



Σχολή Θετικών Επιστημών και Τεχνολογίας
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

**Σύγκριση Αλγορίθμων Μη Επιβλεπόμενης Μάθησης για
Εξαγωγή Λέξεων-Κλειδιών και Ανάπτυξη Πιλοτικής
Εφαρμογής Σύστασης Περιοδικών**

Βασίλειος Βάγιας

Επιβλέπων καθηγητής: Γεώργιος Μαυρομμάτης

Πάτρα, Ιούλιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



Σύγκριση Αλγορίθμων Μη Επιβλεπόμενης Μάθησης για Εξαγωγή
Λέξεων-Κλειδιών και Ανάπτυξη Πιλοτικής Εφαρμογής Σύστασης
Περιοδικών

Βασίλειος Βάγιας

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:
Γεώργιος Μαυρομμάτης
Σ.Ε.Π. ΣΘΕΤ ΕΑΠ

Συν-Επιβλέπων Καθηγητής:
Ανδρέας Καναβός
Σ.Ε.Π. ΣΘΕΤ ΕΑΠ

Πάτρα, Ιούλιος 2026

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Επιβλέποντα Καθηγητή μου Γεώργιο Μαυρομμάτη για τον χρόνο που αφιέρωσε, την καθοδήγηση και τις υποδείξεις του για την πραγματοποίηση της παρούσας εργασίας. Επίσης, τον Συν-επιβλέποντα καθηγητή Ανδρέα Καναβό για τις παρατηρήσεις και τα εύστοχα σχόλιά του για την βελτίωση της εργασίας. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την υπομονή και την αμέριστη συμπαράσταση καθ' όλη τη διάρκεια του Μεταπτυχιακού Προγράμματος Σπουδών.

Περίληψη

Στην παρούσα διπλωματική εργασία συγκρίθηκαν και αξιολογήθηκαν αλγόριθμοι μη επιβλεπόμενης μάθησης για την εξαγωγή λέξεων-φράσεων κλειδιών από επιστημονικά άρθρα γραμμένα στην αγγλική γλώσσα με σκοπό να χρησιμοποιηθούν σε εφαρμογές ανάκτησης πληροφορίας. Οι αλγόριθμοι που εξετάστηκαν ήταν οι TF-IDF, TEXTRANK, RAKE, YAKE και KEYBERT. Επίσης εξετάστηκε και μια παραλλαγή του TF-IDF, ο αλγόριθμος TF-ISF.

Οι αλγόριθμοι εφαρμόστηκαν με stemming και χωρίς stemming και για αριθμό n-grams. Μέσω της χρήσης συλλογών κατάλληλων δεδομένων (δημοσιευμένα άρθρα) οι παραπάνω αλγόριθμοι αξιολογήθηκαν αρχικά στο να κατατάσσουν υψηλότερα τις λέξεις κλειδιά του συγγραφέα ενός άρθρου με τη χρήση των μετρικών $P@k$, $R@k$, MAP, MRR και nDCG και ακολούθως στο να προβλέπουν το περιοδικό και το επιστημονικό πεδίο του περιοδικού που ένα συγκεκριμένο άρθρο είναι δημοσιευμένο, με τη χρήση της μετρικής MRR.

Ως προς την πρόβλεψη των λέξεων κλειδιών του συγγραφέα την καλύτερη επίδοση την είχε ο TEXTRANK, αν και όλοι οι αλγόριθμοι κατάφεραν να κατατάξουν το 70% και το 100% των λέξεων φράσεων κλειδιών του συγγραφέα σε θέσεις που αντιστοιχούν στο 10% και 60% των πιο αντιπροσωπευτικών όρων αντίστοιχα. Το stemming ή ο αριθμός n-grams δεν επηρέασε τα αποτελέσματα των μετρικών.

Ως προς την πρόβλεψη του επιστημονικού περιοδικού που είναι δημοσιευμένο ένα άρθρο ο KEYBERT είχε μια υπεροχή σε σχέση με τους υπόλοιπους αλγορίθμους καθώς προέβλεψε το περιοδικό στην 9^η θέση κατά μέσο όρο σε αντίθεση με τους υπόλοιπους αλγορίθμους που κατέτασσαν το περιοδικό μεταξύ 10^{ης} και 37^{ης} θέσης. Ωστόσο ο KEYBERT ήταν ο πλέον αργός αλγόριθμος. Αναφορικά με τον αλγόριθμο TF-ISF έδωσε παρόμοια αποτελέσματα με τον TF-IDF αλλά σε πολύ χαμηλότερο χρόνο εκτέλεσης.

Ακολούθως έλαβε χώρα ανάπτυξη μιας εφαρμογής συστάσεων περιοδικών προς δημοσίευση η οποία παραμετροποιήθηκε έτσι ώστε να λειτουργεί με οποιοδήποτε από τους παραπάνω αλγορίθμους με stemming ή χωρίς stemming και για οποιοδήποτε αριθμό n-grams. Η εφαρμογή αναπτύχθηκε σε γλώσσα προγραμματισμού Python στο framework streamlit.

Λέξεις – Κλειδιά

Εξαγωγή λέξεων φράσεων κλειδιών, Ανάκτηση πληροφορίας, Επεξεργασία φυσικής γλώσσας, TEXTRANK, KEYBERT, Μη επιβλεπόμενη μάθηση

Comparative assessment of unsupervised keyword extraction algorithms and development of a pilot application for recommending publication journals

Vasileios Vagias

Abstract

In the present thesis, unsupervised algorithms for extracting keyphrases from scientific articles written in English were compared and evaluated, with the aim of applying them to information retrieval systems.

The algorithms examined include TF-IDF, TEXTRANK, RAKE, YAKE, and KEYBERT. In addition, a variation of TF-IDF, namely TF-ISF, was also investigated. The algorithms were applied both with and without stemming, and for different n-gram configurations. Using appropriate datasets consisting of published scientific articles, the algorithms were initially evaluated based on their ability to rank the authors' keywords of each article higher. The evaluation metrics used, were $P@k$, $R@k$, MAP , MRR and $nDCG$. Subsequently, the performance of the algorithms was assessed in predicting the journal and the scientific field in which a given article was published, by using the MRR metric.

Regarding the prediction of authors' keywords, TEXTRANK achieved the best performance. However, all algorithms managed to rank approximately 70% and 100% of the authors' keyphrases within the top 10% and 60% of the most representative terms, respectively. The use of stemming and the choice of n-gram size did not significantly affect the evaluation metrics.

In terms of predicting the journal in which an article was published, KEYBERT outperformed the other algorithms, ranking the correct journal at an average position of 9th, compared to the other algorithms, which ranked it between the 10th and 37th positions. All algorithms however, managed to predict the field of science of the suggested journal within the top two ranking positions. KEYBERT was the slowest algorithm in terms of execution time. As for TF-ISF, it produced results comparable to TF-IDF but with significantly lower computational cost.

Finally, a journal recommendation system was developed in Python Streamlit framework, which was parameterized to operate with any of the aforementioned algorithms, with or without stemming, and for any n-gram configuration.

Keywords

Keyphrase Extraction, Information Retrieval, Natural Language Processing, TEXTRANK, KEYBERT, Unsupervised Learning.

Περιεχόμενα

Περίληψη.....	v
Abstract	vii
Περιεχόμενα	ix
Κατάλογος Σχημάτων	xi
Κατάλογος Πινάκων	xiii
Συντομογραφίες & Ακρωνύμια.....	xiv
1 Εισαγωγή.....	1
1.1 Στόχος της εργασίας.....	1
1.2 Δομή της εργασίας	1
2 Θεωρητικό πλαίσιο	3
2.1 Ορισμοί	3
2.2 Επεξεργασία Φυσικής Γλώσσας (NLP)	3
2.2.1 Γλωσσική Προ-επεξεργασία	3
2.2.2 Διακριτοποίηση λεκτικών (tokenization).....	4
2.2.3 Τυποποίηση-Καθαρισμός (Standardization and cleansing)	5
2.2.4 Αφαίρεση «κοινότυπων» λέξεων (Stop words removal)	6
2.2.5 Εύρεση Ρίζας ή Λήμματος (Stem and Lemmatize).....	7
2.3 Εξαγωγή Λέξεων/Φράσεων Κλειδιών (Keywords/Keyphrases) από κείμενα.....	8
2.3.1 Αλγόριθμος TF-IDF	9
2.3.2 Αλγόριθμος RAKE.....	10
2.3.3 Αλγόριθμος Yet Another Keyword Extraction (YAKE)	13
2.3.4 Μέθοδοι βασισμένες σε γράφους. Ο αλγόριθμος TEXTRANK.....	15
2.3.5 Ο αλγόριθμος KEYBERT	18
2.4 Αναπαράσταση εγγράφων.....	19
2.4.1 Αναπαράσταση εγγράφου – «σάκος λέξεων» (Bag of Words).....	19
2.4.2 Αναπαράσταση μιας συλλογής εγγράφων	19
2.5 Ανάκτηση πληροφορίας – Information Retrieval	21
2.5.1 Γενικό Θεωρητικό Μοντέλο	21
2.5.2 Boolean Retrieval (Boolean ΑΠ)	21
2.5.3 ΑΠ διανυσματικού χώρου (Vector Space Model)	23
2.5.4 Πιθανοθεωρητική ΑΠ (Probabilistic Model).....	24
2.6 Αξιολόγηση αποτελεσματικότητας συστήματος ΑΠ.....	25
2.6.1 Γενικά – Μήτρα σύγχυσης (confusion matrix)	25
2.6.2 Ανάκτηση χωρίς κατάταξη.....	26
2.6.3 Ανάκτηση με κατάταξη.....	27
3 Υλοποίηση αλγορίθμων για την εξαγωγή λέξεων φράσεων κλειδιά από έγγραφα ...	31
3.1 Επιλογή αλγορίθμων	31
3.2 Υλοποίηση αλγορίθμων	32
3.2.1 Γενικά.....	32
3.2.2 Η κλάση Keyword.....	33
3.2.3 Η μητρική κλάση Algorithm.....	33
3.2.4 Η κλάση myRake (Algorithm).....	36
3.2.5 Η κλάση myYake (Algorithm).....	38
3.2.6 Η κλάση myTextRank (Algorithm)	39
3.2.7 Η κλάση myKeyBert (Algorithm).....	42
3.2.8 Η κλάση InvertSentFreq (Algorithm)	42

3.2.9	Διαγράμματα κλάσεων UML	44
4	Συγκριτική αξιολόγηση αλγορίθμων	46
4.1	Συλλογές δεδομένων αξιολόγησης (datasets)	46
4.2	Συγκριτική αξιολόγηση αλγορίθμων για πρόβλεψη λέξεων/φράσεων κλειδιά – Μετρικές σύγκρισης.....	46
4.3	Συγκριτική αξιολόγηση αλγορίθμων για πρόβλεψη λέξεων/φράσεων κλειδιά - Αποτελέσματα.....	47
4.3.1	n_grams =1	47
4.3.2	n_grams έως και 2	53
4.3.3	n_grams έως και 3	58
4.3.4	Σύγκριση ταχύτητας εκτέλεσης των αλγορίθμων	63
4.4	Συγκριτική αξιολόγηση αλγορίθμων στην ανάκτηση πληροφορίας.....	64
4.4.1	Μέθοδος σύγκρισης	64
4.4.2	Αποτελέσματα.....	65
5	Ανάπτυξη εφαρμογής συστάσεων περιοδικών	68
5.1	Γενική περιγραφή της εφαρμογής.....	68
5.2	Περιγραφή του πλαισίου ανάπτυξης (framework) της εφαρμογής.....	68
5.3	Περιγραφή της αρχιτεκτονικής της εφαρμογής	69
5.3.1	Διάγραμμα περιπτώσεων χρήσης.....	69
5.3.2	Οθόνες (Views).....	70
5.3.3	Σχήμα Βάσης Δεδομένων – Περιγραφή σχέσεων οντοτήτων.....	78
5.3.4	Διαγράμματα κλάσεων.....	80
5.4	Υλοποίηση Οθονών (Views).....	85
5.4.1	Η αφηρημένη κλάση View.....	86
5.4.2	Αλληλοεπίδραση αντικειμένων View με την εφαρμογή.....	88
5.4.3	Η κλάση SuggestView(View).....	90
6	Συμπεράσματα – Συζήτηση	92
	Βιβλιογραφία.....	97
	ΠΑΡΑΡΤΗΜΑ: Υλοποίηση κλάσεων της παρούσας ΔΕ	100

Κατάλογος Σχημάτων

Σχήμα 2-2 Αναπαράσταση σε γράφο 5 ιστοσελίδων	16
Σχήμα 3-1 Διάγραμμα κλάσεων UML των αλγορίθμων που υλοποιήθηκαν	45
Σχήμα 4-1 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής P@k, για n_grams=1 με stemming και χωρίς stemming	49
Σχήμα 4-2 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής R@k, για n_grams=1 με stemming και χωρίς stemming	50
Σχήμα 4-3 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής F1@k, για n_grams=1 με stemming και χωρίς stemming	51
Σχήμα 4-4 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για n_grams=1 με stemming και χωρίς stemming	52
Σχήμα 4-5 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής P@k, για n_grams ≤ 2 με stemming και χωρίς stemming	54
Σχήμα 4-6 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής R@k, για n_grams ≤ 2 με stemming και χωρίς stemming	55
Σχήμα 4-7 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής F1@k, για n_grams ≤ 2 με stemming και χωρίς stemming	56
Σχήμα 4-8 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για n_grams ≤ 2 με stemming και χωρίς stemming	57
Σχήμα 4-9 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής P@k, για n_grams ≤ 3 με stemming και χωρίς stemming	59
Σχήμα 4-10 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής R@k, για n_grams ≤ 3 με stemming και χωρίς stemming	60
Σχήμα 4-11 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής F1@k, για n_grams ≤ 3 με stemming και χωρίς stemming	61
Σχήμα 4-12 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για n_grams ≤ 3 με stemming και χωρίς stemming	62
Σχήμα 5-1 Διάγραμμα Περιπτώσεων Χρήσης Application	69
Σχήμα 5-2 Οθόνη καλωσορίσματος	70
Σχήμα 5-3 Οθόνη εγγραφής χρήστη	71
Σχήμα 5-4 Οθόνη προτροπής του χρήστη να ανεβάσει το άρθρο του	71
Σχήμα 5-5 Οθόνη παρουσίασης στον χρήστη των σχετικών περιοδικών	72
Σχήμα 5-6 Οθόνη εισαγωγής νέου άρθρου στη βάση – Εισαγωγή περιοδικού	72
Σχήμα 5-7 Οθόνη εισαγωγής νέου άρθρου στη βάση – Έλεγχος ύπαρξης περιοδικού	73
Σχήμα 5-8 Οθόνη εισαγωγής νέου άρθρου στη βάση – Προσθήκη στοιχείων άρθρου	73
Σχήμα 5-9 Οθόνη εισαγωγής νέου άρθρου στη βάση – Επιβεβαίωση	74
Σχήμα 5-10 Οθόνη διαγραφής περιοδικού από τη βάση	74
Σχήμα 5-11 Οθόνη επικαιροποίησης περιοδικού από τη βάση	75
Σχήμα 5-12 Οθόνη επικαιροποίησης περιοδικού από τη βάση – Αλλαγή ονόματος	75
Σχήμα 5-13 Οθόνη επικαιροποίησης περιοδικού από τη βάση – Επιβεβαίωση	76
Σχήμα 5-14 Οθόνη επικαιροποίησης στοιχείων χρήστη – Εγγεγραμμένος χρήστης	76
Σχήμα 5-15 Οθόνη επικαιροποίησης στοιχείων χρήστη – Διαχειριστής	77
Σχήμα 5-16 Οθόνη δημιουργίας νέου χρήστη	77
Σχήμα 5-17 Οθόνη διαγραφής χρήστη	78
Σχήμα 5-18 Διάγραμμα Οντοτήτων Συσχετίσεων της ΒΔ	79
Σχήμα 5-19 Βασικό Διάγραμμα κλάσεων Journal_App, Session_Manager και View	84

Σχήμα 5-20 Δομή φακέλων της εφαρμογής.....	84
Σχήμα 5-21 Εκτεταμένο Διάγραμμα κλάσεων Journal_App, Session_Manager, View και SuggestView	91

Κατάλογος Πινάκων

Πίνακας 2-1 Πίνακας συνύπαρξης όρων του RAKE	12
Πίνακας 2-2 Σχηματική αναπαράσταση της μήτρας όρος (t)-έγγραφο σε μια συλλογή εγγράφων (D)	20
Πίνακας 2-3 Μήτρα σύγχυσης	26
Πίνακας 4-1 Μέσοι χρόνοι εκτέλεσης των αλγορίθμων με stemming και χωρίς stemming για $n_grams \leq 3$	63
Πίνακας 4-2 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων χωρίς stemming να προβλέπουν το περιοδικό δημοσίευσης μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις).....	66
Πίνακας 4-3 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων με stemming να προβλέπουν το περιοδικό δημοσίευσης μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις).....	66
Πίνακας 4-4 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων χωρίς stemming να προβλέπουν το επιστημονικό πεδίο μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)	67
Πίνακας 4-5 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων με stemming να προβλέπουν το επιστημονικό πεδίο μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)	67

Συντομογραφίες & Ακρωνύμια

ΔΕ: Διπλωματική Εργασία

ΛΦΚ: Λέξη/Φράση Κλειδί

ΔΟΣ: Διάγραμμα οντοτήτων συσχετίσεων

ΒΔ: Βάση δεδομένων

ΑΠ: Ανάκτηση Πληροφορίας (Information Retrieval ή IR)

ΕΦΓ: Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing ή NLP)

1 Εισαγωγή

1.1 Στόχος της εργασίας

Στόχος της παρούσας εργασίας είναι η συγκριτική αξιολόγηση αλγορίθμων αυτοματοποιημένης εξαγωγής λέξεων/φράσεων κλειδιών (ΛΦΚ) από κείμενα της αγγλικής γλώσσας και επίδειξη της λειτουργίας τους μέσω της ανάπτυξης μιας εφαρμογής που θα προτείνει συστάσεις για περιοδικά προς δημοσίευση όταν θα λαμβάνει ως είσοδο ένα άρθρο προς δημοσίευση. Ο συγκεκριμένος στόχος, μπορεί να υποδιαιρεθεί στους παρακάτω υπο-στόχους:

- A) Υλοποίηση αλγορίθμων αυτοματοποιημένης εξαγωγής ΛΦΚ σε γλώσσα Python
- B) Συγκριτική αξιολόγηση των αλγορίθμων της περίπτωσης (A), μέσω κατάλληλων μετρικών, με σκοπό την επιλογή του πλέον κατάλληλου αλγορίθμου για την συγκεκριμένη εφαρμογή.
- Γ) Ανάπτυξη της εφαρμογής σε γλώσσα Python με την χρήση του κατάλληλου αλγορίθμου της περίπτωσης (B)

1.2 Δομή της εργασίας

1^ο Κεφάλαιο: Εισαγωγή Στόχος.

2^ο Κεφάλαιο: Θεωρητικό πλαίσιο (Μηχανική Μάθηση – Επεξεργασία Φυσικής γλώσσας (NLP) – Ανάκτηση Πληροφορίας (Information retrieval) -Εξαγωγή ΛΦΚ – Αναζήτηση – Αξιολόγηση.

3^ο Κεφάλαιο: Στόχος (A): Υλοποίηση των αλγορίθμων μη επιβλεπόμενης μάθησης.

4^ο Κεφάλαιο: Στόχος (B) Συγκριτική αξιολόγηση των αλγορίθμων του 3^{ου} κεφαλαίου με ομάδες δεδομένων (datasets). Οι ομάδες δεδομένων που θα χρησιμοποιηθούν είναι οι Krapivin_2009 (2750 πλήρη άρθρα επιστημονικών περιοδικών με τις λέξεις κλειδιά για κάθε άρθρο), και Aminer Citation Network (ACN) (περίπου 2000 περιλήψεις δημοσιευμένων άρθρων που περιέχουν επίσης το περιοδικό και τα επιστημονικά πεδία του κάθε άρθρου). Η συγκριτική αξιολόγηση των αλγορίθμων θα γίνει με την χρήση μετρικών $Precision@k$, $Recall@K$, mean average precision (MAP), mean reciprocal rank (MRR) και normalized discounted cumulative gain (nDCG). Επίσης, θα λάβει χώρα και σύγκριση της ταχύτητας υλοποίησης του κάθε αλγόριθμου.

5^ο Κεφάλαιο: Στο συγκεκριμένο κεφάλαιο θα επεξηγηθεί αναλυτικά η υλοποίηση της εφαρμογής. Η εφαρμογή θα υλοποιηθεί επίσης σε γλώσσα προγραμματισμού Python και συγκεκριμένα με τη χρήση του πλαισίου ανάπτυξης (framework) ανοιχτού κώδικα Streamlit. Επίσης, θα παρουσιαστούν οι περιπτώσεις χρήσης, το διάγραμμα οντοτήτων συσχετίσεων (ΔΟΣ) της βάσης δεδομένων (ΒΔ), οι κλάσεις και τα διαγράμματα τους σε γλώσσα UML.

6^ο Κεφάλαιο: Στο συγκεκριμένο κεφάλαιο θα γίνει μια σύνοψη της εργασίας και θα αναλυθούν τα κυριότερα συμπεράσματα αυτής.

Ακολουθούν οι βιβλιογραφικές αναφορές και το Παράρτημα της εργασίας, όπου παρατίθεται κώδικας υλοποίησης των αλγορίθμων και επιλεγμένα αποσπάσματα κώδικα της εφαρμογής.

