
F1-Score: 1.000
probabilistic_retrieval
soldier OR worker
```

```
Evaluation Metrics:
Precision: 1.000
Recall: 1.000
F1-Score: 1.000
vsm_retrieval
soldier OR worker
```

```
Evaluation Metrics:
Precision: 1.000
Recall: 1.000
F1-Score: 1.000
boolean_retrieval
soldier OR worker
```

Βλέποντας από τα παραπάνω παραδείγματα, βλέπουμε σε ποιες περιπτώσεις λειτουργούν σωστοί οι διαφορετικοί μέθοδοι και σε ποιες παρουσιάζουν προβλήματα. Επιπλέον, παρατηρήθηκε ότι ανάλογα τον τρόπο αναζήτησης, δημιουργούνται και κάποια αδύνατα σημεία.

**Boolean:** Σε αυτή την μέθοδο, έχουμε απόλυτα σωστή αναζήτηση χωρίς κάποιο λάθος με την διαφορά όμως ότι δεν χρησιμοποιείται κάποιος τρόπος κατάταξης ώστε μέσα στα αποτελέσματα που βρέθηκαν να υπάρχει δεδομένο που να είναι πιο σχετικό από κάποιο άλλο.
**VSM:** Σε αυτή την μέθοδο, έχουμε αναζήτηση κατά την σχετικότητα ανάμεσα στα δεδομένα που έχουμε και στα ζητούμενα που παίρνουμε από τον χρήστη. Η μέθοδος αυτή έχει την δυνατότητα να μας δώσει κατάταξη στα αποτελέσματα ώστε να βοηθήσει τους χρήστες. Το αδύνατο σημείο του VSM είναι ότι χρειάζεται περισσότερη υπολογιστή δύναμη και άρα περισσότερο χρόνο για να πραγματοποιηθεί καθώς και τα αποτελέσματα δεν είναι απολύτως σωστά.

**Probabilistic retrieval:** Σε αυτή την μέθοδο, έχουμε αναζήτηση κατά την πιθανότητα να υπάρχει το ζητούμενο μέσα σε κάθε δεδομένο πάνω στο οποίο ψάχνει η μηχανή αναζήτησης. Παρομοίως με την VSM μέθοδο έχει την δυνατότητα να μας δώσει κατάταξη στα αποτελέσματα και ανάλογα με τα ζητούμενα τα αποτελέσματα δεν είναι απολύτως σωστά με την διαφορά ότι βγάζει αποτελέσματα σε λιγότερο χρόνο.