2 Θεωρητικό πλαίσιο

2.1 Ορισμοί

Η διαχείριση μεγάλου όγκου εγγράφων αποτελεί καθημερινή δραστηριότητα σε ένα μεγάλο εύρος επιχειρήσεων (π.χ. νομικού ενδιαφέροντος, εμπορικού ενδιαφέροντος, οικονομικού ενδιαφέροντος, παροχών υγείας, δημόσια διοίκηση, εκπαίδευση κλπ). Η πλοήγηση σε αυτά τα έγγραφα με σκοπό την ανάκτηση της πληροφορίας που εμπεριέχουν μπορεί να αποτελέσει εμπόδιο στην αποτελεσματική λειτουργία του εκάστοτε οργανισμού, εάν η ανάκτηση της πληροφορίας είναι μη αποδοτική τόσο από πλευράς χρόνου όσο και από πλευράς ποιότητας της ανακτημένης πληροφορίας. Συνεπώς είναι επιβεβλημένη η ανάγκη για την ανάπτυξη συστημάτων ανάκτησης πληροφοριών (information retrieval systems) από ένα μεγάλο όγκο εγγράφων, ανακτώντας μόνο εκείνα τα έγγραφα που περιέχουν την ζητούμενη πληροφορία στο συντομότερο δυνατό χρονικό διάστημα.

Σύμφωνα με τους Manning et al (2008), ως Ανάκτηση Πληροφορίας (Information Retrieval ή IR ή ΑΠ), θεωρείται η εύρεση πόρων (συνήθως εγγράφων) μη δομημένης φύσης (συνήθως κείμενο) που ικανοποιεί μια ανάγκη εύρεσης πληροφοριών μέσα από μεγάλες συλλογές (συνήθως αποθηκευμένες σε υπολογιστές). Σύμφωνα με τον παραπάνω ορισμό, προκειμένου ο υπολογιστής να ανακτήσει τις πληροφορίες που του ζητήθηκαν, θα πρέπει να είναι σε θέση να «αντιλαμβάνεται» τα κείμενα που είναι αποθηκευμένα σε αυτόν. Συνεπώς ένα ψηφιακό σύστημα ανάκτησης πληροφοριών θα πρέπει οπωσδήποτε να υλοποιεί και ένα σύστημα Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing ή NLP ή ΕΦΓ) που να είναι σε θέση να αναλύει, να επεξεργάζεται και να αναπαριστά τα κείμενα (γραπτά ή προφορικά) της φυσικής γλώσσας (Liddy, 2001).

Παρακάτω, θα ακολουθήσει πιο λεπτομερής περιγραφή πρώτα της ΕΦΓ και ακολούθως της ΑΠ, καθώς η ΑΠ προϋποθέτει το σύστημα να έχει υλοποιήσει την ΕΦΓ. Επίσης, καθώς η παρούσα ΔΕ αφορά την εφαρμογή των ΑΠ και ΕΦΓ σε αγγλικό κείμενο, όλα τα παραδείγματα του κεφαλαίου, θα αφορούν κείμενο της αγγλικής γλώσσας.

2.2 Επεξεργασία Φυσικής Γλώσσας (NLP)

2.2.1 Γλωσσική Προ-επεξεργασία

Τα περισσότερα έγγραφα κειμένου έχουν κάποιο είδος ιεραρχικής δομής. Λαμβάνοντας ένα βιβλίο ως παράδειγμα, το οποίο συνήθως χωρίζεται σε κεφάλαια, τα κεφάλαια μπορεί να έχουν ενότητες, και κάθε ενότητα αποτελείται από παραγράφους. Οι παράγραφοι περιέχουν

προτάσεις οι οποίες αποτελούνται από λεκτικά (tokens). Το κάθε λεκτικό αποτελεί μια αυτόνομη νοηματική οντότητα, η οποία με την σειρά της αποτελείται από ένα σύνολο εκτυπώσιμων (ή μη) χαρακτήρων. Ως προς την αναπαράσταση του σε ένα υπολογιστικό σύστημα, ένα έγγραφο ισοδυναμεί με ένα αλφαριθμητικό (string) ή ως μια λίστα χαρακτήρων. Η αναπαράσταση ενός εγγράφου ως λίστα χαρακτήρων είναι πολύ δύσκολο να κατανοηθεί και να υποστεί επεξεργασία τόσο από τα υπολογιστικά συστήματα όσο και για ανθρώπους και για τον λόγο αυτό κάθε κείμενο πρέπει να υποστεί γλωσσική προ-επεξεργασία, ακολουθώντας τα παρακάτω στάδια με την σειρά που αναφέρονται (Anandarajan et al, 2019):

- Διακριτοποίηση λεκτικών (tokenization)
- Τυποποίηση – Καθαρισμός (Standardization and cleansing)
- Αφαίρεση «κοινότυπων» λέξεων (Stop word removal)
- Εύρεση Ρίζας ή Λήμματος (Stem and Lemmatize)

Παρακάτω ακολουθεί πιο λεπτομερής περιγραφή του κάθε σταδίου.

2.2.2 Διακριτοποίηση λεκτικών (tokenization)

Με την διακριτοποίηση λεκτικών, στοχεύουμε στην διάσπαση των προτάσεων στα επιμέρους στοιχεία τους (λέξεις ή λεκτικά), με σκοπό την περαιτέρω επεξεργασία τους. Σε πολλές γλώσσες (συμπεριλαμβανομένης της αγγλικής και της ελληνικής) η διακριτοποίηση μπορεί να επιτευχθεί αρκετά εύκολα διασπώντας τις προτάσεις όπου υπάρχει ο χαρακτήρας του κενού (white space). Αν θεωρήσουμε την πρόταση:

“The most exciting thing is that I love playing the guitar twice per week.”,

με απαλοιφή του κενού προκύπτουν τα παρακάτω λεκτικά (ή λέξεις):

ΓΜ = { ‘The’, ‘most’, ‘exciting’, ‘thing’, ‘is’, ‘that’, ‘I’, ‘love’, ‘playing’, ‘the’, ‘guitar’,
‘twice’, ‘per’, ‘week.’ }

Αν και η παραπάνω διαδικασία είναι αρκετά εύκολη σε ορισμένες γλώσσες, παρόλα αυτά δεν ισχύει το ίδιο και σε άλλες γλώσσες (π.χ. Κινέζικα) που οι λέξεις δεν έχουν κενό μεταξύ τους (Herbrich and Graepel., 2010).

Μια άλλη παράμετρος που θα πρέπει να ληφθεί υπόψη σε αυτό το στάδιο προ-επεξεργασίας είναι οι εκφράσεις πολλαπλών λέξεων, για παράδειγμα, ονόματα τοποθεσιών ή οντοτήτων, (π.χ. "Los Angeles"). Η διαδικασία ανίχνευσης και επεξεργασίας τέτοιων περιπτώσεων είναι μάλλον δύσκολη, εκτός και εάν η βάση μας περιέχει όλες τις πιθανές εκφράσεις πολλαπλών λέξεων που θα μπορούσαν να θεωρηθούν ως αυτόνομα λεκτικά. Ακόμα

και σε μια τέτοια περίπτωση, η διαδικασία εντοπισμού τέτοιων φράσεων θα ήταν χρονοβόρα και με υψηλές απαιτήσεις σε μνήμη. Μια πρακτική που αναφέρεται στην βιβλιογραφία για να χειριστούμε τις εκφράσεις πολλαπλών λέξεων είναι η δημιουργία των n-grams (Herbrich and Graepel., 2010, Anandarajan et al, 2019). Τα n-grams είναι αυτόνομα λεκτικά που αποτελούνται από n συνεχόμενες λέξεις με σειρά εμφάνισης όπως διαβάζονται στο κείμενο (π.χ. για τις περισσότερες γλώσσες από αριστερά προς τα δεξιά). Συνεπώς εάν τα λεκτικά περιέχουν n=1 λέξεις τότε πρόκειται για unigrams, εάν περιέχουν n=2 λέξεις τότε πρόκειται για bi-grams, για n=3 πρόκειται για tri-grams κλπ. Στην πρόταση της προηγούμενης ενότητας τα unigrams είναι τα παρακάτω:

{‘The’, ‘most’, ‘exciting’, ‘thing’, ‘is’, ‘that’, ‘I’, ‘love’, ‘playing’, ‘the’, ‘guitar’, ‘twice’, ‘per’, ‘week.’}

Τα bi-grams είναι

{‘The most’, ‘most exciting’, ‘exciting thing’, ‘thing is’, ‘is that’, ‘that I’, ‘I love’, ‘love playing’, ‘playing the’, ‘the guitar’, ‘guitar twice’, ‘twice per’, ‘per week.’}

ενώ τα tri-grams είναι

{‘The most exciting’, ‘most exciting thing’, ‘exciting thing is’, ‘thing is that’, ‘is that I’, ‘that I love’, ‘I love playing’, ‘love playing the’, ‘playing the guitar’, ‘the guitar twice’, ‘guitar twice per’, ‘twice per week.’}

2.2.3 Τυποποίηση-Καθαρισμός (Standardization and cleansing)

Μετά τη διάσπαση του κειμένου, τα λεκτικά θα πρέπει να υποστούν περαιτέρω επεξεργασία προκειμένου να είναι ευκολότερη η εφαρμογή του ΑΠ σε αυτά. Τα λεκτικά σε αυτό το σημείο μπορεί να περιλαμβάνουν μη αλφαριθμητικούς χαρακτήρες, όπως απόστροφους ή σημεία στίξης, που μπορεί να είναι προβληματικοί για την ανάλυση εγγράφων. Λέξεις όπως "is not" και "isn't" θα πρέπει να αντιστοιχούν στο ίδιο λεκτικό καθώς πρόκειται για λεκτικά με ακριβώς το ίδιο νόημα. Επίσης, στο παράδειγμα της προηγούμενης παραγράφου το λεκτικό "week.", θα πρέπει να αντιστοιχηθεί στο "week", γιατί σε διαφορετική περίπτωση ο υπολογιστής θα θεωρήσει τα παραπάνω λεκτικά ως δύο ξεχωριστές οντότητες.

Ακόμα, σε αυτό το στάδιο μια κοινή πρακτική είναι η μετατροπή των λεκτικών σε πεζούς χαρακτήρες, αν και σύμφωνα με τους Herbrich and Graepel, (2010) υπάρχουν κάποιες εξαιρέσεις που μια τέτοια πρακτική μπορεί να αλλοιώσει το νόημα τους. Για παράδειγμα τα λεκτικά "US" (=United States) ή "IT" (=Information Technology) εάν μετατραπούν σε πεζά θα γίνουν "us" (=εμάς) ή "it" (=αυτό).

2.2.4 Αφαίρεση «κοινότυπων» λέξεων (Stop words removal)

Μόλις το κείμενο μετατραπεί σε μια ροή από λεκτικά, η προ-επεξεργασία συνεχίζεται με τον καθορισμό εκείνων των λεκτικών που είναι χρήσιμα για την εφαρμογή. Σε αυτό το στάδιο, λαμβάνει χώρα ένα φιλτράρισμα των λεκτικών παραλείποντας εκείνα που θεωρούνται ασήμαντα ως προς το νόημα που εμπεριέχουν (stop words).

Αν και η επιλογή του φιλτραρίσματος εξαρτάται σε μεγάλο βαθμό από την εφαρμογή ΑΠ, ωστόσο υπάρχουν κάποιες «κοινότυπες» λέξεις που έχουν μικρή πιθανότητα να συνεισφέρουν νοηματικά ή να εισάγουν νέα πληροφορία στο κείμενο. Τέτοιες κοινές λέξεις στην αγγλική γλώσσα είναι "the", "to", "and", "of", "be/is/are/was/am" και άλλες. Αυτές οι λέξεις έχουν κυρίως συντακτική χρησιμότητα και για αυτό υπερτερούν αριθμητικά στα κείμενα.

Εμπειρικά, η κατανομή όλων των λέξεων ενός κειμένου φυσικής γλώσσας ακολουθεί το νόμο του Zipf (Zipf, 1932). Σύμφωνα με τον νόμο του Zipf για μια λέξη εντός ενός κειμένου ισχύει η παρακάτω σχέση:

$$\text{συχνότητα εμφάνισης} = \alpha \frac{1}{\text{σειρά κατάταξης}} \quad (1)$$

Σύμφωνα με την παραπάνω σχέση κάποιες λέξεις σε ένα κείμενο εμφανίζονται πολύ περισσότερες φορές σε σχέση με κάποιες άλλες. Για παράδειγμα η πρώτη πιο συχνή λέξη εμφανίζεται περίπου διπλάσιες φορές σε σχέση με τη δεύτερη πιο συχνή, περίπου τριπλάσιες φορές σε σχέση με την τρίτη πιο συχνή κ.ο.κ. Συνεπώς, η εννοιολογική διαφοροποίηση μεταξύ κειμένων, δεν θα πρέπει να στηρίζεται αποκλειστικά στις λέξεις που έχουν πολύ μεγάλη συχνότητα εμφάνισης (όπως οι «κοινότυπες λέξεις»), καθώς αυτές οι λέξεις αναμένεται να είναι κοινές μεταξύ τους. Αντιθέτως, λέξεις με χαμηλότερη συχνότητα εμφάνισης αναμένεται να μην είναι κοινές μεταξύ των κειμένων και κατά συνέπεια να αποτελούν παράγοντες που τα διαφοροποιούν μεταξύ τους. Στο παράδειγμα της ενότητας 2.2.1, μετά το φιλτράρισμα των «κοινότυπων» λέξεων, τα unigram λεκτικά της πρότασης θα μπορούσαν να είναι τα παρακάτω:

{‘most’, ‘exciting’, ‘thing’, ‘love’, ‘playing’, ‘guitar’, ‘twice’, ‘per’, ‘week’}

Μπορεί να παρατηρήσει κανείς, ότι δεν έχει χαθεί καμία νοηματική πληροφορία, μετά την εξάλειψη των «κοινότυπων» λέξεων, ακόμα και στην περίπτωση των bi-grams ή tri-grams αφού οι νοηματικές φράσεις (π.χ. ‘exciting thing’, ‘twice per week’, ‘playing guitar’ κ.α.) εμπεριέχονται στα n-grams μετά την απαλοιφή των «κοινότυπων» λέξεων.

Άλλα λεκτικά που θα μπορούσαν να αφαιρεθούν μέσω του φιλτραρίσματος των «κοινότυπων» λέξεων είναι σύμβολα, αριθμοί και ημερομηνίες υπό την προϋπόθεση ότι δεν συνεισφέρουν νοηματικά για μια δεδομένη ΑΠ εφαρμογή. Η παράλειψη των «κοινότυπων» λέξεων μπορεί να βελτιώσει αλλά και να υποβαθμίσει την απόδοση μιας ΑΠ εφαρμογής. Η βελτίωση της απόδοσης επιτυγχάνεται με την ελάττωση του θορύβου στα δεδομένα, απορρίπτοντας τα λεκτικά που είναι πραγματικά ασήμαντα για την εφαρμογή. Στον αντίποδα υπάρχει πιθανότητα να χαθεί σημαντική πληροφορία εάν κάποια λεκτικά εκληφθούν λανθασμένα ως «κοινότυπες» λέξεις (για παράδειγμα "US" ή "IT"). Κατά συνέπεια, η επιλογή των «κοινότυπων» λέξεων θα πρέπει να γίνεται με γνώμονα τις απαιτήσεις της συγκεκριμένης ΑΠ εφαρμογής.

Σύμφωνα με τους Anandarajan et al, (2019) στην λίστα των «κοινότυπων» λέξεων μιας εφαρμογής, μπορεί να συμπεριληφθούν και λέξεις οι οποίες σχετίζονται με τη θεματική ενότητα των εγγράφων που διαχειρίζεται μια εφαρμογή. Τέτοιες λέξεις αφενός αναμένεται να εμπεριέχονται περισσότερο συχνά στα έγγραφα και αφετέρου δεν προσφέρουν κάποια εξειδικευμένη πληροφορία για έγγραφα της ίδιας θεματικής ενότητας. Για παράδειγμα σε ένα ευρετήριο εγγράφων νομικού περιεχομένου, λέξεις όπως "law", "legislation", "without prejudice to", "provisions" κλπ είναι χαρακτηριστικές για την συγκεκριμένη θεματική ενότητα και θα μπορούσαν να εκληφθούν ως «κοινότυπες». Μια καλή πρακτική για τον εμπλουτισμό της λίστας των «κοινότυπων» λέξεων είναι η προσθήκη σε αυτή, των λέξεων με τη μεγαλύτερη συχνότητα εμφάνισης στα κείμενα της ΑΠ εφαρμογής (Anandarajan et al, 2019). Σε κάθε περίπτωση ωστόσο, θα πρέπει να λαμβάνεται σοβαρά υπόψη το ενδεχόμενο να χαθεί κάποια σημαντική πληροφορία.

2.2.5 Εύρεση Ρίζας ή Λήμματος (Stem and Lemmatize)

Αυτό είναι το τελευταίο στάδιο της προ-επεξεργασίας και εμπεριέχει γλωσσολογική ανάλυση του κάθε λεκτικού. Κατά την εύρεση της ρίζας (stemming) αφαιρούνται από το μόρφημα οι διάφορες καταλήξεις του με σκοπό να πάρουμε μόνο την ρίζα (Porter, 1980). Για παράδειγμα οι λέξεις {"train", "trains", "trained", "training", "trainer"} μετά από αφαίρεση των καταλήξεων τους αποκαλύπτουν την ρίζα "train". Στην περίπτωση που ο αλγόριθμος εύρεσης ρίζας, λαμβάνει υπόψη και το γραμματικό μέρος του λόγου (π.χ. ουσιαστικό, τρίτο πρόσωπο, ανώμαλο ρήμα κλπ) τότε η διαδικασία ονομάζεται ληματοποίηση (lemmatization) (Manning et al, 2008). Για την αναγνώριση του μέρους του λόγου που ανήκει κάθε λεκτικού χρησιμοποιούνται διάφορες τεχνικές (part of speech tagging ή POS tagging), που επιστρέφουν το κάθε λεκτικό με μια ετικέτα που αναφέρει το μέρος του λόγου που ανήκει (Manning and

Schótzze 1999). Ο σκοπός της εύρεσης ρίζας ή της λημματοποίησης είναι η ομαδοποίηση λεκτικών με κοινή ρίζα (ή λήμμα) και αντιπροσώπευση τους από την ρίζα (ή το λήμμα) τους. Με τον τρόπο αυτό επιτυγχάνεται περαιτέρω μείωση του τελικού πλήθους των λεκτικών, τα οποία θα τροφοδοτήσουν την ΑΠ εφαρμογή. Σύμφωνα με τους Anandarajan et al, (2019) οι συγκεκριμένες διαδικασίες αν και μειώνουν αισθητά τον τελικό πλήθος των λεκτικών, θα πρέπει να χρησιμοποιούνται με προσοχή λόγω του γεγονότος ότι αρκετές λέξεις ρίζες ή λήμματα έχουν περισσότερες από μια ερμηνείες.

2.3 Εξαγωγή Λέξεων/Φράσεων Κλειδιών (Keywords/Keyphrases) από κείμενα

Ένας από τους στόχους της ΕΦΓ είναι η εύρεση των πλέον αντιπροσωπευτικών ΛΦΚ για κάθε έγγραφο, προκειμένου το εν λόγω έγγραφο μέσω της μηχανής ΑΠ να ανακτηθεί από το σύστημα, όταν ένας χρήστης αναζητήσει μια πληροφορία μέσω ενός ερωτήματος. Στην βιβλιογραφία έχουν προταθεί πολλές μέθοδοι επιβλεπόμενης ή μη επιβλεπόμενης μηχανικής μάθησης, που έχουν ως σκοπό να εξαγουν μεμονωμένες λέξεις ή συνδυασμούς λέξεων από ένα συγκεκριμένο κείμενο και να τις κατατάζουν ως πιθανές ΛΦΚ για το ίδιο το κείμενο (Siddiqi and Sharan 2015, Gupta 2017, Mahedi Hasan et al, 2017, Nadim et al 2023). Η κύρια διαφορά μεταξύ των μεθόδων επιβλεπόμενης μάθησης σε σχέση με εκείνες της μη επιβλεπόμενης μάθησης είναι ότι οι πρώτες προαπαιτούν την ύπαρξη ενός κατάλληλα εκπαιδευμένου μοντέλου που θα προβλέπει τις ΛΦΚ ενός μεμονωμένου εγγράφου. Η εκπαίδευση του μοντέλου πρόβλεψης γίνεται μέσω ενός συνόλου εγγράφων (training set) με επισημασμένα (labelled) δεδομένα. Αν και οι μέθοδοι επιβλεπόμενης μάθησης είναι πιο αποτελεσματικές σχέση με εκείνες της μη επιβλεπόμενης μάθησης, ωστόσο έχουν κάποια σοβαρά μειονεκτήματα. Τα μειονεκτήματα τους σύμφωνα με τους Nadim et al, 2023 είναι (α) ο φόρτος που απαιτείται για την επισήμανση του συνόλου εγγράφων που θα χρησιμοποιηθεί για την εκπαίδευση (β) η έλλειψη γενικότητας και (γ) η ανάγκη για συνεχή εκπαίδευση που με τη σειρά της αυξάνει το φόρτο λόγω της ανάγκης νέων επισημασμένων δεδομένων. Οι πιο διαδεδομένες μέθοδοι επιβλεπόμενης μάθησης είναι αλγόριθμος KEA (Keyword Extraction Algorithm, Witten et al, 1999) και το εκπαιδευμένο μοντέλο BERT (Devlin et al, 2019, Sharma and Li, 2019).

Στις μεθόδους μη επιβλεπόμενης μηχανικής μάθησης δεν απαιτείται η ύπαρξη κάποιου μοντέλου εκπαίδευσης και κάθε έγγραφο θεωρείται μια ξεχωριστή οντότητα. Η κατάταξη των πιθανών ΛΦΚ επιτυγχάνεται με τις παρακάτω μεθόδους (Gupta, 2017, Nadim et al, 2023):

(α) Στατιστικές μέθοδοι

I. Βασισμένες σε συχνότητα εμφάνισης των όρων (Frequency-based)

II. Βασισμένες σε Γράφους (Graph-based)

(β) Γλωσσολογικές/Συντακτικές μέθοδοι

(γ) Συνδυασμένες μέθοδοι.

Στις Στατιστικές μεθόδους ανήκουν κυρίως μέθοδοι που σχετίζονται με την συχνότητα εμφάνισης μιας υποψήφιας λέξης/φράσης κλειδί στο κείμενο, ή συνύπαρξη της με άλλες λέξεις στην ίδια παράγραφο ή σε όλο το κείμενο. Στις γλωσσολογικές μεθόδους ανήκουν μέθοδοι που θα μπορούσαν να κατηγοριοποιήσουν νοηματικά μια λέξη σύμφωνα με το μέρος του λόγου που ανήκει, εάν εμφανίζεται στην αρχή του κειμένου, εάν είναι σε bold ή italics κλπ. Οι συνδυασμένες μέθοδοι χρησιμοποιούν ταυτόχρονα τις στατιστικές και γλωσσολογικές μεθόδους και είναι οι πιο συνηθισμένες. Για παράδειγμα όταν λαμβάνει προ-επεξεργασία κειμένου σύμφωνα με ΕΦΓ τότε μιλάμε για το συνδυασμένο μοντέλο. Παρακάτω ακολουθεί μια πιο λεπτομερής περιγραφή των μεθόδων μη επιβλεπόμενης μάθησης που θα χρησιμοποιηθούν στην παρούσα εργασία.

2.3.1 Αλγόριθμος TF-IDF

Πρόκειται ενδεχομένως για την πιο ευρέως διαδεδομένη μέθοδο (Manning et al, 2008) και στηρίζεται στην συχνότητα εμφάνισης ενός όρου στο έγγραφο (TF), λαμβάνοντας επίσης υπόψη και τον νόμο του Zipf (Zipf, 1932). Η τιμή TF-IDF δίδεται από την παρακάτω εξίσωση:

$$TF - IDF = TF \times IDF \quad (2)$$

Όπου IDF (Inverse Document Frequency) είναι η αντίστροφη συχνότητα εμφάνισης ενός όρου στην συλλογή εγγράφων. Η τιμή IDF δίδεται από την παρακάτω σχέση:

$$IDF = \log \left(\frac{N}{n} \right) \quad (3)$$

όπου N είναι ο συνολικός αριθμός εγγράφων της συλλογής και n ο αριθμός των εγγράφων που περιέχουν τον όρο. Παρατηρούμε λοιπόν ότι όσο η ποσότητα N/n τείνει στο 1 (δηλαδή όσο το n πλησιάζει την μέγιστη τιμή που μπορεί να φτάσει) τόσο η τιμή IDF πλησιάζει στο μηδέν. Συνεπώς ένας όρος που εμφανίζεται με υψηλή συχνότητα αλλά περιέχεται σε πολλά έγγραφα,

τότε πιθανότατα είναι ένας κοινότυπος όρος και η τιμή TF-IDF θα είναι πολύ χαμηλή ανεξάρτητα εάν το TF είναι υψηλό, ικανοποιώντας τον νόμο του Zipf (**Zipf, 1932**). Αντιθέτως, ένας όρος που απαντάται συχνά σε ένα έγγραφο αλλά βρίσκεται σε έναν συγκεκριμένο υποσύνολο εγγράφων της συλλογής τότε έχει υψηλό IDF, το οποίο αντισταθμίζει το χαμηλότερο TF σε σχέση με τους κοινότυπους όρους και βάζει σοβαρή υποψηφιότητα να είναι σημαντικό ΛΦΚ εάν το TF του είναι υψηλότερο από το TF άλλων -μη κοινότυπων- όρων. Για να οριστεί μαθηματικά ο όρος IDF θα πρέπει να ισχύει η παρακάτω σχέση:

$$\frac{N}{n} \neq 0, \quad \text{δηλαδή } N \neq 0 \text{ και } n \neq 0$$

Συνεπώς ο μόνος περιορισμός για τον υπολογισμό του όρου IDF είναι ότι θα πρέπει να υπολογίζεται σε μη κενές συλλογές εγγράφων ($N=0$). Σε ότι αφορά στον όρο n αυτός θα είναι πάντοτε διαφορετικός του μηδέν, αφού για να εξεταστεί ως πιθανή ΛΦΚ θα έχει βρεθεί τουλάχιστον σε ένα κείμενο. Παρόλα αυτά η γενικευμένη μαθηματική αποτύπωση της συνάρτησης TF-IDF είναι η παρακάτω:

$$TF - IDF = \begin{cases} 0, & N = 0 \text{ ή } n = 0 \\ TF \times \log\left(\frac{N}{n}\right), & N \neq 0 \text{ και } n \neq 0 \end{cases} \quad (4)$$

όπου N είναι ο συνολικός αριθμός εγγράφων της συλλογής και n ο αριθμός εγγράφων που περιέχουν τον όρο.

2.3.2 Αλγόριθμος RAKE

Ο αλγόριθμος RAKE (Rose et al, 2010) χρησιμοποιεί ως στατιστικές παραμέτρους (α) τον αριθμό συνολικών εμφανίσεων ενός όρου σε ένα έγγραφο και (β) τον αριθμό συνυπάρξεων αυτού του όρου με τους υπόλοιπους όρους του εγγράφου. Σύμφωνα με τους Rose et al, (2010) έχει παρατηρηθεί ότι οι ΛΦΚ κατά κανόνα δεν περιέχουν «κοινότυπους» όρους. Έτσι, οι παραπάνω ερευνητές ανέπτυξαν έναν αλγόριθμο ο οποίος διασπά το κείμενο σε φράσεις, στα σημεία που υπάρχουν «κοινότυποι» όροι ή στα διάφορα σημεία στίξης που υποδηλώνουν τέλος πρότασης (π.χ. τελείες) ή ακόμα και νοήματος (π.χ. κόμματα, παρενθέσεις). Με τον τρόπο αυτό αποκαλύπτονται φράσεις που περιέχουν πιθανούς όρους κλειδιά. Οι πιθανοί όροι βαθμολογούνται ανάλογα με τον βαθμό συνυπάρξεως με άλλους πιθανούς όρους του εγγράφου. Για να γίνει αυτό κατανοητό ας δούμε το παρακάτω απόσπασμα κειμένου:

Welcome to the tutorial on natural language processing. In that tutorial, we are going to use python, as it is the predominant programming language for machine learning tasks. Natural language processing is one of machine learning fields.

Επίσης ας θεωρήσουμε ως «κοινότυπους» όρους τις λέξεις {to, the, on, in, that, we, are, going, as, it, is, for, of }, ο αλγόριθμος RAKE αρχικά θα διασπάσει το κείμενο όπου συναντήσει «κοινότυπη» λέξη ή σημείο στίξης που υποδηλώνουν τέλος πρότασης συντακτικά (π.χ. τελεία) ή νοηματικά (π.χ. κόμμα), όπως φαίνεται παρακάτω:

Welcome to the tutorial on natural language processing. In that tutorial, we are going to use python, as it is the predominant programming language for machine learning tasks. Natural language processing is one of the machine learning fields.

Μετά την διάσπαση θα δημιουργηθούν οι παρακάτω φράσεις, μετά από μετατροπή σε πεζούς χαρακτήρες, όπως έχει προταθεί στην ενότητα της ΕΦΓ:

{welcome – tutorial - natural language processing – tutorial - use python – predominant programming language - machine learning tasks- natural language processing – one – machine learning fields}

Οι πιθανοί όροι κλειδιά είναι η κάθε μια λέξη που απαρτίζει την κάθε φράση που αναφέρεται παραπάνω. Για παράδειγμα κάποιιοι από τους πιθανούς όρους κλειδιά είναι οι {natural, language, processing} που απαρτίζουν την φράση “natural language processing” που αποκάλυψε ο αλγόριθμος σε αυτό το στάδιο εκτέλεσης του. Οι όροι που απαρτίζουν την ίδια φράση (π.χ. οι όροι του παραδείγματος) ονομάζονται γειτονικοί όροι.

Στο επόμενο στάδιο του αλγορίθμου, δημιουργείται ο πίνακας συνύπαρξης (co-occurrence matrix) των πιθανών όρων. Ο πίνακας συνύπαρξης είναι ένας πίνακας $N \times N$, όπου N το πλήθος όλων των πιθανών όρων του εγγράφου. Τόσο οι στήλες όσο και οι γραμμές του πίνακα αντιστοιχούν σε κάθε ένα από τους πιθανούς όρους ενώ όλα τα κελιά αρχικοποιούνται στην τιμή μηδέν. Ακολούθως, ο πίνακας συνύπαρξης ενημερώνεται για κάθε φράση, όπου ο κάθε όρος της φράσης αυξάνει κατά 1 τα κελιά όλων των γειτονικών του όρων που βρίσκονται στην ίδια γραμμή αλλά και του ίδιου του όρου. Για παράδειγμα για την φράση natural language processing, ο όρος natural θα αυξήσει κατά 1 την τιμή των κελιών [natural, natural], [natural, language] και [natural, processing], όρος language θα αυξήσει κατά 1 την τιμή των κελιών [language, language], [language, natural] και [language, processing] και ο όρος processing θα αυξήσει κατά 1 την τιμή των κελιών [processing, processing], [processing, natural] και

[processing, language]. Για το παραπάνω απόσπασμα κειμένου, ο πίνακας συνύπαρξης, των όρων του, λαμβάνει την παρακάτω μορφή:

	welcome	tutorial	natural	language	processing	use	python	predominant	programming	language	machine	learning	tasks	one	fields
welcome	1														
tutorial		2													
natural			1	1	1										
language			1	1	1										
processing			1	1	1										
use						1	1								
python						1	1								
predominant								1	1	1					
programming								1	1	1					
language								1	1	1					
machine											2	2	1		1
learning											2	2	1		1
tasks											1	1	1		
one														1	
fields											1	1			1
frequency (f)	1	2	1	1	1	1	1	1	1	1	2	2	1	1	1
degree (d)	1	2	3	3	3	2	2	3	3	3	6	6	3	1	3
d/f	1	1	3	3	3	2	2	3	3	3	3	3	3	1	3

Πίνακας 2-1 Πίνακας συνύπαρξης όρων του RAKE

Για την βαθμολόγηση των λέξεων κλειδιά οι Rose et al, (2010) πρότειναν τρεις μετρικές:

- (α) Την συχνότητα εμφάνισης κάποιου πιθανού όρου
- (β) τον βαθμό συνύπαρξης με τους άλλους πιθανούς όρους (degree), και
- (γ) το πηλίκο (α) / (γ)

Για τις παραπάνω μετρικές προκύπτουν κάποια ενδιαφέροντα συμπεράσματα. Η συχνότητα εμφάνισης (αφού έχουν απομακρυνθεί οι «κοινότυποι» όροι) δίνει πλεονέκτημα στις λέξεις που εμφανίζονται περισσότερο στο κείμενο {tutorial, machine, learning}. Με βάση τη μετρική της συχνότητας εμφάνισης, το παραπάνω κείμενο θα μπορούσε να χαρακτηριστεί ως ένα machine learning tutorial. Από την άλλη μεριά, ο βαθμός συνύπαρξης δίνει έμφαση περισσότερο στις λέξεις που συνυπάρχουν μαζί με άλλες όπως οι {machine, learning}. Σύμφωνα με την (β) μετρική το παραπάνω κείμενο θα μπορούσε να χαρακτηριστεί ότι αφορά το machine learning. Σύμφωνα με την (γ) μετρική, οι ερευνητές προσπάθησαν να εφαρμόσουν τον νόμο του Zipf (Zipf, 1932) και στους πιθανούς όρους του εγγράφου, ώστε να ευνοήσουν εκείνους τους πιθανούς όρους που βρίσκονται σε μεγάλες φράσεις. Οι πιθανοί όροι με την

υψηλότερη βαθμολογία της (γ) μετρικής για το απόσπασμα κειμένου είναι {natural, language, processing, predominant, programming, language, machine, learning, tasks, fields}, οι οποίοι επεξηγούν καλύτερα το νόημα του κειμένου, αν και δεν περιέχουν την λέξη python που είναι η predominant programming language. Φαίνεται ωστόσο, ότι η μετρική (γ) επεξηγεί καλύτερα το νόημα του κειμένου σε σχέση με την (α) ή την (β). Επίσης, λέξεις όπως {one, welcome}, οι οποίες δεν συνυπάρχουν με άλλες λέξεις βαθμολογούνται πολύ χαμηλότερα σε σχέση με εκείνες που συνυπάρχουν με άλλες λέξεις. Σε ότι αφορά στην βαθμολογία των n-grams, οι Rose et al (2010) προτείνουν το άθροισμα των βαθμολογιών των επιμέρους όρων τους.

2.3.3 Αλγόριθμος Yet Another Keyword Extraction (YAKE)

Ο αλγόριθμος YAKE (Campos et al, 2020), ανήκει στην κατηγορία των στατιστικών αλγορίθμων που ωστόσο λαμβάνουν υπόψη -περισσότερο σε σχέση με τους προηγούμενους αλγόριθμους που αναφέρθηκαν- και χαρακτηριστικά του εγγράφου. Πριν εφαρμοστεί ο αλγόριθμος, το έγγραφο υπόκειται σε προ-επεξεργασία ώστε τουλάχιστον να απομακρυνθούν οι κοινότυποι όροι από αυτό. Οι παράμετροι που χρησιμοποιεί ο αλγόριθμος YAKE είναι οι παρακάτω:

- (α) Η κανονικοποιημένη συχνότητα εμφάνισης του όρου (f_{norm})
- (β) Η αναλογία των φράσεων του εγγράφου που απαντάται ένας όρος (f_{sent})
- (γ) Σχετικότητα όρου με το περιεχόμενο του εγγράφου (f_{rel})
- (δ) Η σχετική θέση των προτάσεων εντός του εγγράφου που απαντάται ένας όρος (f_{pos})
- (ε) Τυπογραφικά χαρακτηριστικά του όρου (f_{case})

Η κανονικοποιημένη συχνότητα εμφάνισης ενός όρου (t) στο έγγραφο υπολογίζεται από την παρακάτω σχέση:

$$f_{norm}(t) = \frac{tf(t)}{tf_{mean} + tf_{std}} \quad (5)$$

όπου $tf(t)$ είναι η συχνότητα εμφάνισης ενός όρου στο έγγραφο, tf_{mean} είναι η μέση συχνότητα εμφάνισης των όρων του εγγράφου και tf_{std} η τυπική απόκλιση των συχνοτήτων εμφάνισης των όρων του εγγράφου.

Η αναλογία φράσεων του εγγράφου στις οποίες απαντάται ένας όρος (t), υπολογίζεται από την παρακάτω σχέση:

$$f_{sent}(t) = \frac{sf(t)}{|sentences|} \quad (6)$$

όπου $sf(t)$ είναι ο συνολικός αριθμός φράσεων (sentences) που περιέχουν τον όρο (t) και $|sentences|$ ο συνολικός αριθμός φράσεων του εγγράφου. Παρόμοια με τον f_{norm} , υψηλές τιμές του f_{sent} σχετίζονται με υψηλής σημασίας όρους για το έγγραφο.

Για τον υπολογισμό της σχετικότητας του όρου με το νόημα του εγγράφου, χρησιμοποιείται ο βαθμός συνύπαρξης του με τους υπόλοιπους όρους του εγγράφου. Σύμφωνα με τους Campos et al (2020), όταν ένας όρος είναι «διεσπαρμένος» με πολλούς άλλους όρους σε ένα έγγραφο τότε είναι λιγότερο πιθανό να είναι κάποιος εξειδικευμένος όρος κλειδί. Η σχετικότητα του όρου (t) με το νόημα του εγγράφου, υπολογίζεται από την παρακάτω σχέση:

$$f_{rel}(t) = 1 + \frac{tf(t)}{tf_{max} \cdot \sum_{i=1}^n C[t][i]} \cdot \sum_{\substack{i=1 \\ C[t][i] \neq 0}}^n 1 \quad (7)$$

όπου $tf(t)$ η συχνότητα του όρου, tf_{max} η μέγιστη συχνότητα που εμφανίζεται ένας όρος στο έγγραφο και C ο πίνακας συνύπαρξης όρων του εγγράφου. Όπως φαίνεται από την παραπάνω σχέση, η σχετικότητα του όρου είναι αντιστρόφως ανάλογη της ποσότητας $\sum_{i=1}^n C[t][i]$ που υποδηλώνει το πόσες φορές συνυπάρχει ένας όρος με τους υπόλοιπους όρους του εγγράφου και ευθέως ανάλογη της ποσότητας $\sum_{\substack{i=1 \\ C[t][i] \neq 0}}^n 1$, που υπολογίζει με πόσους όρους συνυπάρχει

ο συγκεκριμένος όρος. Παρατηρούμε λοιπόν, ότι σε αντίθεση με τον RAKE, ο YAKE! δίνει συγκριτικό πλεονέκτημα σε όρους που εμφανίζονται πολλές φορές αλλά σε όσο το δυνατόν μικρότερες φράσεις ή σε όρους που μπορεί να συνυπάρχουν με άλλους όρους, αλλά λίγες φορές.

Η σχετική θέση των προτάσεων, που απαντάται ένας όρος (t), εντός του εγγράφου υπολογίζεται από την παρακάτω σχέση:

$$f_{pos}(t) = \ln(\ln(\text{median}(\text{pos}_{sent}(t)) + e)) \quad (8)$$

όπου $\ln$ ο νεπέριος λογάριθμος, e η σταθερά 2,72.. και $\text{median}(\text{pos}_{sent}(t))$ είναι η διάμεσος των προτάσεων που απαντάται ο όρος t. Η μετρική f_{pos} χρησιμοποιείται στον αλγόριθμο καθώς σύμφωνα με τους Campos et al (2020), οι λέξεις κλειδιά είναι πιθανότερο να απαντώνται στην αρχή του εγγράφου όπως στον τίτλο ή την εισαγωγή, όπου μάλλον και θα επεξηγούν τις επόμενες παραγράφους του εγγράφου.

Τέλος, τυπογραφικά χαρακτηριστικά όπως για παράδειγμα τα κεφαλαία γράμματα ή αρκτικόλεξα λαμβάνονται υπόψη στον συγκεκριμένο αλγόριθμο. Σύμφωνα με τους παραπάνω ερευνητές, λέξεις με κεφαλαίο το πρώτο γράμμα τους -που δεν είναι «κοινότυπες»- υπάρχει πιθανότητα να ξεκινούν μια φράση οπότε και να προσδίδουν αυξημένο νοηματικό ενδιαφέρον στις άλλες λέξεις με τις οποίες συνυπάρχουν. Επίσης, τα αρκτικόλεξα π.χ. (N.A.S.A) υπάρχει μεγάλη πιθανότητα να αποτελούν λέξεις κλειδιά από μόνες τους. Τα τυπογραφικά χαρακτηριστικά ενός όρου (t) υπολογίζονται από την παρακάτω σχέση:

$$f_{case}(t) = \frac{\max(tf_{upper}(t), tf_{acronym}(t))}{\ln(tf(t))} \quad (8)$$

όπου tf_{upper} είναι η συχνότητα που ο όρος t εμφανίζεται να ξεκινά με κεφαλαία γράμματα, $tf_{acronym}$, η συχνότητα με την οποία ο όρος t εμφανίζεται με όλα τα γράμματα κεφαλαία και $tf(t)$, η συχνότητα εμφάνισης του όρου.

Η τελική βαθμολογία για έναν όρο (t) δίνεται από την παρακάτω σχέση, όπου ένας όρος θεωρείται σημαντικός όσο χαμηλότερη είναι η βαθμολογία του:

$$key_score(t) = \frac{f_{rel}(t) \cdot f_{pos}(t)}{f_{case}(t) + \frac{f_{norm}(t) + f_{sent}(t)}{f_{rel}(t)}} \quad (9)$$

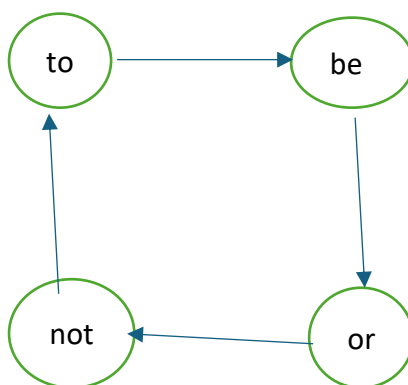
Επίσης, η βαθμολογία μιας φράσης (s), που περιέχει n όρους, δίνεται από την παρακάτω σχέση:

$$phrase_score(s) = \frac{\prod_{i=1}^n key_score(t_i)}{tf(s) \cdot (1 + \sum_{i=1}^n key_score(t_i))} \quad (10)$$

Όπου $tf(s)$ είναι η συχνότητα εμφάνισης της φράσης στο κείμενο. Όπως και στις λέξεις κλειδιά, έτσι και στις φράσεις κλειδιά, η χαμηλότερη βαθμολογία υποδηλώνει υψηλή σημαντικότητα.

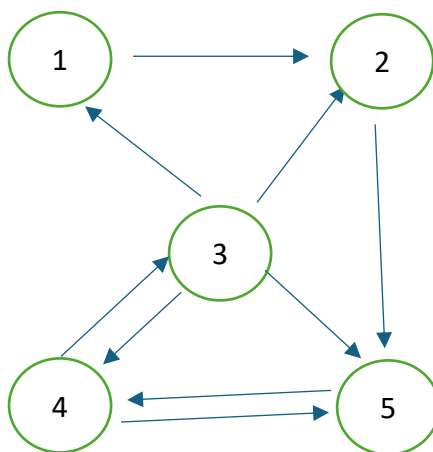
2.3.4 Μέθοδοι βασισμένες σε γράφους. Ο αλγόριθμος TEXTRANK

Σε αυτές τις μεθόδους όλο το έγγραφο θεωρείται ένας γράφος $G(V, E, W)$ που οι λέξεις είναι οι κόμβοι του (V) και οι συνδέσεις των λέξεων μεταξύ τους αποτελούν τις ακμές του γράφου (E). Επίσης, είναι δυνατόν οι ακμές να φέρουν βάρη (W). Ο γράφος μπορεί να είναι κατευθυνόμενος (directional) ή μη κατευθυνόμενος (unidirectional), ανάλογα με τις ανάγκες του αλγόριθμου που θα χρησιμοποιηθεί για την διαπέραση του. Για παράδειγμα, ο παρακάτω κατευθυνόμενος γράφος (**Σχήμα 2-1**), μπορεί να αποτελέσει αναπαράσταση της φράσης “To be or not to be”



Σχήμα 2-1 Αναπαράσταση του κειμένου “to be or not to be” σε γράφο

Ο αλγόριθμος TEXTRANK (Mihalcea and Tarau, 2004) βασίζεται στον αλγόριθμο PageRank της Google (Brin and Page, 1998). Ο TEXTRANK βρίσκει τις πιο σημαντικές ΛΦΚ ενός εγγράφου, με τον ίδιο αλγοριθμικό τρόπο που ο PageRank αναδεικνύει τις πιο σημαντικές ιστοσελίδες και τις παρουσιάζει στον χρήστη. Για να γίνει πιο κατανοητός ο τρόπος που λειτουργεί ο PageRank, ας θεωρήσουμε τον παρακάτω κατευθυνόμενο γράφο πέντε ιστοσελίδων (Σχήμα 2-2) όπου οι ακμές του γράφου δείχνουν την αναφορά κάποιας ιστοσελίδας σε κάποια άλλη. Επίσης, ο γράφος δεν λαμβάνει υπόψη τυχόν αυτοαναφορές των σελίδων, ενώ υπάρχει μόνο μια ακμή που συνδέει μια συγκεκριμένη σελίδα με μια άλλη, ανεξάρτητα του πόσες φορές αναφέρεται μια σελίδα στο περιεχόμενο μιας άλλης σελίδας.



Σχήμα 2-2 Αναπαράσταση σε γράφο 5 ιστοσελίδων

Ο αλγόριθμος βαθμολογεί τις σελίδες ανάλογα με την πιθανότητα που έχει ένας χρήστης να επισκεφθεί μια σελίδα εάν βρίσκεται σε κάποια άλλη σελίδα είτε άμεσα είτε μέσω κάποιας άλλης συνδεδεμένης με αυτή σελίδας. Στο παραπάνω παράδειγμα έστω ένας χρήστης βρίσκεται στην σελίδα 2, τότε από εκεί μπορεί να επισκεφθεί τόσο την σελίδα 5 όσο και την σελίδα 4 μέσω της σελίδας 5 κ.ο.κ. Συνεπώς, σελίδες που έχουν τις περισσότερες εισερχόμενες

ακμές θεωρούνται πιο σημαντικές αφού αυξάνεται η πιθανότητα να τις επισκεφθεί κάποιος χρήστης είτε άμεσα είτε μέσω ενδιάμεσων σελίδων που καταλήγουν σε αυτές. Ο αλγόριθμος ωστόσο, δίνει εσφαλμένα αποτελέσματα υψηλής βαθμολογίας σε κόμβους που δεν έχουν εξερχόμενες ακμές (αφού τα μονοπάτια των υπολοίπων κόμβων θα καταλήξουν σε αυτούς και εκεί θα εγκλωβιστούν). Η ύπαρξη τέτοιων κόμβων πρέπει να ανιχνεύεται πριν ξεκινήσει να τρέχει ο αλγόριθμος και σε αυτούς τους κόμβους προστίθενται εξερχόμενες ακμές προς όλους τους υπόλοιπους κόμβους του γράφου, θεωρώντας ότι από έναν τέτοιο κόμβο υπάρχει πιθανότητα $1/(σύνολο\ των\ κόμβων)$, να βρεθεί κανείς σε έναν άλλο κόμβο. Η βαθμολογία ενός κόμβου (p_i) σύμφωνα με τον αλγόριθμο PageRank υπολογίζεται από την παρακάτω σχέση:

$$PR(p_i) = \frac{1-d}{N} + d \cdot \sum_{p_j \in in(p_i)} \frac{PR(p_j)}{|out(p_j)|} \quad (11)$$

Όπου $in(p_i)$ είναι οι εισερχόμενες ακμές στον κόμβο p_i , $out(p_j)$ είναι οι εξερχόμενες ακμές του κόμβου p_j , $|out(p_j)|$ είναι το πλήθος των εξερχόμενων ακμών του κόμβου p_j , N το πλήθος των κόμβων του γράφου και d μια σταθερά. Η σταθερά d που έχει την τιμή 0,85 ονομάζεται *damping factor* και χρησιμεύει στην περίπτωση που υπάρχουν κόμβοι χωρίς εισερχόμενες ακμές ώστε να πάρουν μια βαθμολογία, αλλά και να επιλύει τα προβλήματα που δημιουργούν οι κόμβοι χωρίς εξερχόμενες ακμές επιτρέποντας στον αλγόριθμο να συγκλίνει..

Η υλοποίηση του αλγορίθμου PageRank γίνεται επαναληπτικά. Στην πρώτη επανάληψη η τιμή βαθμολογίας όλων των κόμβων αρχικοποιείται στην τιμή $1/N$. Ακολούθως και σε κάθε επανάληψη, υπολογίζεται εκ νέου η βαθμολογία του κάθε κόμβου σύμφωνα με την παραπάνω σχέση, ενώ η βαθμολογία των εισερχόμενων κόμβων είναι αυτή που υπολογίστηκε από την προηγούμενη επανάληψη. Ο αριθμός επαναλήψεων δεν είναι σταθερός και για αυτό τον λόγο συνήθως επιλέγεται ένας μεγάλος αριθμός επαναλήψεων (μεταξύ 20 και 100). Ο επαναληπτικός βρόχος του αλγορίθμου μπορεί να διακοπεί όταν ο αριθμός επαναλήψεων φτάσει την μέγιστη τιμή του ή και νωρίτερα στην περίπτωση που για όλους τους κόμβους η διαφορά βαθμολογίας τους μεταξύ δύο διαδοχικών επαναλήψεων είναι χαμηλότερη από μια πολύ μικρή τιμή (π.χ. 0,001), που την επιλέγει ο χρήστης ανάλογα με την ευαισθησία που επιθυμεί.

Ο TEXTRANK, ο οποίος στηρίζεται στην λογική του PageRank, αρχικά απομακρύνει τις «κοινότητες» λέξεις και ακολούθως κατασκευάζει τον γράφο του εγγράφου, συνδέοντας με ακμές τις λέξεις που βρίσκονται στην ίδια φράση. Ο γράφος είναι μη κατευθυνόμενος, συνεπώς

η κάθε λέξη της φράσης συνδέεται με ακμές με όλες τις υπόλοιπες λέξεις της φράσης. Ακολουθώντας εφαρμόζεται ο αλγόριθμος PageRank στον γράφο του εγγράφου, όπως αναφέρθηκε παραπάνω, και οι πιο σημαντικές λέξεις κλειδιά είναι εκείνες που λαμβάνουν την υψηλότερη βαθμολογία. Μια παραλλαγή του TextRank, έχει προταθεί από τους Pan et al, (2019) όπου η βαθμολογία ενός κόμβου p_i δίδεται από την παρακάτω σχέση:

$$TR(p_i) = \frac{1-d}{N} + d \cdot \sum_{p_j \in in(p_i)} \frac{TR(p_j) \cdot w(p_j)}{|out(p_j)|} \quad (12)$$

Όπου $w(p_j)$ είναι ένα βάρος (weight) του όρου p_j , όπως για παράδειγμα η τιμή TF-IDF.

Με την παραπάνω τεχνική, ο TEXTRANK βαθμολογεί μόνο τα unigrams ενός εγγράφου αφού μόνο εκείνα συμμετέχουν στην κατασκευή του γράφου του εγγράφου. Ωστόσο οι Rajput et al (2020), πρότειναν μια υλοποίηση του TEXTRANK όπου ο γράφος του εγγράφου εκτός από τα unigrams να περιέχει και τα n-grams (που σχηματίζονται από τα unigrams) ως ξεχωριστοί κόμβοι στον γράφο. Στην υλοποίηση που πρότειναν οι Rajput et al (2020), παρατήρησαν ότι η βαθμολογία των n-grams ήταν κατά πολύ χαμηλότερη εκείνης των αντίστοιχων unigrams και μετά από διάφορες δοκιμές πρότειναν για την βαθμολογία των bigrams το τετραπλάσιο της τιμής που τους βαθμολογεί ο TEXTRANK και για τα trigrams και άνω το εξαπλάσιο της τιμής που τους βαθμολογεί ο TEXTRANK.

2.3.5 Ο αλγόριθμος KEYBERT

Ο KEYBERT (Grootendorst, 2020) αποτελεί μια μη επιβλεπόμενη μέθοδο εξαγωγής λέξεων-κλειδιών που αξιοποιεί σημασιολογικές αναπαραστάσεις (embeddings) από μοντέλα που αξιοποιούν την αρχιτεκτονική Transformer (Vaswani et al., 2017). Σύμφωνα με την αρχιτεκτονική Transformer, το μοντέλο επιτρέπει την 'παράλληλη' επεξεργασία ολόκληρης της ακολουθίας εισόδου ενός εγγράφου (π.χ. παράγραφοι, σύνολο παραγράφων κλπ) ώστε να μπορεί να υπολογίζει τη σχετική σημασία κάθε λέξης σε σχέση με όλες τις υπόλοιπες λέξεις μέσα στο ίδιο κείμενο και να αναδεικνύει συντακτικές και σημασιολογικές εξαρτήσεις μεταξύ των λέξεων (Vaswani et al., 2017). Ο KEYBERT, κωδικοποιεί τόσο ολόκληρο το έγγραφο όσο και ένα σύνολο υποψήφιων ΛΦΚ (συνήθως n-grams) σε έναν κοινό διανυσματικό χώρο, χρησιμοποιώντας προεκπαιδευμένα Transformer μοντέλα όπως το Sentence-BERT (Reimers & Gurevych, 2019, Wang et. al, 2020). Η συνάφεια κάθε υποψήφιας ΛΦΚ υπολογίζεται μέσω της ομοιότητας συνημίτονου του διανύσματος του εγγράφου και των διανυσμάτων των

υποψηφίων ΛΦΚ, επιλέγοντας εκείνες με τη μεγαλύτερη σημασιολογική εγγύτητα (Grootendorst, 2020). Σε αντίθεση με τις μεθόδους που αναφέρθηκαν στις προηγούμενες υποενότητες ο KEYBERT βασίζεται σε συμφραζόμενες αναπαραστάσεις, επιτρέποντας την αναγνώριση εννοιολογικά συναφών όρων ακόμη και όταν δεν εμφανίζονται με υψηλή συχνότητα.

2.4 Αναπαράσταση εγγράφων

2.4.1 Αναπαράσταση εγγράφου – «σάκος λέξεων» (Bag of Words)

Η απλούστερη μορφή που μπορεί να αναπαρασταθεί ένα έγγραφο είναι μέσω των όρων που περιέχει και τον αριθμό εμφάνισης του κάθε όρου σε αυτό. Για παράδειγμα, το κείμενο “*this and that make this and the other*”, αναπαρίσταται ως εξής:

$$\{("this", 2), ("and", 2), ("that", 1), ("make", 1), ("the", 1), ("other", 1)\}$$

Η συγκεκριμένη αναπαράσταση που θυμίζει ένα «σάκο» που περιέχει όλες τις λέξεις ονομάζεται στην βιβλιογραφία (Manning et al, 2008), ως bag of words (BoW). Σε αυτή την αναπαράσταση, την θέση του αριθμού εμφανίσεων του όρου θα μπορούσε να πάρει κάποια άλλη μετρική σχετικά με τον όρο (π.χ συχνότητα εμφάνισης ή κάποια βαθμολογία ως λέξη κλειδί) ή να προσαρτηθούν περισσότερες από μια μετρικές για τον ίδιο όρο σύμφωνα με την γενικευμένη μορφή («όρος», μετρική₁, μετρική₂, ...). Επίσης, είναι πολύ σημαντικό για την παραπάνω συλλογή να υπάρχει ένας δείκτης (αναγνωριστικός κωδικός εγγράφου) που να δείχνει σε αυτή.

2.4.2 Αναπαράσταση μιας συλλογής εγγράφων

Ο τρόπος με τον οποίο θα αναπαρασταθεί μια συλλογή εγγράφων σε σχέση με τους όρους που περιέχουν, έχει πολύ μεγάλη σημασία προκειμένου η ΑΠ να είναι όσο το δυνατόν πιο αποδοτική τόσο από άποψη χρόνου όσο και από άποψη χώρου. Έστω μια συλλογή n εγγράφων με $\{D_1, D_2, \dots, D_n\}$ στα οποία εμπεριέχονται οι m όροι $\{t_1, t_2, \dots, t_m\}$, παρακάτω θα δούμε τους πιθανούς τρόπους με τους οποίους μπορεί να αναπαρασταθεί η συλλογή.

2.4.2.1 Μήτρες όρος – έγγραφο / έγγραφο όρος (term document matrix / document term matrix)

Η μήτρα όρος-έγγραφο έχει την μορφή ενός πίνακα $M^{m \times n}$, όπως παρακάτω:

$$M^{m \times n} = \begin{cases} M[k][j] = 0, & \text{ο όρος } t_k \nexists \text{ στο έγγραφο } D_j \\ M[k][j] = 1, & \text{ο όρος } t_k \exists \text{ στο έγγραφο } D_j \end{cases}$$

με $1 \leq k \leq m$ και $1 \leq j \leq n$

Μια σχηματική παράσταση της μήτρας όρος-έγγραφο φαίνεται στον Πίνακα 2-2

	D ₁	D ₂	...	D _j		D _n
t ₁	1	0		0	0/1	1
t ₂	0	1		0	0/1	1
t ₃	1	1		0	0/1	1
					0/1	
t _k	1	1		1	0/1	0
					0/1	
t _n	0	1		1	0/1	0

Πίνακας 2-2 Σχηματική αναπαράσταση της μήτρας όρος (t)-έγγραφο σε μια συλλογή εγγράφων (D)

Όπως φαίνεται και από την σχηματική αναπαράσταση, η μήτρα όρος-έγγραφο είναι πολύ αραιά, δηλαδή περιέχει πολύ περισσότερα 0 σε σχέση με το 1, και βέβαια καταλαμβάνει πολύ χώρο στην μνήμη. Ωστόσο, είναι πολύ αποδοτική από άποψη χρόνου σε περιπτώσεις δυαδικής ανάκτησης πληροφορίας όπως θα εξηγηθεί στην ενότητα της ΑΠ.

Η μήτρα έγγραφο – όρος προκύπτει από την αναστροφή της μήτρας όρος – έγγραφο. Στην μήτρα έγγραφο-όρος κάθε έγγραφο της συλλογής αναπαρίσταται από έναν εκτεταμένο σάκο λέξεων, με μηδενικές τιμές στα κελιά των λέξεων που δεν περιέχονται στο έγγραφο. Αν και η μήτρα έγγραφο όρος είναι επίσης μια αραιά δομή, ωστόσο χρησιμοποιείται στο θεωρητικό μοντέλο της ΑΠ διανυσματικού χώρου, όπως θα εξηγηθεί σε επόμενες ενότητες.

2.4.2.2 Αντεστραμμένος δείκτης (inverted index)

Το μειονέκτημα, σε απαιτήσεις χώρου, της μήτρας όρος-έγγραφο, μπορεί να ξεπεραστεί με την αναπαράσταση της συλλογής εγγράφων ως ένα λεξικό που κάθε όρος του θα δείχνει σε μια συνδεδεμένη λίστα (ή κάποια άλλη συλλογή) που θα αποτελείται από το id των εγγράφων που περιέχουν αυτόν τον όρο. Έτσι από την αραιή μήτρα όρος-έγγραφο αφαιρούνται όλες οι μηδενικές τιμές κελιών και καταλήγουμε στην αναπαράσταση του αντεστραμμένου δείκτη όπως φαίνεται στο παρακάτω παράδειγμα 4 όρων (t) και 5 εγγράφων (D):

t₁ : D₁, D₂, D₄

t_2 : D_2, D_4

t_3 : D_1, D_3, D_5

t_4 : D_3, D_4

Με τον αντεστραμμένο δείκτη μπορούμε να γνωρίζουμε τον αριθμό των εγγράφων που εμφανίζεται ένας όρος στην συλλογή. Επίσης, εάν αντί για το id του εγγράφου, ο κάθε όρος έδειχνε σε ένα tuple της μορφής ($doc_id, count$), όπου $count$ ο αριθμός εμφανίσεων ενός όρου σε ένα έγγραφο, τότε θα είχαμε όλη την απαιτούμενη πληροφορία για να υπολογίσουμε την τιμή TF-IDF για έναν συγκεκριμένο όρο σε ολόκληρη την συλλογή και να τον κατατάξουμε ανά σειρά σημαντικότητας (ανάκτηση με κατάταξη ή ranked retrieval).

2.5 Ανάκτηση πληροφορίας – Information Retrieval

2.5.1 Γενικό Θεωρητικό Μοντέλο

Στην συγκεκριμένη υποενότητα θα ασχοληθούμε με την ΑΠ και τα μοντέλα προκειμένου να ανακτηθεί η πληροφορία μέσω αναζήτησης της σε μία συλλογή εγγράφων. Σύμφωνα με τους Manning et al (2008), το γενικό θεωρητικό μοντέλο της ΑΠ χωρίζεται στις παρακάτω κατηγορίες:

- (α) Boolean
- (β) Διανυσματικού Χώρου (Vector space)
- (γ) Πιθανοτική (Probabilistic)

2.5.2 Boolean Retrieval (Boolean ΑΠ)

Η Boolean ΑΠ χρησιμοποιεί τους λογικούς τελεστές "AND", "OR" και "NOT" για να φιλτράρει τη συλλογή και να επιστρέψει τα έγγραφα της συλλογής, που ικανοποιούν το αποτέλεσμα κάποιας λογικής πράξης (π.χ. περιέχουν έναν όρο ή δεν περιέχουν ένα όρο κλπ). Σύμφωνα με τους Manning et al (2008), η λίστα των λογικών τελεστών μπορεί να επεκταθεί ώστε να συμπεριλαμβάνει και τελεστές τοποθεσίας εντός του εγγράφου, όπως για παράδειγμα ο τελεστής "NEAR", που επιστρέφει κάποιο αποτέλεσμα σε περίπτωση που μια λέξη είναι x λέξεις κοντά σε μια άλλη. Οι τοπικοί τελεστές ωστόσο, προϋποθέτουν να είναι γνωστός ο αριθμός θέσεως της κάθε λέξης στο έγγραφο.

Η Boolean ΑΠ είναι ιδανική σε περιπτώσεις που ο χρήστης επιθυμεί να λάβει όλα τα έγγραφα που είναι σχετικά με το ερώτημα του. Ωστόσο, η σειρά των ανακτημένων εγγράφων είναι αυθαίρετη και για αυτό το μοντέλο αυτό απευθύνεται σε περιπτώσεις που δεν είναι

αναγκαία η κατάταξη των εγγράφων κατά σειρά σχετικότητας (**Manning et al 2008**). Η μέθοδος μειονεκτεί σε περιπτώσεις που απαιτείται γρήγορη ανάκτηση πληροφορίας, καθώς ανάλογα με το ερώτημα του χρήστη, το εύρος των ανακτημένων εγγράφων μπορεί να είναι είτε πολύ ευρύ (ο χρήστης πρέπει να ψάξει έναν μεγάλο όγκο ανακτημένων εγγράφων για να βρει το πλέον κατάλληλο) ή πολύ στενό (ενδεχομένως το ερώτημα να πρέπει να επαναδιατυπωθεί καλύτερα).

Το μοντέλο της Boolean ΑΠ, μπορεί να καλύψει και περιπτώσεις που ο όροι του ερωτήματος βρίσκονται σε πεδία (sections ή fields) του εγγράφου, με διαφορετικό βαθμό βαρύτητας μεταξύ τους (weights). Το συγκεκριμένο μοντέλο ονομάζεται Εκτεταμένο Μοντέλο Boolean ΑΠ επεκταθεί (extended Boolean model ή ranked Boolean retrieval) και προϋποθέτει ότι τα έγγραφα θα έχουν αναπαρασταθεί στο μοντέλο ως πεδία. Η σχεδίαση της αναπαράστασης του μοντέλου μπορεί να γίνει με διάφορους τρόπους. Μια σχεδίαση είναι να γίνει ανά πεδίο, ενώ μια άλλη σχεδίαση είναι να γίνει ανά έγγραφο όπου σε κάθε έγγραφο να υπάρχουν τα διαφορετικά του πεδία. Για παράδειγμα, έστω ότι είναι επιθυμητό να αναζητήσει κανείς από μία βάση ερευνητικών εργασιών τις δημοσιευμένες εργασίες ενός ερευνητή, με βάση το επώνυμο του. Το επώνυμο του ερευνητή εκτός από το πεδίο “authors”, μπορεί να βρίσκεται επίσης στο πεδίο “main_text”, όταν γίνεται κάποια αναφορά σε αυτόν ή ακόμα και στο πεδίο “references” σε περίπτωση αυτό- ή ετερο-αναφοράς. Για την ικανοποίηση του συγκεκριμένου ερωτήματος θα μπορούσε ο βαθμός σημαντικότητας (weight) μεταξύ των πεδίων να είναι διαφορετικός ανά πεδίο (π.χ. authors=0,6 / main_text=0.2 / references = 0,2), ώστε το σύνολο των βαρών να είναι ίσο με 1. Για κάθε έγγραφο το αποτέλεσμα του συγκεκριμένου ερωτήματος υπολογίζεται από την παρακάτω σχέση:

$$\sum_{i=1}^n p_i \cdot w_i \quad (13)$$

Όπου n , ο αριθμός πεδίων, p_i το αποτέλεσμα της λογικής πράξης στο πεδίο i και w_i , ο βαθμός σημαντικότητας του πεδίου i . Ακολουθώντας, τα έγγραφα κατατάσσονται ανάλογα με την βαθμολογία και το αποτέλεσμα παρουσιάζεται στον χρήστη. Η κατάταξη των πεδίων ανά βαθμό σημαντικότητας εξαρτάται από το ερώτημα. Για παράδειγμα εάν το πεδίο “references” είχε υψηλότερη βαθμολογία σε σχέση με το πεδίο “authors”, η ταξινομημένη λίστα εγγράφων θα μπορούσε να επιστρέψει την απάντηση σε ερώτημα σχετικά με τις ετερο-αναφορές του ερευνητή. Η τιμή του βαθμού σημαντικότητας καθορίζεται από τον χρήστη αν και σύμφωνα

με τους Manning et al, (2008) μπορεί να προσαρμοστεί και μέσω μοντέλων μηχανικής μάθησης.

2.5.3 ΑΠ διανυσματικού χώρου (Vector Space Model)

Σε αυτό το μοντέλο, η συλλογή εγγράφων είναι ένας κοινός διανυσματικός χώρος (vector space) που αποτελείται από έγγραφα με την μορφή διανυσμάτων (Manning et al, 2008). Σύμφωνα με το μοντέλο, η διανυσματική μορφή ενός εγγράφου d που περιέχει n όρους (λήμματα ή λέξεις) είναι η παρακάτω:

$$\overrightarrow{V_{(d)}} = [x_1, x_2, x_3 \dots, x_n]$$

όπου x_i ($1 \leq i \leq n$), ένα χαρακτηριστικό του όρου i στο εν λόγω έγγραφο (π.χ. συχνότητα εμφάνισης του όρου) και απαιτείται για την ανάκτηση πληροφορίας από το έγγραφο σχετικά με τον συγκεκριμένο όρο. Σε μια πιο γενικευμένη μορφή αναπαράστασης του σάκου των λέξεων ενός εγγράφου, η τιμή του n είναι όλο το λεξιλόγιο της συλλογής, με τιμές του χαρακτηριστικού $x_i = 0$ για τους όρους που δεν περιέχονται στο έγγραφο.

Η διανυσματική μορφή του μοντέλου επιτρέπει, μέσω της εκτέλεσης αλγεβρικών πράξεων, να είναι δυνατός ο υπολογισμός του βαθμού ομοιότητας μεταξύ δύο ή περισσότερων εγγράφων μεταξύ τους ή μεταξύ ενός ερωτήματος και ενός εγγράφου. Ο υπολογισμός του βαθμού ομοιότητας μεταξύ εγγράφων, είναι χρήσιμος όταν είναι επιθυμητή η ταξινόμηση ή δημιουργία ομάδων των εγγράφων ως προς ένα κοινό χαρακτηριστικό, ενώ ο βαθμός ομοιότητας ως προς ένα ερώτημα, είναι επιθυμητός για την κατάταξη της συλλογής ως προς το βαθμό σχετικότητας με το ερώτημα. Είναι εμφανές ότι ενώ το Boolean ΑΠ μοντέλο ανακτά πληροφορία ως 1/0, το μοντέλο διανυσματικού χώρου ανακτά την πληροφορία με σειρά κατάταξης (ranked retrieval) ανάλογα με ένα χαρακτηριστικό.

Διάνυσμα ερωτήματος – Υπολογισμός ομοιότητας

Η διαφορά της Boolean ΑΠ σε σχέση με την ΑΠ διανυσματικού χώρου, είναι ότι στην τελευταία τα ερωτήματα μπορούν να είναι και εκείνα με την μορφή κειμένου. Συνεπώς, ένα ερώτημα q μπορεί να αναπαρασταθεί ως σάκος λέξεων, με την παρακάτω διανυσματική μορφή:

$$\overrightarrow{V_{(q)}} = [y_1, y_2, y_3 \dots, y_n]$$

όπου $y_i \geq 1$, εάν ο όρος i , υπάρχει τουλάχιστον μια φορά στο ερώτημα ή $y_i = 0$, εάν όρος i , δεν υπάρχει στο ερώτημα. Για τον υπολογισμό του βαθμού ομοιότητας μεταξύ δύο

διανυσμάτων, είτε είναι έγγραφα μεταξύ τους είτε είναι έγγραφα σε σχέση με ένα ερώτημα υπολογίζεται το συνημίτονο της γωνίας $\hat{\theta}$ που σχηματίζουν μεταξύ τους. Εάν συμπίπτουν τότε $\hat{\theta} = 0^\circ$ και $\cos(\hat{\theta}) = 1$, ενώ εάν δεν συμπίπτουν καθόλου τότε $\hat{\theta} = 90^\circ$ και $\cos(\hat{\theta}) = 0$. Το συνημίτονο της γωνίας $\hat{\theta}$ μεταξύ των δύο διανυσμάτων $\vec{V}_{(d)} = [x_1, x_2, x_3 \dots, x_n]$ και $\vec{V}_{(q)} = [y_1, y_2, y_3 \dots, y_n]$ δίνεται από τον παρακάτω τύπο:

$$\cos(\hat{\theta}) = \frac{\vec{V}_{(d)} \cdot \vec{V}_{(q)}}{|\vec{V}_{(d)}| |\vec{V}_{(q)}|} \quad (14)$$

όπου $\vec{V}_{(d)} \cdot \vec{V}_{(q)}$, είναι το εσωτερικό γινόμενο των δύο διανυσμάτων και ίσο με

$$\vec{V}_{(d)} \cdot \vec{V}_{(q)} = \sum x_i y_i \quad (15)$$

και $|\vec{V}_{(d)}|$, $|\vec{V}_{(q)}|$ τα μέτρα των διανυσμάτων που υπολογίζονται ως ακολουθεί:

$$\begin{aligned} |\vec{V}_{(d)}| &= \sqrt{\sum x_i^2} \\ |\vec{V}_{(q)}| &= \sqrt{\sum y_i^2} \end{aligned} \quad (16)$$

2.5.4 Πιθανοθεωρητική ΑΠ (Probabilistic Model)

Η πιθανοθεωρητική ανάκτηση πληροφορίας (Probabilistic Information Retrieval, PIR) αποτελεί ένα θεμελιώδες πλαίσιο για την αξιολόγηση της σχετικότητας όρων ή εγγράφων με ένα ερώτημα, βασιζόμενο στην εκτίμηση πιθανοτήτων. Σύμφωνα με τους Manning et al, (2008), η σχετικότητα εκφράζεται συνήθως ως

$$P(\text{relevance} | \text{document}, \text{query})$$

δηλαδή η πιθανότητα ένα έγγραφο ή όρος να είναι σχετικός δεδομένου του περιεχομένου και του ερωτήματος. Ένα από τα πιο γνωστά παραδείγματα μη επιβλεπόμενου πιθανοθεωρητικού μοντέλου ΑΠ είναι η συνάρτηση BM25 (Spärck et al, 2000, Robertson and Zaragoza, 2009), όπου η σημαντικότητα των όρων υπολογίζεται από στατιστικές παρατηρήσεις σχετικά με την εμφάνιση όρων σε σχετικά και μη σχετικά έγγραφα, χωρίς να απαιτούνται επισημασμένα δεδομένα. Αντίθετα, στις επιβλεπόμενες πιθανοθεωρητικές προσεγγίσεις, όπως οι Naive Bayes classifier (Friedman and Goldszmidt, 1996) ή νευρωνικά μοντέλα που προσαρμόζονται σε επισημασμένα δεδομένα, οι πιθανότητες σχετικότητας εκτιμώνται άμεσα από επισημασμένα

σύνολα δεδομένων, επιτρέποντας την πρόβλεψη της σημασίας νέων όρων ή εγγράφων με βάση προηγούμενη γνώση. Η διάσταση της πιθανοθεωρητικής μοντελοποίησης είναι ανεξάρτητη από τον τρόπο μάθησης: ένα μοντέλο μπορεί να είναι βασισμένο σε στατιστικές υποθέσεις και κανόνες (π.χ. BM25), ή βασισμένο σε εκπαίδευση από επισημασμένα δεδομένα (π.χ. Naive Bayes). Επιπλέον, η πιθανοθεωρητική προσέγγιση προσφέρει ένα ενιαίο πλαίσιο για τη βαθμολόγηση υποψήφιων ΛΦΚ είτε μέσω στατιστικών μεθόδων, είτε μέσω μοντέλων εκμάθησης, διευκολύνοντας τη σύνδεση παραδοσιακών μεθόδων ανάκτησης πληροφορίας με σύγχρονες τεχνικές εξαγωγής λέξεων-κλειδιών.

2.6 Αξιολόγηση αποτελεσματικότητας συστήματος ΑΠ

2.6.1 Γενικά – Μήτρα σύγχυσης (confusion matrix)

Η αξιολόγηση της αποτελεσματικότητας του μοντέλου της ΑΠ ή κάποιου αλγόριθμου για εξαγωγή ΛΦΚ από κάποιο έγγραφο μπορεί να γίνει με τις παρακάτω προσεγγίσεις (Manning et al, 2008):

- Χωρίς ανατροφοδότηση από τον χρήστη
- Με ανατροφοδότηση από τον χρήστη

Στην πρώτη περίπτωση το μοντέλο εφαρμόζεται σε συλλογές εγγράφων, όπου υπάρχουν έτοιμοι συνδυασμοί ερωτημάτων και απαντήσεων τους (ανακτημένα έγγραφα της συλλογής). Οι συγκεκριμένες συλλογές περιέχουν έναν μεγάλο αριθμό εγγράφων και επίσης έναν μεγάλο αριθμό συνδυασμών ερωτήσεων/ανακτημένων εγγράφων και με την βοήθεια κάποιων μετρικών που θα παρουσιαστούν παρακάτω, μετριέται η αποδοτικότητα του μοντέλου, ως προς το να προβλέπει σωστά τα σχετικά έγγραφα ανά ερώτημα. Στην δεύτερη περίπτωση (relevance feedback), το μοντέλο δοκιμάζεται από τον ίδιο τον χρήστη σε πραγματικές συνθήκες, και ο χρήστης μέσω ανατροφοδότησης «εκπαιδεύει» κατάλληλα το μοντέλο ανάλογα με το εάν προέβλεψε σωστά τα ανακτημένα έγγραφα ή όχι.

Ένα μοντέλο ΑΠ ανακτά έναν αριθμό εγγράφων από μια συλλογή εγγράφων ως απάντηση σε ένα ερώτημα. Το μοντέλο θεωρεί ότι όλα τα ανακτημένα έγγραφα είναι σχετικά ως προς το ερώτημα ενώ τα υπόλοιπα έγγραφα της συλλογής (μη ανακτημένα) ότι δεν είναι σχετικά με το εν λόγω ερώτημα. Ένα ανακτημένο έγγραφο, μπορεί να είναι πράγματι σχετικό με το ερώτημα (True Positive ή TP), μπορεί ωστόσο να μην είναι πράγματι σχετικό (False Positive ή FP) με το ερώτημα. Κατά αντιστοιχία, ένα μη ανακτημένο έγγραφο μπορεί πράγματι να μην είναι σχετικό με το ερώτημα (True Negative ή TN), μπορεί ωστόσο να είναι σχετικό με

το ερώτημα (False Negative ή FN). Οι παραπάνω περιπτώσεις συνοψίζονται σε έναν πίνακα (Tan et al, 2018), ο οποίος ονομάζεται μήτρα σύγχυσης (confusion matrix), ως εξής (Πίνακας 2-3):

		Πραγματικότητα	
		Σχετικά Έγγραφα	Μη Σχετικά Έγγραφα
Εκτίμηση	Σχετικά Έγγραφα	TP	FP
	Μη Σχετικά Έγγραφα	FN	TN

Πίνακας 2-3 Μήτρα σύγχυσης

Στην παρούσα ΔΕ, θα ασχοληθούμε μόνο με την πρώτη προσέγγιση και για αυτό το λόγο θα παρουσιαστούν οι μετρικές της. Οι μετρικές που θα παρουσιαστούν παρακάτω θα αναφέρονται μόνο με την αγγλική τους ορολογία, λόγω ομοιομορφίας με την βιβλιογραφία.

2.6.2 Ανάκτηση χωρίς κατάταξη

Σε αυτή την περίπτωση το μοντέλο της ΑΠ ανακτά τα σχετικά έγγραφα ως προς ένα ερώτημα, χωρίς να τα κατατάσσει ως προς τον βαθμό σχετικότητας με το ερώτημα. Σε αυτές τις περιπτώσεις ανήκει η Boolean ΑΠ (χωρίς την υποπερίπτωση της κατάταξης των όρων ανάλογα το βάρος τους), καθώς επίσης και περιπτώσεις ομαδοποίησης εγγράφων (classification / clustering). Οι σημαντικότερες μετρικές αυτής της περίπτωσης είναι οι παρακάτω:

Precision (P)

Η μετρική P δίνεται από τον παρακάτω τύπο και αντιπροσωπεύει το ποσοστό των πραγματικά σχετικών εγγράφων επί του συνόλου των ανακτημένων εγγράφων που επέστρεψε το μοντέλο:

$$P = \frac{\text{αριθμός σχετικών εγγράφων που ανακτήθηκαν}}{\text{συνολικός αριθμός ανακτημένων εγγράφων}} = \frac{TP}{TP + FP} \quad (17)$$

Recall (R)

Η μετρική R δίνεται από τον παρακάτω τύπο και αντιπροσωπεύει το ποσοστό των εγγράφων που επέστρεψε το μοντέλο και είναι πράγματι σχετικά ως ένα ερώτημα:

$$R = \frac{\text{αριθμός σχετικών εγγράφων που ανακτήθηκαν}}{\text{συνολικός αριθμός σχετικών εγγράφων}} = \frac{TP}{TP + FN} \quad (18)$$

F measure (F)

Οι μετρικές P και R σε πολλές περιπτώσεις αλληλοσυγκρούονται μεταξύ τους και για τον λόγο αυτό η κάθε μια έχει την δική της χρησιμότητα. Για παράδειγμα, ένας τυπικός χρήστης του διαδικτύου θα προτιμούσε όλοι οι σχετικοί υπερσύνδεσμοι να βρίσκονται στην πρώτη σελίδα (υψηλό P), χωρίς να τον ενδιαφέρει εάν οι υπερσύνδεσμοι των επόμενων σελίδων είναι σχετικοί με το ερώτημα του. Αντιθέτως, ένας επιστήμονας που επιθυμεί μια πιο ολοκληρωμένη αναζήτηση ώστε να αναζητήσει όσο το δυνατόν περισσότερη πληροφορία διαθέσιμη στον παγκόσμιο ιστό, τότε θα ενδιαφερόταν για υψηλότερα επίπεδα R προκειμένου να μην «χάσει» κάποια πληροφορία. Προκειμένου, να αξιολογηθεί ένα μοντέλο με μία συμβιβαστική τιμή μεταξύ των P και R χρησιμοποιείται ο αρμονικός τους όρος (μετρική F) ως εξής:

$$F = \frac{2PR}{P + R} \quad (19)$$

2.6.3 Ανάκτηση με κατάταξη

Σε αντίθεση με την ανάκτηση χωρίς κατάταξη, που τα έγγραφα που επιστρέφει το μοντέλο κατηγοριοποιούνται δυαδικά ως προς ένα ερώτημα (σχετικό/μη σχετικό), στην ανάκτηση με κατάταξη τα έγγραφα που επιστρέφει το μοντέλο κατατάσσονται με φθίνουσα σειρά ανάλογα τον βαθμό σχετικότητας ως προς ένα ερώτημα. Στην ανάκτηση με κατάταξη κυρίως μας ενδιαφέρουν τα πρώτα k έγγραφα της κατάταξης, παρά το γεγονός ότι και τα επόμενα $k+n$ έγγραφα θα μπορούσαν να είναι σχετικά ως προς το ερώτημα (αλλά οπωσδήποτε λιγότερο σχετικά από τα πρώτα k). Η τιμή του k ορίζεται από τον χρήστη προκειμένου να ικανοποιεί τις ανάγκες αναζήτησης του.

Οι σημαντικότερες μετρικές αυτού του μοντέλου είναι οι παρακάτω (Valcarce et al, 2020):

Precision@k (P@k), Recall@k (R@k)

Οι μετρικές $P@k$ και $R@k$ είναι οι τιμές του P και R , αντίστοιχα για τα πρώτα k έγγραφα που επιστρέφει το μοντέλο και υπολογίζονται από τους παρακάτω τύπους:

$$P@k = \frac{\text{αριθμός σχετικών εγγράφων στα πρώτα } k \text{ αποτελέσματα}}{k} \quad (20)$$

$$R@k = \frac{\text{αριθμός σχετικών εγγράφων στα πρώτα } k \text{ αποτελέσματα}}{\text{συνολικός αριθμός σχετικών εγγράφων}} \quad (21)$$

Average Precision (AP@k) – Mean Average Precision (MAP@k)

Το μεινέκτημα της $P@k$ είναι ότι δεν εξετάζει εάν τα σχετικά έγγραφα επιστράφηκαν νωρίς ή αργά από το μοντέλο. Για παράδειγμα έστω ότι σε μια συλλογή υπάρχουν n σχετικά έγγραφα και το μοντέλο στην ιδανική περίπτωση τα επιστρέφει όλα στις θέσεις από 1 έως n . Σε αυτήν την περίπτωση η τιμή της $P@k$ (με $k > n$) λαμβάνει την τιμή $(k - n / k)$. Παρόλα αυτά, η $R@k$ θα λάβει επίσης την ίδια τιμή ακόμα και στην χειρίστη περίπτωση που τα n έγγραφα επιστραφούν τελευταία από το μοντέλο και καταλάβουν τις θέσεις $k-n$ έως k . Προκειμένου να ξεπεραστεί αυτό το πρόβλημα προτείνεται αντί της $P@k$ να χρησιμοποιείται η μέση τιμή της (Average Precision, $AP@k$) για τιμές από 1 έως k ως εξής:

$$AP@k = \frac{1}{k} \sum_{i=1}^k P@i \quad (22)$$

Για q αριθμό ερωτημάτων, η μέση τιμή της $AP@k$ του κάθε ερωτήματος είναι η μετρική $MAP@k$, που δίδεται από τον παρακάτω τύπο:

$$MAP@k = \frac{1}{q} \sum_{i=1}^q AP_i@k \quad (23)$$

Reciprocal Rank (RR) – Mean Reciprocal Rank (MRR)

Η μετρική RR δείχνει πόσο μακριά από την κορυφή, επιστρέφεται το πρώτο σχετικό έγγραφο από το μοντέλο. Η εν λόγω μετρική χρησιμοποιείται κυρίως όταν ο χρήστης ενδιαφέρεται για την υψηλότερη θέση κατάταξης που θα λάβει μια σχετική απάντηση σε ένα ερώτημα. Η RR παίρνει τιμές μεταξύ 0 και 1, και δίδεται από τον παρακάτω τύπο:

$$RR = \frac{1}{\text{αριθμός κατάταξης του πρώτου σχετικού εγγράφου}} \quad (24)$$

Η μέση τιμή της RR για έναν αριθμό ερωτημάτων q είναι η μετρική Mean Reciprocal Rank (MRR) που δίδεται από τον παρακάτω τύπο:

$$MRR = \frac{1}{q} \sum_{i=1}^q RR_i \quad (25)$$

Discounted Cumulative Gain (DCG) – Normalized Discounted Cumulative Gain (nDCG)

Με την μετρική *DCG*, μπορεί κανείς να αξιολογήσει εάν το μοντέλο ΑΠ κατατάσσει τα πιο σχετικά έγγραφα νωρίτερα σε σχέση με τα λιγότερο σχετικά. Η συγκεκριμένη μετρική προϋποθέτει την ύπαρξη μιας βαθμονομημένης παραμέτρου (*rel*) που «μετρά» τον βαθμό σχετικότητας. Για παράδειγμα, μια βαθμονόμηση με *rel* από 0 έως και 3 είναι η παρακάτω:

‘υψηλή σχετικότητα’ → *rel*=3

‘μέτρια σχετικότητα’ → *rel*=2

‘μικρή σχετικότητα’ → *rel*=1

‘άσχετο’ → *rel*=0

Για ένα έγγραφο που επιστρέφεται από το μοντέλο ΑΠ, το *rel* αντιστοιχεί στο κέρδος (*gain*) του χρήστη (υψηλή σχετικότητα αντιστοιχεί σε μεγάλο κέρδος ενώ άσχετο έγγραφο αντιστοιχεί σε μηδενικό κέρδος). Σύμφωνα με την λογική της μετρικής *DCG* θα πρέπει να εφαρμόζεται ένα είδος αναλογικής ποινής στο κέρδος (*discounted gain*) όσο πιο χαμηλά στην λίστα εμφανίζονται τα σχετικά έγγραφα. Εάν *k* ο αριθμός εγγράφων που επιστρέφει ένα μοντέλο, η τιμή της μετρικής *DCG* για το μοντέλο δίδεται από τον παρακάτω τύπο:

$$DCG_k = \sum_{i=1}^k \frac{rel_i}{\log_2(i+1)} \quad (26)$$

όπου *rel_i* είναι η βαθμονόμηση της σχετικότητας ενός εγγράφου που επιστράφηκε από το μοντέλο στην θέση *i*. Επειδή η τιμή της *DCG* εξαρτάται από τον αριθμό των εγγράφων που επιστρέφεται από ερώτημα, προτείνεται να χρησιμοποιείται η κανονικοποιημένη μορφή της (*Normalized Discounted Cumulative Gain - nDCG*) που δίδεται από την παρακάτω σχέση:

$$nDCG_k = \frac{DCG_k}{IDCG} \quad (27)$$

Το *IDCG* αποτελεί την ιδανική τιμή του *DCG* (*Ideal DCG*), δηλαδή την περίπτωση που το μοντέλο επιστρέφει τα πιο σημαντικά έγγραφα υψηλότερα στην λίστα σε σχέση με τα λιγότερο

σημαντικά. Για το παράδειγμα που η τιμή rel παίρνει τις τιμές 0 έως και 3, η τιμή της $IDCG$ υπολογίζεται από την παρακάτω σχέση:

$$IDCG = \sum_{i=1}^3 \frac{i}{\log_2(i+1)} \quad (28)$$

3 Υλοποίηση αλγορίθμων για την εξαγωγή λέξεων φράσεων κλειδιά από έγγραφα

3.1 Επιλογή αλγορίθμων

Οι αλγόριθμοι που επιλέχθηκαν να εξεταστούν είναι οι παρακάτω:

- TF-IDF
- RAKE
- TEXT RANK
- YAKE
- KEYBERT

Επίσης, χρησιμοποιήθηκε και μια παραλλαγή του TF-IDF, όπου η τιμή IDF δεν αφορά την συλλογή των εγγράφων, αλλά το ίδιο το έγγραφο. Τη συγκεκριμένη παραλλαγή θα την συμβολίζουμε κατά σύμβαση ως TF-ISF (από το Inverted Sentence Frequency). Σύμφωνα με αυτή την παραλλαγή, κάθε έγγραφο θεωρείται μια συλλογή προτάσεων οπότε η τιμή του όρου ISF, για έναν όρο του εγγράφου, θα υπολογίζεται από την παρακάτω σχέση:

$$ISF = \begin{cases} 1, & S = 1 \\ \log\left(\frac{S}{s}\right), & S > 1 \end{cases} \quad (29)$$

Όπου, κατά αντιστοιχία με την παράμετρο IDF (εξίσωση 3), S είναι ο συνολικός αριθμός προτάσεων του εγγράφου και s ο αριθμός προτάσεων που περιέχουν τον όρο. Στην περίπτωση που το έγγραφο αποτελείται από μια μόνο πρόταση η τιμή του ISF θα είναι ίση με 1. Ως πρόταση σε ένα έγγραφο θεωρείται ένα υποσύνολο του εγγράφου που διαχωρίζεται από δύο συνεχόμενα σημεία στίξης

Σε ότι αφορά τον TEXTRANK επιλέχθηκαν οι παρακάτω υλοποιήσεις:

- Με κατευθυνόμενο γράφο (TX_RNK_dir)
- Με μη κατευθυνόμενο γράφο (TX_RNK_undir)

Αναφορικά με τον KEYBERT, χρησιμοποιήθηκε το εκπαιδευμένο μοντέλο “all-MiniLM-L6-v2” ([sentence-transformers/all-MiniLM-L6-v2 · Hugging Face](https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2)). Το συγκεκριμένο μοντέλο είναι ένα μοντέλο τύπου Transformer, σχεδιασμένο για την παραγωγή σημασιολογικών αναπαραστάσεων των όρων ενός κειμένου με χαμηλό υπολογιστικό κόστος. Βασίζεται στην

αρχιτεκτονική MiniLM (Wang et al, 2020), η οποία αποτελεί μια περισσότερο ελαφριά παραλλαγή του BERT. Το “all-MiniLM-L6-v2” αποτελείται από 6 επίπεδα transformer, με διανύσματα αναπαράστασης 384 διαστάσεων, γεγονός που το καθιστά ταχύτερο και πιο αποδοτικό σε σχέση με μεγαλύτερα μοντέλα όπως το BERT. Το μοντέλο, παράγει πυκνά διανύσματα που αποτυπώνουν τη σημασιολογία του κειμένου και όχι μόνο την επιφανειακή ομοιότητα λέξεων. Μέσω του KEYBERT, τα πυκνά διανύσματα του μοντέλου χρησιμοποιούνται για τον υπολογισμό της ομοιότητας συνημίτονου μεταξύ του εγγράφου και των υποψήφιων λέξεων-κλειδιών, επιτρέποντας την κατάταξή τους με βάση τη σημασιολογική τους συνάφεια. Παρά το μικρό του μέγεθος (περίπου 22 εκατομμύρια παράμετροι), το “all-MiniLM-L6-v2” παρουσιάζει πολύ καλή απόδοση σε εργασίες σημασιολογικής ομοιότητας και ανάκτησης πληροφορίας, καθιστώντας το κατάλληλο για εφαρμογές εξαγωγής ΛΦΚ, ιδιαίτερα σε περιβάλλοντα με περιορισμένους πόρους ή απαιτήσεις πραγματικού χρόνου.

3.2 Υλοποίηση αλγορίθμων

3.2.1 Γενικά

Οι αλγόριθμοι υλοποιήθηκαν με τη χρήση της γλώσσας προγραμματισμού Python v.3.13. Κάθε αλγόριθμος λάμβανε ως είσοδο το έγγραφο ως ένα αλφαριθμητικό και επέστρεφε το έγγραφο σε μορφή BoW. Η επιστρεφόμενη μορφή του εγγράφου ήταν μια λίστα που αποτελούνταν από πλειάδες δύο στοιχείων, την λέξη/φράση κλειδί και την βαθμολογία που λάμβανε από τον αλγόριθμο, ως εξής:

[(“keyword_1”, score1), (“keyword_2”, score2),...]

Επίσης, οι αλγόριθμοι υλοποιήθηκαν ώστε να υπάρχει η δυνατότητα να εφαρμοστεί stemming (εύρεση ρίζας) επί των ολόκληρων (unstemmed) όρων του κειμένου που βαθμολόγησαν. Το stemming έλαβε χώρα με τη χρήση του αλγόριθμου του Porter (1980). λαμβάνοντας υπόψη τα παρακάτω:

- Στην περίπτωση των n_grams ($n > 1$) η stemmed μορφή θα είναι ένα αλφαριθμητικό που αποτελείται από τις stemmed μορφές των unigrams (π.χ. “informatics technology” οδηγεί σε “inform techn”)
- Εάν δύο ή περισσότεροι unstemmed όροι οδηγήσουν στον ίδιο stemmed όρο (π.χ. “technology”, “technician” οδηγούν στον “techn”), τότε η βαθμολογία του stemmed όρου θα είναι η μέγιστη βαθμολογία των stemmed όρων.

Η λίστα των όρων (unstemmed ή stemmed) επιστρεφόταν ταξινομημένη σε φθίνουσα σειρά, προκειμένου οι πιο σημαντικοί όροι να είναι ταξινομημένοι πιο υψηλά, αφού το κείμενο είχε υποστεί γλωσσική επεξεργασία (ΕΦΓ), που συμπεριλάμβανε τα παρακάτω.

- Διακριτοποίηση όρων (tokenization)
- Αφαίρεση Κοινότυπων όρων (stopwords removal). Οι κοινότυποι όροι που χρησιμοποιήθηκαν στην παρούσα εργασία -και αφορούν την αγγλική γλώσσα-, παρουσιάζονται στο Παράρτημα 1.
- Αφαίρεση σημείων στίξης και δημιουργία προτάσεων
- Δημιουργία n_grams

3.2.2 Η κλάση Keyword

Πρόκειται για μια κλάση τα αντικείμενα της οποίας μοντελοποιούν τους όρους ενός εγγράφου. Συνεπώς ένα έγγραφο θεωρείται ως μια συλλογή αντικειμένων Keyword. Η συγκεκριμένη κλάση δεν έχει μεθόδους, παρά μόνο ιδιότητες στις οποίες αποθηκεύονται τιμές τις οποίες χρειάζονται οι αλγόριθμοι για να βαθμολογήσουν τους όρους ενός εγγράφου. Οι ιδιότητες self.txt, self.length και self.score, χρησιμοποιούνται από όλους τους αλγόριθμους, ενώ κάποιες άλλες ιδιότητες π.χ. self.tf, self.start_capital κλπ χρησιμοποιούνται από ορισμένους (π.χ TF ή YAKE). Η υλοποίηση της κλάσης μαζί με τα σχόλια για περισσότερες διευκρινίσεις, αναφέρονται στο Παράρτημα 2.

3.2.3 Η μητρική κλάση Algorithm

Η συγκεκριμένη κλάση αποτελεί την μητρική κλάση από την οποία, μέσω κληρονομικότητας, θα προκύψουν οι κλάσεις που υλοποιούν τον κάθε αλγόριθμο ξεχωριστά. Τα αντικείμενα των κλάσεων Algorithm είναι περιέκτες αντικειμένων Keyword. Οι προστατευμένες ιδιότητες της κλάσης δημιουργούνται στον κατασκευαστή της και είναι οι παρακάτω:

- self._text: Το κείμενο σε μορφή string ή λίστα από strings
- self._kwd_list: Η λίστα των αντικειμένων Keyword που θα προκύψουν από το κείμενο
- self._kwd_dict_idx: Ένα λεξικό της μορφής: κείμενο_όρου: θέση στην self._kwd_list

Η κλάση Algorithm περιέχει επίσης στατικές μεθόδους, προστατευμένες μεθόδους και δημόσιες μεθόδους, στις οποίες γίνεται αναφορά παρακάτω, ενώ ο κώδικας όλης της κλάσης αναφέρεται στο Παράρτημα 3.

Κατασκευαστής

- `def __init__(self, text)`

Παράμετροι: `text`: ένα αλφαριθμητικό (ή μία λίστα αλφαριθμητικών)

Το όρισμα `text` αντιπροσωπεύει το κείμενο επάνω στο οποίο θα τρέξουν οι αλγόριθμοι το οποίο αποθηκεύεται ως αλφαριθμητικό στην μεταβλητή `self._text`.

Προστατευμένες μέθοδοι

- `def __add_kwd_to_list(self, kwd_text)`

Παράμετροι: `kwd_text`: ένα αλφαριθμητικό

Η παραπάνω μέθοδος δέχεται ως όρισμα ένα αλφαριθμητικό και το επιστρέφει ως αντικείμενο `Keyword` από την `self._kwd_list`. Σε περίπτωση που δεν υπάρχει αντίστοιχο αντικείμενο `Keyword` στην `self._kwd_list`, τότε το αντικείμενο κατασκευάζεται, προστίθεται στην `self._kwd_list` και ακολούθως επιστρέφεται για περαιτέρω επεξεργασία.

- `def _stem_results(self)`

Η μέθοδος επιστρέφει μια νέα `self._kwd_list` και ένα νέο `self._kwd_dict_indx`, που περιέχει τους stemmed όρους της αρχικής `self._kwd_list`. Η μέθοδος χρησιμοποιεί την στατική μέθοδο `porter_stem()`.

- `def _text_to_single_string(self, text_list)`

Παράμετροι: `text_list`: Μία λίστα αλφαριθμητικών (π.χ. λίστα ολόκληρων παραγράφων)

Η μέθοδος συνενώνει τη λίστα αλφαριθμητικών σε ένα αλφαριθμητικό.

- `def _text_to_list_string(self, text)`

Παράμετροι: `text`: Ένα αλφαριθμητικό

Η μέθοδος μετατρέπει το αλφαριθμητικό σε λίστα παραγράφων, διασπώντας το αλφαριθμητικό στα σημεία αλλαγής γραμμής.

Δημόσιες μέθοδοι

- `def get_kwd_list(self, descending, top_n)`

Παράμετροι: `descending`: `True` (φθίνουσα σειρά) / `False` (αύξουσα σειρά)

`top_n`: Ακέραιος αριθμός ή `None`

Η μέθοδος ταξινομεί (δημιουργώντας μια νέα λίστα) τη `self._kwd_list` ανάλογα με τη βαθμολογία του κάθε Keyword αντικειμένου κατά αύξουσα ή φθίνουσα σειρά και επιστρέφει τα πρώτα `top_n` αντικείμενα υπό μορφή μιας λίστας πλειάδων της μορφής (`kwd`, `score`). Όταν το `top_n` είναι `None`, τότε επιστρέφονται όλα τα αντικείμενα της ταξινομημένης `self._kwd_list` ως λίστα πλειάδων.

- `def get_kwd_list_normalized (self, descending, top_n )`

Παράμετροι: `descending`: `True` (φθίνουσα σειρά) / `False` (αύξουσα σειρά)

`top_n`: Ακέραιος αριθμός ή `None`

Η μέθοδος ταξινομεί (δημιουργώντας μια νέα λίστα) τη `self._kwd_list` ανάλογα με τη βαθμολογία του κάθε Keyword και επιστρέφει τα πρώτα `top_n` αντικείμενα υπό μορφή μιας λίστας πλειάδων της μορφής (`kwd`, `normalized score`). Όταν το `top_n` είναι `None`, τότε επιστρέφονται όλα τα αντικείμενα της ταξινομημένης `self._kwd_list` ως λίστα πλειάδων.

Στατικές μέθοδοι

Οι στατικές μέθοδοι της κλάσης παρέχουν διαδικασίες ΕΦΓ όπως παρακάτω:

- `def tokenize_with_new_lines (text, stopwords, lowercase)`

Παράμετροι: `text`: Αλφαριθμητικό

`stopwords`: Λίστα αλφαριθμητικών

`lowercase`: Boolean μεταβλητή

Η μέθοδος απαιτεί την εισαγωγή του module `re` (regular expressions) και επιστρέφει το αλφαριθμητικό `text` ως μια λίστα όρων (`tokens`) και σημείων στίξεως αγνοώντας όρους που εμπεριέχονται στην λίστα `stopwords`. Επίσης, όταν η παράμετρος `lowercase` είναι `True`, οι όροι επιστρέφονται με πεζά γράμματα.

- `def make_sentences (tokenized_text)`

Παράμετροι: `tokenized_text`: Λίστα αλφαριθμητικών (και σημείων στίξεως)

Η μέθοδος αφαιρεί τα σημεία στίξης από τη λίστα του ορίσματος της και δημιουργεί «προτάσεις» μεταξύ δύο συνεχόμενων σημείων στίξης. Ως «πρόταση» νοείται μια ξεχωριστή λίστα όρων. Η μέθοδος επιστρέφει το όρισμα της ως μια λίστα που αποτελείται από επιμέρους λίστες («προτάσεις») όπως φαίνεται παρακάτω:

Κείμενο = [['όρος1_πρόταση1', 'όρος2_πρόταση1', ...],
['όρος1_πρόταση2', 'όρος2_πρόταση2', ...],

...

[‘όρος_k_πρόταση_n’, ‘όρος_m_πρόταση_n’, ...]]

- `def extract_all_n_grams (tokens_list, n_gram_size, min_n)`

Παράμετροι: `tokens_list`: Λίστα αλφαριθμητικών`n_gram_size`: Ακέραιος`min_n`: Ακέραιος

Η μέθοδος χρησιμοποιεί το module `ngrams` του πακέτου `nltk.util` και δημιουργεί `n_grams` από μια λίστα `unigrams`. Τα επιμέρους `n_grams` επιστρέφονται ως μια λίστα αλφαριθμητικών, ενώ η μέθοδος επιστρέφει μια λίστα με όλα τα `n_grams`. Το `n_gram_size`, καθορίζει τον μέγιστο αριθμό `n_grams` ενώ το `min_n` τον ελάχιστο αριθμό `n_grams`.

Για παράδειγμα όταν η μέθοδος δεχθεί ως όρισμα `['my', 'name', 'is', 'John']`, θα επιστρέψει για `min_n=1` και `n_gram_size=3`:

```
[  
    ['my', 'name', 'is', 'John'], → unigrams  
    ['my name', 'name is', 'is John'], → bigrams  
    ['my name is', 'name is John'] → trigrams  
]
```

- `def porter_stem (phrase)`

Παράμετροι: `phrase`: Αλφαριθμητικό

Η μέθοδος υλοποιεί το module `PorterStemmer` του πακέτου `nltk.stem` της `python` και επιστρέφει το αλφαριθμητικό του ορίσματος της όταν εφαρμοστεί σε αυτό ο αλγόριθμος του Porter (1980).

3.2.4 Η κλάση `myRake` (Algorithm)

Η συγκεκριμένη κλάση κληρονομεί την κλάση `Algorithm` και υλοποιεί τον αλγόριθμο `RAKE`, στον κατασκευαστή της, ο οποίος υπάρχει σαν έτοιμο πακέτο (`rake_nltk`) της `python`. Ωστόσο, η υλοποίηση του `RAKE` από το έτοιμο πακέτο βαθμολογεί μόνο τις φράσεις του εγγράφου αλλά όχι τα `n_grams` των όρων τους. Συνεπώς ο κατασκευαστής της κλάσης `myRake` έχει υλοποιηθεί ώστε να προσφέρει την επιθυμητή λειτουργικότητα, δηλαδή να υπολογίζει τη βαθμολογία των `unigrams` και ακολούθως να υπολογίζει τη βαθμολογία των `n_grams` των προτάσεων.

Κατασκευαστής

- `def __init__(self, text, n_grams, stop_words, stemmer)`

Παράμετροι: text: αλφαριθμητικό

n_grams: ο μέγιστος αριθμός n_grams

stop_words: Λίστα αλφαριθμητικών κοινότυπων λέξεων

stemmer: Boolean (True να λάβει χώρα stemming)

Ο κατασκευαστής αρχικά καλεί τον κατασκευαστή της Algorithm και ακολούθως υλοποιεί τον αλγόριθμο από το έτοιμο πακέτο της python και αποθηκεύει σε μεταβλητές (α) τις συνυπάρξεις των unigrams (β) τις συχνότητες του κάθε unigram και (γ) τις φράσεις που προκρίνει ο αλγόριθμος του πακέτου. Με βάση τις παραπάνω πληροφορίες ο κατασκευαστής δημιουργεί τη self._kwd_list, πρώτα για τα unigrams και ακολούθως της προσθέτει n_grams της κάθε φράσης προτάσεων ως άθροισμα των βαθμολογιών των επιμέρους unigrams. Στο τέλος και σε περίπτωση που η μεταβλητή stemmer έχει την τιμή True, καλείται η προστατευμένη μέθοδος self._stem_results() της Algorithm ώστε η self._kwd_list να περιέχει τις γλωσσικές ρίζες των όρων και των φράσεων.

Ιδιωτικές μέθοδοι:

- `def _sum_rake_n_grams(self, n_gram_string)`

Παράμετροι: n_gram_string: αλφαριθμητικό (φράση/όρος)

Η μέθοδος λαμβάνει ένα όρισμα και δημιουργεί ένα αντικείμενο Keyword το οποίο το προσθέτει στη self._kwd_list, αφού πρώτα υπολογίσει τη βαθμολογία του ως άθροισμα των βαθμολογιών των επιμέρους unigram που το απαρτίζουν. Η μέθοδος λειτουργεί και εάν το n_gram_string είναι unigram (στη βαθμολογία του προστίθεται το 0).

Πρόκειται για μια βοηθητική μέθοδο του κατασκευαστή.

Για να υπολογίσει ο αλγόριθμος myRake, τις λέξεις-φράσεις κλειδιά από ένα κείμενο (text) και να τις επιστρέψει ως ταξινομημένη λίστα πλειάδων (όρος, βαθμολογία) απαιτείται να κληθεί ο κατασκευαστής της κλάσης και η δημόσια μέθοδος get_kwd_list (self, descending, top_n) της κλάσης Algorithm, ως παρακάτω:

`r= myRake(text, n_grams, stop_words, stemmer)`

`r. get_kwd_list (descending, top_n )`

Ο κώδικας της κλάσης myRake αναφέρεται στο Παράρτημα 4.

3.2.5 Η κλάση myYake (Algorithm)

Η συγκεκριμένη κλάση κληρονομεί την κλάση Algorithm και υλοποιεί τον αλγόριθμο YAKE, στον κατασκευαστή της, όπως περιγράφεται από τους **Campos et al, 2020**.

Κατασκευαστής

- `def __init__(self, text, n_grams, stop_words, stemmer)`

Παράμετροι: text: αλφαριθμητικό

n_grams: ο μέγιστος αριθμός n_grams

stop_words: Λίστα αλφαριθμητικών κοινότυπων λέξεων

stemmer: Boolean (True να λάβει χώρα stemming)

Στον κατασκευαστή αρχικά λαμβάνει χώρα ΕΦΓ (με stemming εάν stemmer είναι True) των όρων του κειμένου (text), ώστε το κείμενο να μετατραπεί ως μια λίστα n_gram προτάσεων, όπου η κάθε n-gram πρόταση είναι μία λίστα των n_gram όρων της. Ακολούθως και με βάση τα unigrams (α) προστίθενται τα unigram Keywords στη `self._kwd_list` και (β) υπολογίζονται και αποθηκεύονται στις ιδιωτικές μεταβλητές `seld._doc_word_max_count`, `self._doc_mean_counts` και `self.doc_stdv_counts` η συχνότητα του όρου που αναφέρεται περισσότερο στο έγγραφο, η μέση συχνότητα των όρων στο έγγραφο και η διασπορά τους αντίστοιχα. Στη συνέχεια υπολογίζεται για κάθε unigram η βαθμολογία του. Ακολούθως λαμβάνει δεύτερη διαπέραση των n_gram προτάσεων του εγγράφου ώστε να προστεθούν τα Keyword αντικείμενα που αντιστοιχούν στα bigrams, trigrams κλπ στη `self._kwd_list`. Η βαθμολογία των n_grams (n>1) είναι συνάρτηση της βαθμολογίας των αντίστοιχων unigrams όπως προκύπτει από την συνάρτηση των Campos et al (2020). Ακολούθως οι βαθμολογίες όλων των όρων της `self._kwd_list` αντιστρέφονται, διότι ο YAKE βαθμολογεί με χαμηλότερη βαθμολογία τους πιο σημαντικούς όρους/φράσεις σε ένα κείμενο.

Προστατευμένες - Ιδιωτικές μέθοδοι

Η λειτουργικότητα που αναφέρεται στον κατασκευαστή επιτυγχάνεται μέσω των παρακάτω προστατευμένων μεθόδων:

- `def _preprocess(self, text, stopwords, n_gram_size, stemmer)`

Η μέθοδος επιστρέφει το κείμενο (text) του ορίσματος της ως λίστα που περιέχει τις n_gram προτάσεις, ως λίστες των n_gram όρων τους αφαιρώντας τους κοινότυπους όρους και κάνοντας stemming εάν το όρισμα stemmer είναι True.

- `def _run_Yake_unigrams(self, kwd)`

Η μέθοδος υπολογίζει τη βαθμολογία ενός unigram όρου που δέχεται ως όρισμα, χρησιμοποιώντας τις παρακάτω βοηθητικές ιδιωτικές μεθόδους:

- `def __calculate_casing(self, kwd)`
- `def __calculate_positional(self, kwd):`
- `def __calculate_word_frequency(self, kwd)`
- `def __calculate_relatedness_to_context(self, kwd)`
- `def __calculate_different_sentence(self, kwd)`

- `def _run_Yake_n_phrase(self, kwd_phrase)`

Η μέθοδος υπολογίζει τη βαθμολογία της φράσης του ορίσματος της σύμφωνα με τις βαθμολογίες των unigram όρων που την απαρτίζουν.

Για να υπολογίσει ο αλγόριθμος myYake, τις λέξεις-φράσεις κλειδιά από ένα κείμενο (text) και να τις επιστρέψει ως ταξινομημένη λίστα πλειάδων (όρος, βαθμολογία) απαιτείται να κληθεί ο κατασκευαστής της κλάσης και η δημόσια μέθοδος `get_kwd_list (self, descending, top_n )` της κλάσης Algorithm, ως παρακάτω:

```
r= myYake(text, n_grams, stop_words, stemmer)
r. get_kwd_list (descending, top_n )
```

Ο αλγόριθμος YAKE μπορεί να υλοποιηθεί και από το έτοιμο πακέτο yake της Python. Ωστόσο, σε προκαταρκτικές μετρήσεις η υλοποίηση μέσω του έτοιμου πακέτου ήταν εξαιρετικά αργή (περισσότερο από 500 sec για ένα μέσο άρθρο). Για τον λόγο αυτό προτιμήθηκε να υλοποιηθεί ο αλγόριθμος σύμφωνα με την βιβλιογραφία (Campos et al, 2020) από την αρχή. Ο κώδικας της κλάσης myYake αλλά και της κλάσης μέσω του έτοιμου πακέτου της Python (`myYake_from_pip`) αναφέρονται στα Παραρτήματα 5 και 6 αντίστοιχα, όπου μέσω των σχολίων αναφέρονται και περισσότερες πληροφορίες για τον τρόπο υπολογισμού των βαθμολογιών.

3.2.6 Η κλάση myTextRank (Algorithm)

Η συγκεκριμένη κλάση κληρονομεί την κλάση Algorithm και υλοποιεί τον αλγόριθμο TEXTRANK με κατευθυνόμενο γράφο (κάθε `n_gram` ενώνεται με ακμή με τα επόμενα του σε μια πρόταση) ή μη κατευθυνόμενο γράφο (κάθε `n_gram` ενώνεται με ακμή με όλα τα `n_grams` μιας πρότασης). Η συγκεκριμένη υλοποίηση είναι εκείνη που προτείνεται από τους Rajput et al 2020, σύμφωνα με την οποία (α) όλα τα `n_grams` είναι κόμβοι του γράφου (και όχι μόνο τα

unigrams) και (β) η τελική βαθμολογία κάθε n_gram θα είναι 4πλάσια ή 6πλάσια εκείνης που υπολογίζει ο αλγόριθμος PAGERANK, όταν $n=2$ ή $n \geq 3$ αντίστοιχα. Για την υλοποίηση του TEXTRANK χρησιμοποιήθηκε το έτοιμο πακέτο networkx της Python, το οποίο έχει βιβλιοθήκες για τη δημιουργία αντικειμένων γράφων αλλά και μεθόδων που επενεργούν σε αυτούς όπως η μέθοδος pagerank() η οποία χρησιμοποιήθηκε στην παρούσα υλοποίηση.

Κατασκευαστής

- `def __init__(self, text, n_grams, stop_words, directed, stemmer, n_gram_weights=[1,4,6], window_size=4)`

Παράμετροι: text: αλφαριθμητικό

n_grams: ο μέγιστος αριθμός n_grams

stop_words: Λίστα αλφαριθμητικών κοινότυπων λέξεων

directed: Boolean (True για κατευθυνόμενο γράφο αλλιώς μη κατευθυνόμενος γράφος)

stemmer: Boolean (True να λάβει χώρα stemming)

n_gram_weights: πίνακας βαρών των n_grams (το n είναι αντίστοιχο της θέσης του πίνακα). Λαμβάνει την προεπιλεγμένη τιμή [1,4,6] όπως προτείνεται από **Rajput et al 2020**,

window_size: Μέγιστος αριθμός επόμενων κόμβων για να δημιουργηθούν ακμές από έναν κόμβο. Λαμβάνει προεπιλεγμένη (default) τιμή ίση με 4, όπως προτείνεται και από τη βιβλιογραφία

Στον κατασκευαστή αρχικά λαμβάνει χώρα ΕΦΓ των όρων του κειμένου (text), ώστε το κείμενο να μετατραπεί ως μια λίστα n_gram προτάσεων, όπου η κάθε n_gram πρόταση είναι μία λίστα των n_gram όρων της. Ακολούθως δημιουργείται ο γράφος του εγγράφου (κατευθυνόμενος ή μη ανάλογα την τιμή του ορίσματος directed) για όλα τα n_grams του εγγράφου. Ακολούθως καλείται η μέθοδος pagerank(graph) με όρισμα τον γράφο του εγγράφου, η οποία βαθμολογεί ανάλογα όλους τους κόμβους (n_grams) του γράφου σύμφωνα με τον αλγόριθμο PAGERANK. Στη συνέχεια τα Keyword αντικείμενα προστίθενται στη self._kwd_list λαμβάνοντας υπόψη τα παρακάτω:

- `unigram_score = PageRank_score`
- `bigram_score = 4 * PageRank_score`
- `trigram (or higher)_score = 6 * PageRank_score`

Στο τέλος και σε περίπτωση που η μεταβλητή stemmer έχει την τιμή True, καλείται η προστατευμένη μέθοδος self._stem_results() της Algorithm ώστε η self._kwd_list να περιέχει τις γλωσσικές ρίζες των όρων και των φράσεων.

Προστατευμένες - μέθοδοι

Η λειτουργικότητα που αναφέρεται στον κατασκευαστή επιτυγχάνεται μέσω των παρακάτω προστατευμένων μεθόδων:

- def _preprocess(self, text, stopwords, n_gram_size)

Η μέθοδος επιστρέφει το κείμενο (text) του ορίσματος της ως λίστα που περιέχει τις n_gram προτάσεις, ως λίστες των n_gram όρων τους αφαιρώντας τους κοινότυπους όρους

- def _build_graph(self, candidates, directed, window_size)

Παράμετροι: candidates: η λίστα όλων των n_grams του εγγράφου

directed: Boolean (Κατευθυνόμενος ή μη γράφος)

window_size: Μέγιστος αριθμός επόμενων κόμβων για να δημιουργηθούν ακμές από έναν κόμβο

Η μέθοδος επιστρέφει τον γράφο (κατευθυνόμενο ή μη ανάλογα το όρισμα directed) του εγγράφου, από τη λίστα candidates που έχει επιστραφεί από τη μέθοδο self._preprocess()

- def _weight_textrank_results(self, weights)

Παράμετροι: weights: πίνακας βαρών των n_grams (το n είναι αντίστοιχο της θέσης του πίνακα)

Η μέθοδος διατρέχει την self._kwd_list και βαθμολογεί τα Keywords αντικείμενα ανάλογα με τον πίνακα βαρών που έχει δεχθεί ως όρισμα.

Για να υπολογίσει ο αλγόριθμος myTextRank, τις λέξεις-φράσεις κλειδιά από ένα κείμενο (text) και να τις επιστρέψει ως ταξινομημένη λίστα πλειάδων (όρος, βαθμολογία) απαιτείται να κληθεί ο κατασκευαστής της κλάσης και η δημόσια μέθοδος get_kwd_list (self, descending, top_n) της κλάσης Algorithm, ως παρακάτω:

```
r= myTextRank(text, n_grams, stop_words, directed, stemmer) # with default params  
r. get_kwd_list (descending, top_n )
```

Ο κώδικας της κλάσης myTextRank αναφέρεται στο Παράρτημα 7.

3.2.7 Η κλάση myKeyBert (Algorithm)

Η συγκεκριμένη κλάση κληρονομεί την κλάση Algorithm και υλοποιεί τον αλγόριθμο KEYBERT στον κατασκευαστή της. Η κλάση δεν έχει ιδιωτικές μεθόδους/ιδιότητες και λειτουργεί περισσότερο ως περιτυλιχτής, καθώς υπάρχει έτοιμη υλοποίηση του αλγόριθμου μέσω των πακέτων keybert και sentence-transformers. Στην συγκεκριμένη υλοποίηση το εκπαιδευμένο μοντέλο 'all-MiniLM-L6-v2' φορτώθηκε μέσω του πακέτου keybert, και ακολούθως περάστηκε ως παράμετρος στον κατασκευαστή της κλάσης myKeybert. Στη περίπτωση που απαιτείται stemming αυτό λαμβάνει χώρα αφού ο αλγόριθμος KEYBERT υπολογίσει τις βαθμολογίες επί των unstemmed όρων του κειμένου, με την κλήση της προστατευμένης μεθόδου self._stem_results() της Algorithm. Για να υπολογίσει ο αλγόριθμος myKeyBert, τις λέξεις-φράσεις κλειδιά από ένα κείμενο (text) και να τις επιστρέψει ως ταξινομημένη λίστα πλειάδων (όρος, βαθμολογία) απαιτείται να κληθεί ο κατασκευαστής της κλάσης και η δημόσια μέθοδος get_kwd_list (self, descending, top_n) της κλάσης Algorithm, ως παρακάτω:

```
keybert_model = Keybert(model='all-MiniLM-L6-v2')  
r= myKeybert(text, n_grams, stop_words, stemmer, keybert_model)  
r.get_kwd_list(descending, top_n )
```

Ο κώδικας της κλάσης myKeybert αναφέρεται στο Παράρτημα 8.

3.2.8 Η κλάση InvertSentFreq (Algorithm)

Η συγκεκριμένη κλάση κληρονομεί την κλάση Algorithm και υλοποιεί τον αλγόριθμο TF-ISF, ενώ παράλληλα μέσω της δημόσιας μεθόδου της get_frequencies επιστρέφει (όχι ταξινομημένες) τις συχνότητες εμφάνισης των n_gram όρων του κειμένου, προκειμένου να τις χρησιμοποιήσει ο αλγόριθμος TF-IDF.

Κατασκευαστής

- def __init__(self, text, n_grams, stop_words, stemmer)

Παράμετροι: text: αλφαριθμητικό

n_grams: ο μέγιστος αριθμός n_grams

stop_words: Λίστα αλφαριθμητικών κοινότυπων λέξεων

stemmer: Boolean (True να λάβει χώρα stemming)

Στον κατασκευαστή αρχικά λαμβάνει χώρα ΕΦΓ των όρων του κειμένου (text), ώστε το κείμενο να μετατραπεί ως μια λίστα `n_gram` προτάσεων, όπου η κάθε `n-gram` πρόταση είναι μία λίστα των `n_gram` όρων της. Η ΕΦΓ συμπεριλαμβάνει stemming στην περίπτωση που η παράμετρος `stemmer` λάβει την τιμή `True`. Ακολούθως προστίθενται στη `self._kwd_list` τα αντίστοιχα `Keyword` αντικείμενα. Για κάθε `n_gram` αντικείμενο υπολογίζεται η συχνότητα εμφάνισης στο έγγραφο ως αριθμός εμφανίσεων/σύνολο αντιστοιχών `n_gram` (για παράδειγμα σε ένα `bigram` ο παρονομαστής θα είναι το σύνολο των `bi-grams` του εγγράφου) και υπολογίζεται η παράμετρος `ISF`, όπως αναφέρθηκε στην ενότητα 3.1 (εξίσωση 29).

Προστατευμένες - μέθοδοι

Η λειτουργικότητα που αναφέρεται στον κατασκευαστή επιτυγχάνεται μέσω των παρακάτω προστατευμένων μεθόδων:

- `def _preprocess(self, text, stopwords, n_gram_size, stemmer)`
Η μέθοδος επιστρέφει το κείμενο (text) του ορίσματος της ως λίστα που περιέχει τις `n_gram` προτάσεις, ως λίστες των `n_gram` όρων τους αφαιρώντας τους κοινότυπους όρους και κάνοντας stemming εάν το όρισμα `stemmer` είναι `True`.
- `def _update_freq_score(self)`
Η μέθοδος υπολογίζει τη βαθμολογία κάθε όρου υλοποιώντας την παραλλαγή αλγορίθμου TF-LIDF.

Δημόσιες μέθοδοι

- `def get_frequencies(self)`
Η μέθοδος επιστρέφει ως λίστα πλειάδων (όρος, συχνότητα) τις συχνότητες εμφάνισης των `n_gram` όρων του κειμένου. Η λίστα πλειάδων δεν είναι ταξινομημένη.

Για να υπολογίσει ο αλγόριθμος TF-ISF, τις λέξεις-φράσεις κλειδιά από ένα κείμενο (text) και να τις επιστρέψει ως ταξινομημένη λίστα πλειάδων (όρος, βαθμολογία) απαιτείται να κληθεί ο κατασκευαστής της κλάσης και η δημόσια μέθοδος `get_kwd_list (self, descending, top_n )` της κλάσης `Algorithm`, ως παρακάτω:

```
r= InvertSentFreq(text, n_grams, stop_words, stemmer)
r. get_kwd_list (descending, top_n )
```

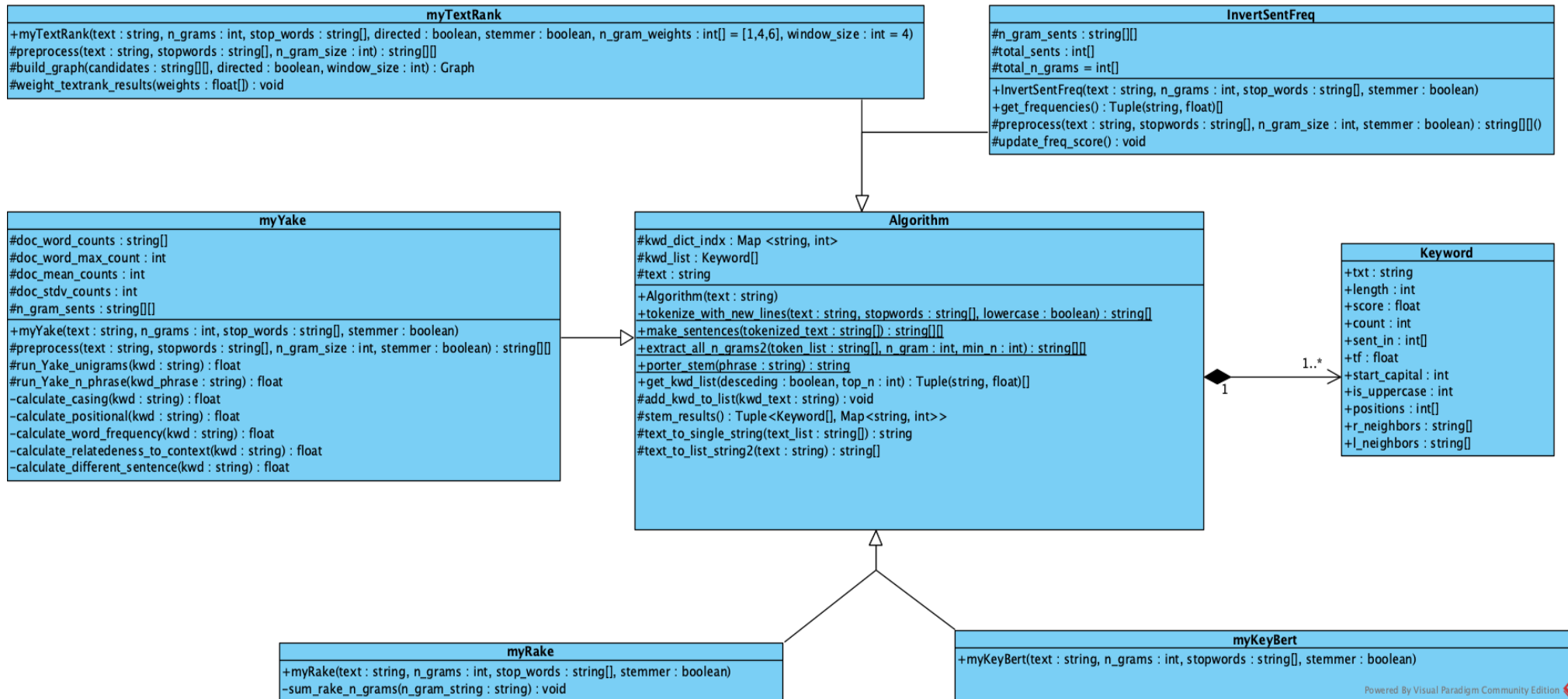
Επίσης, οι συχνότητες εμφάνισης των `n_gram` όρων ενός κειμένου, προκειμένου να χρησιμοποιηθούν στον αλγόριθμο TF-IDF, μπορούν να ληφθούν αφού κληθεί ο κατασκευαστής και η δημόσια μέθοδος `get_frequencies()` της `InvertSentFreq`, ως παρακάτω:

```
r= InvertSentFreq(text, n_grams, stop_words, stemmer)
r.get_frequencies () # attention not a sorted list!
```

Ο κώδικας της κλάσης `InvertSentFreq` αναφέρεται στο Παράρτημα 9.

3.2.9 Διαγράμματα κλάσεων UML

Οι κλάσεις που περιεγράφηκαν στις προηγούμενες υπο-ενότητες απεικονίζονται στο διάγραμμα κλάσεων της UML, του Σχήματος 1. Η δημιουργία του διαγράμματος UML έγινε με τη χρήση του λογισμικού Visual Paradigm Community Edition v.17.2.



Σχήμα 3-1 Διάγραμμα κλάσεων UML των αλγορίθμων που υλοποιήθηκαν

4 Συγκριτική αξιολόγηση αλγορίθμων

4.1 Συλλογές δεδομένων αξιολόγησης (datasets)

Για την σύγκριση των αλγορίθμων χρησιμοποιήθηκαν οι συλλογές εγγράφων Krapivin_2009 (<https://github.com/SDuari/Keyword-Extraction-Datasets/blob/master/Krapivin2009.zip>) και Aminer Citation Network (ACN) (<https://www.kaggle.com/datasets/mathurinache/citation-network-dataset>). Η Krapivin_2009 είναι μια συλλογή από 2300 πλήρη άρθρα επιστημονικών περιοδικών και έχει σε ξεχωριστό αρχείο τις λέξεις/φράσεις κλειδιά κάθε μιας από αυτές. Αντίστοιχης μορφής είναι και η ACN με περίπου 2000 εγγραφές περιλήψεων εργασιών που είναι δημοσιευμένες σε επιστημονικά περιοδικά. Η συλλογή Krapivin_2009 χρησιμοποιήθηκε για τη συγκριτική αξιολόγηση της ικανότητας των αλγορίθμων να βαθμολογούν με υψηλή βαθμολογία τις λέξεις κλειδιά ενός εγγράφου, ενώ η συλλογή ACN χρησιμοποιήθηκε προκειμένου να αξιολογηθούν οι αλγόριθμοι κατά πόσο είναι σε θέση να προβλέψουν (ή να κάνουν σύσταση) το περιοδικό στο οποίο είναι δημοσιευμένη μια εργασία.

4.2 Συγκριτική αξιολόγηση αλγορίθμων για πρόβλεψη λέξεων/φράσεων κλειδιά– Μετρικές σύγκρισης

Για κάθε έγγραφο της συλλογής και πριν εφαρμοστεί η σύγκριση των αλγορίθμων επάνω σε αυτό, γινόταν έλεγχος εάν οι φράσεις/λέξεις κλειδιά του εγγράφου εμπεριέχονταν στο κείμενο του εγγράφου. Στην περίπτωση που μια λέξη κλειδί δεν υπήρχε στο κείμενο, τότε αυτή αγνοούνταν από τους υπολογισμούς των μετρικών αξιολόγησης. Στην περίπτωση που μια φράση κλειδί δεν εμπεριέχονταν στο κείμενο, τότε η φράση διασπώταν σε unigrams και ακολούθως λαμβάνονταν υπόψη στους υπολογισμούς των μετρικών, μόνο εκείνα τα unigrams που περιέχονταν στο κείμενο, υπό την προϋπόθεση ότι δεν είχαν ήδη συμπεριληφθεί νωρίτερα στις λέξεις κλειδιά του εγγράφου.

Κάθε φράση/λέξη κλειδί του εγγράφου υπόκειτο στην ίδια ΕΦΓ με εκείνη του αλγόριθμου που θα εφαρμοζόταν σε αυτό, ενώ οι μετρήσεις έλαβαν χώρα για αριθμό $n_grams=1$, $n_grams \leq 2$ και $n_grams \leq 3$. Επίσης, για κάθε n_gram οι αλγόριθμοι αξιολογούνταν τόσο με stemming όσο και χωρίς stemming.

Οι μετρικές που χρησιμοποιήθηκαν για την αξιολόγηση των αλγορίθμων ήταν οι εξής:

- *Mean Recall (MR@k)*

- *Mean Precision (MP@k)*
- *Mean Average Precision (MAP@k)*
- *Mean Reciprocal Rank (MRR)*
- *Mean Normalized Discounted Cumulative Gain (MnDCG)*

Οι $MR@k$, $MP@k$ και $MAP@k$, επιλέχθηκε να υπολογιστούν στο 5%, 10%, 30% και 60% των πιο σημαντικών λέξεων/φράσεων κλειδιών που επιστρέφει ο κάθε αλγόριθμος. Επίσης, ως προς την $MnDCG$ επιλέχθηκαν να υπολογιστούν δύο τιμές που τις συμβολίζουμε κατά σύμβαση ως $MnDCG_unordered$ και $MnDCG_ordered$. Για τον υπολογισμό της $MnDCG_unordered$ θεωρήθηκε ότι όλες οι λέξεις/φράσεις κλειδιά του εγγράφου της συλλογής έχουν τον ίδιο βαθμό σημαντικότητας, ενώ κατά τον υπολογισμό της $MnDCG_ordered$ θεωρήθηκε ότι οι λέξεις/φράσεις κλειδιά του εγγράφου της συλλογής βαθμολογούνται από αριστερά προς τα δεξιά με φθίνοντα βαθμό σημαντικότητας. Η παραπάνω θεώρηση στηρίχτηκε στην υπόθεση ότι όταν ένας ερευνητής επιλέγει τις λέξεις/φράσεις κλειδιά για μια εργασία του, επιλέγει να αναφέρει πρώτα εκείνες που είναι περισσότερο αντιπροσωπευτικές για το περιεχόμενο της.

Τέλος, οι αλγόριθμοι συγκρίθηκαν και ως προς τη μέση ταχύτητα εκτέλεσης τους, εκφρασμένη σε msec, να βαθμολογήσουν τους όρους/φράσεις κλειδιά ενός άρθρου της συλλογής για $n_grams=3$. Οι μετρήσεις έλαβαν χώρα σε υπολογιστή Apple M4/16GB και σε λειτουργικό σύστημα macOS Sequoia 15.5. Η υλοποίηση της συγκριτικής αξιολόγησης των αλγορίθμων έγινε σε γλώσσα προγραμματισμού Python v.3.13.

4.3 Συγκριτική αξιολόγηση αλγορίθμων για πρόβλεψη λέξεων/φράσεων κλειδιά - Αποτελέσματα

4.3.1 $n_grams=1$

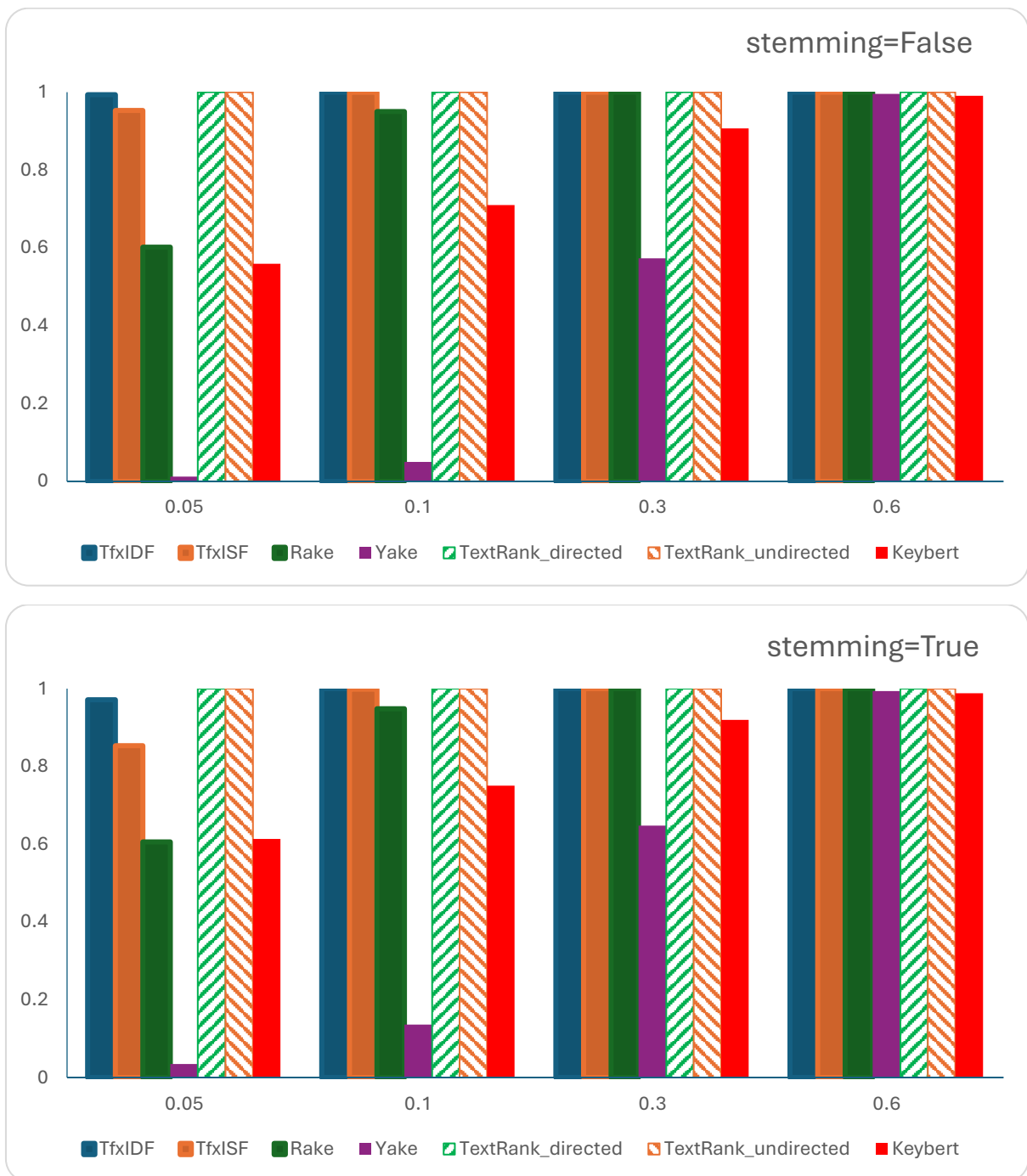
Σε ότι αφορά στην ακρίβεια οι αλγόριθμοι δεν εμφάνισαν υψηλές επιδόσεις, καθώς η υψηλότερη τιμή παρατηρήθηκε για τον TEXTRANK (0,25 και 0,2 όταν εφαρμόστηκε και όταν δεν εφαρμόστηκε stemming αντίστοιχα). Ωστόσο, οι υπόλοιποι αλγόριθμοι εμφάνισαν πολύ υψηλές τιμές της μετρικής ανάκλασης ακόμα και στο 5% των λέξεων που είχαν προβλέψει, με μόνη εξαίρεση τον YAKE, οποίος καθυστέρησε ($k>0,3$) να εμφανίσει υψηλά ποσοστά ανάκλασης. Ακόμα και σε αυτή την περίπτωση οι πιο αποτελεσματικοί αλγόριθμοι ήταν οι TF-IDF, TF-ISF και TEXTRANK (ανεξάρτητα της υλοποίησης του) με τιμές ανάκλασης >0.9 ακόμα και στο 5% των λέξεων κλειδιών που είχαν προβλέψει, ενώ οι αντίστοιχη τιμή της

ανάκλασης για τους αλγόριθμους RAKE και KEYBERT ήταν χαμηλότερη του 0,6. Ωστόσο για $k > 0,1$, τόσο ο RAKE όσο και ο KEYBERT αύξησαν την επίδοσή τους (τιμή $R@0,3 > 0,8$). Οι τιμές των μετρικών φάνηκε να μην επηρεάζονται από το εάν έλαβε χώρα stemming ή όχι. Για παράδειγμα η μέγιστη τιμή της μετρικής F1 παρατηρήθηκε στον TEXTRANK και ήταν 0,39 και 0,31 με stemming και χωρίς stemming αντίστοιχα.

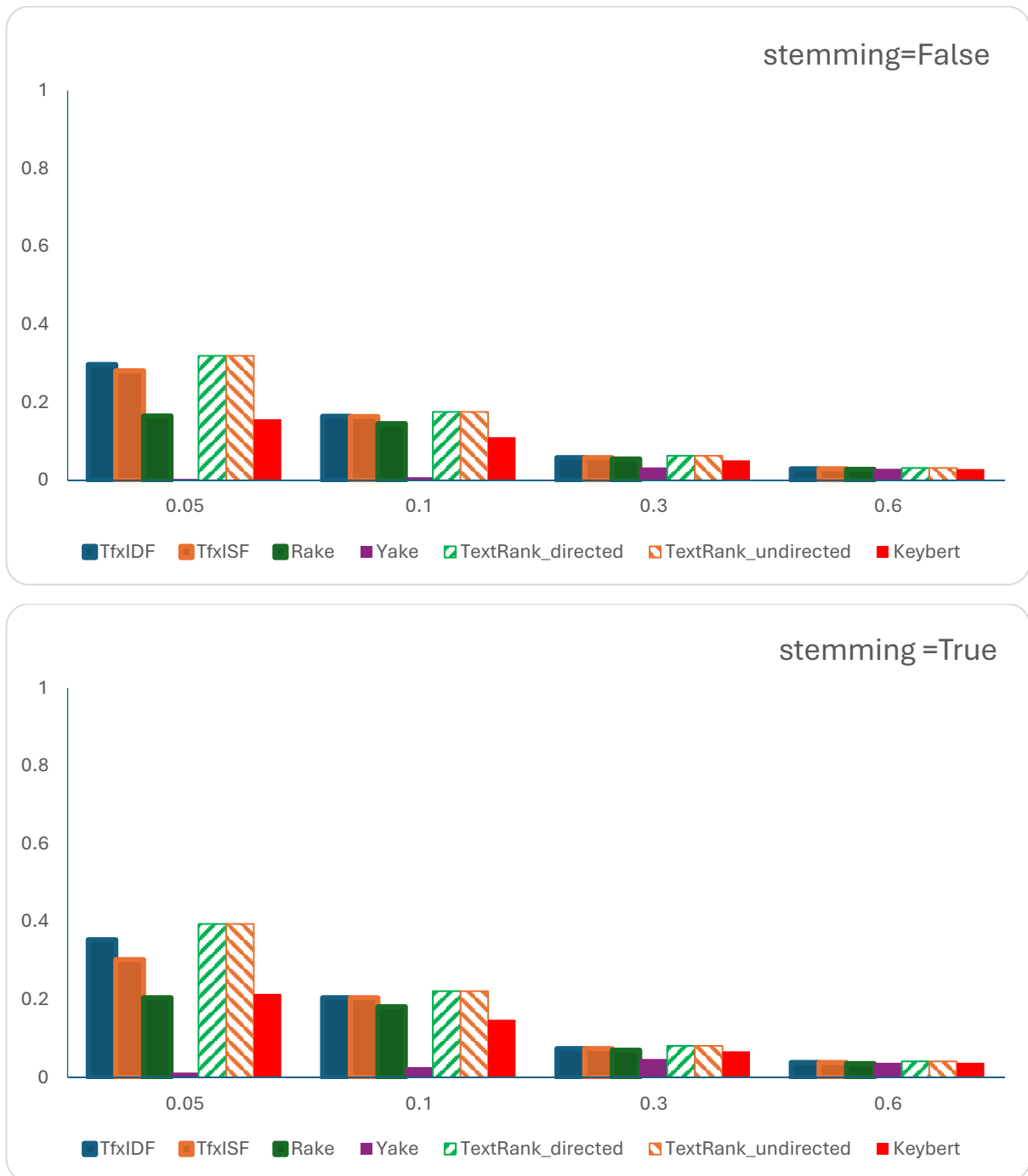
Την ίδια εικόνα παρουσίασαν οι αλγόριθμοι και ως προς τις μετρικές MAP, MRR και nDCG, με τον TEXTRANK και ειδικότερα την υλοποίηση του με κατευθυνόμενο γράφο να εμφανίζει τις υψηλότερες επιδόσεις σε σχέση με τους υπόλοιπους αλγόριθμους, ενώ οι YAKE και RAKE να εμφανίζουν την χαμηλότερη επίδοση. Οι TF αλγόριθμοι ξεπέρασαν σε επίδοση τον KEYBERT για όλες τις μετρικές με εξαίρεση την MRR όπου όταν δεν εφαρμόστηκε stemming η τιμή των TF αλγορίθμων και του KEYBERT ήταν στα ίδια επίπεδα (0.5). Βέβαια στην περίπτωση που εφαρμόστηκε stemming η τιμή της συγκεκριμένης μετρικής αυξήθηκε στο 0,68 και ίσως αποτελεί την μοναδική περίπτωση που η τιμή μιας μετρικής διαφοροποιήθηκε όταν εφαρμόστηκε stemming και όταν δεν εφαρμόστηκε. Τα αποτελέσματα των μετρήσεων φαίνονται στα διαγράμματα (Σχήματα 4-1 έως και 4-4) που ακολουθούν.



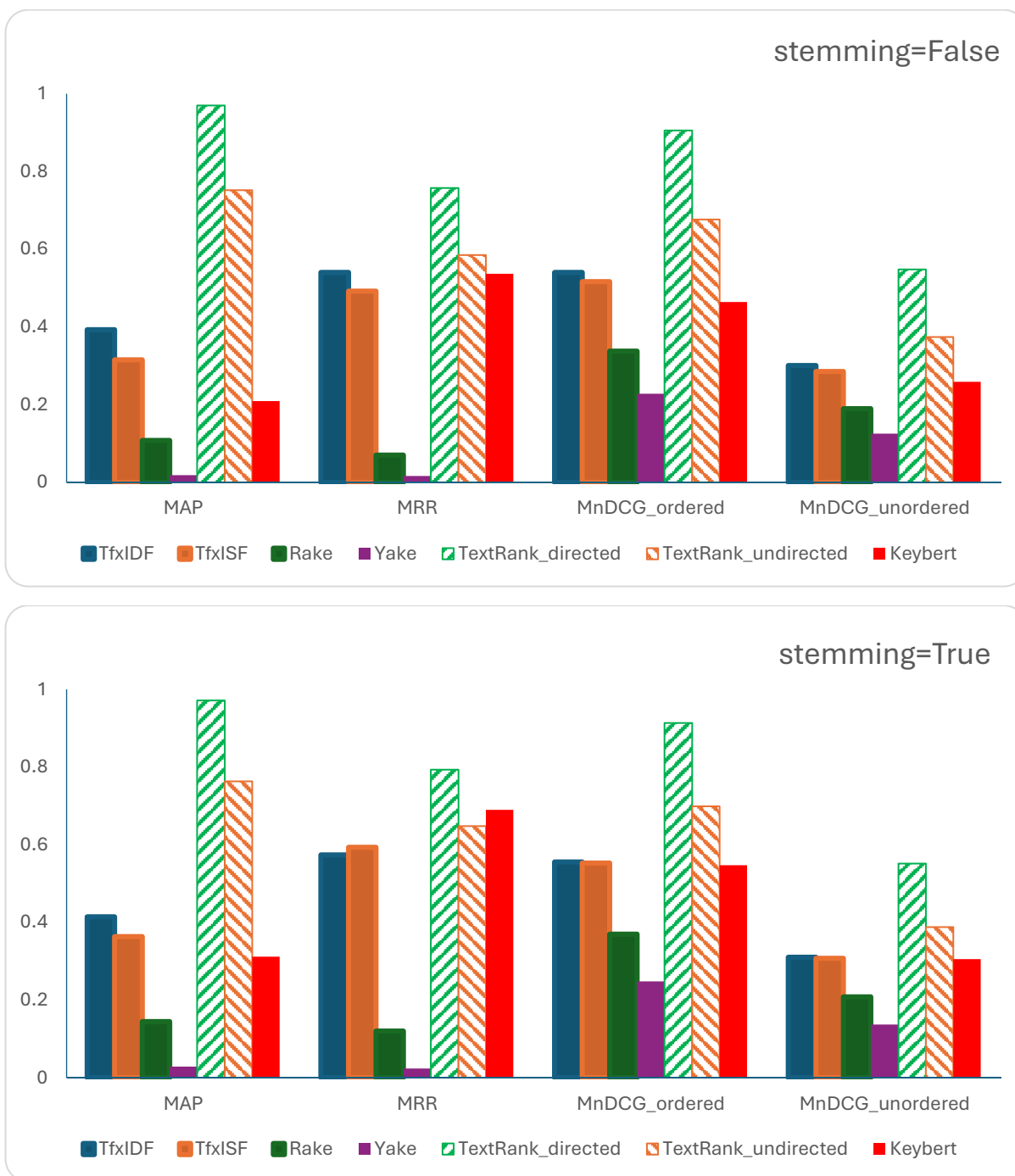
Σχήμα 4-1 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $P@k$, για $n_grams=1$ με stemming και χωρίς stemming



Σχήμα 4-2 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $R@k$, για $n_grams=1$ με stemming και χωρίς stemming



Σχήμα 4-3 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $F1@k$, για $n_grams=1$ με stemming και χωρίς stemming

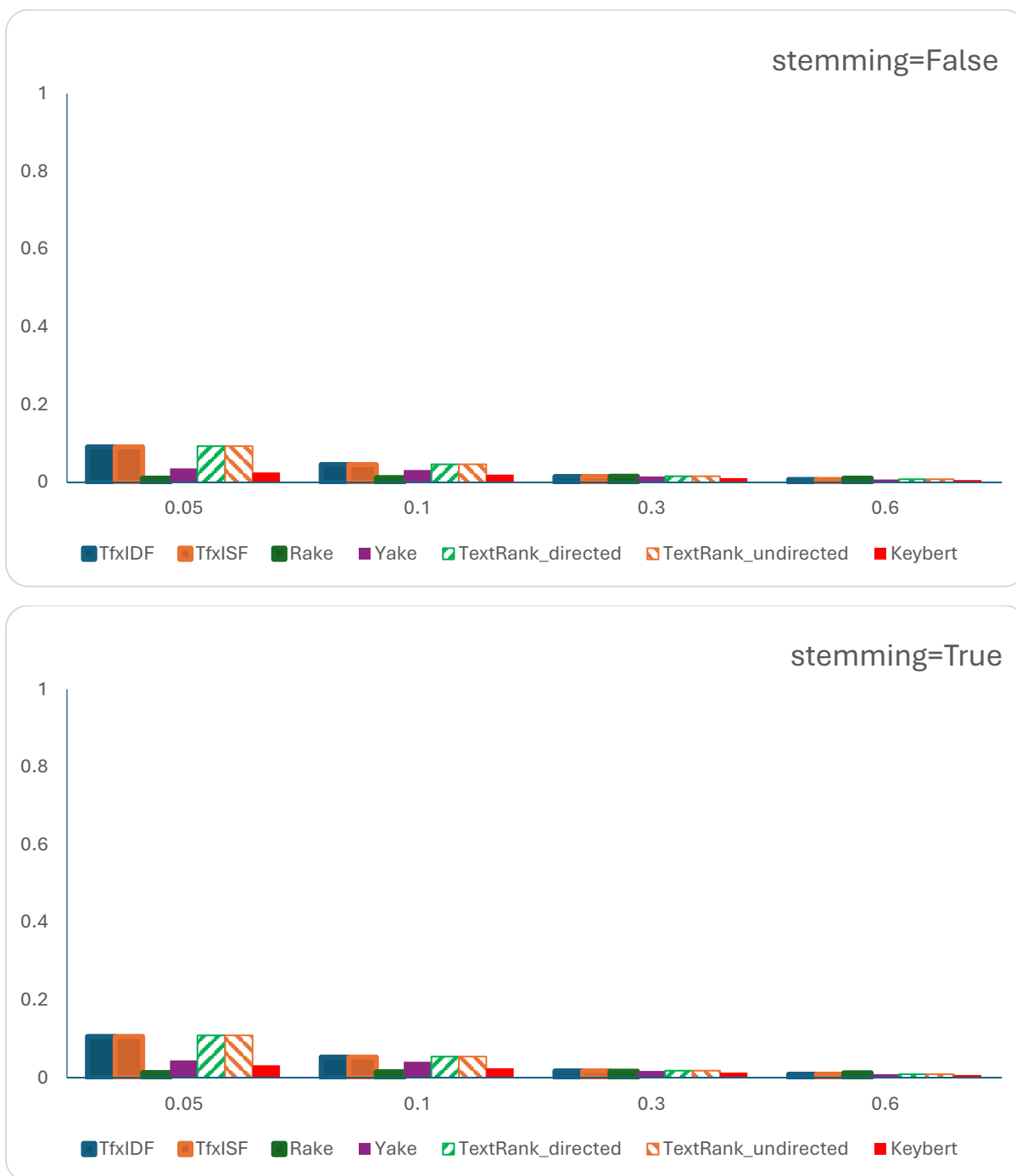


Σχήμα 4-4 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για n_grams=1 με stemming και χωρίς stemming

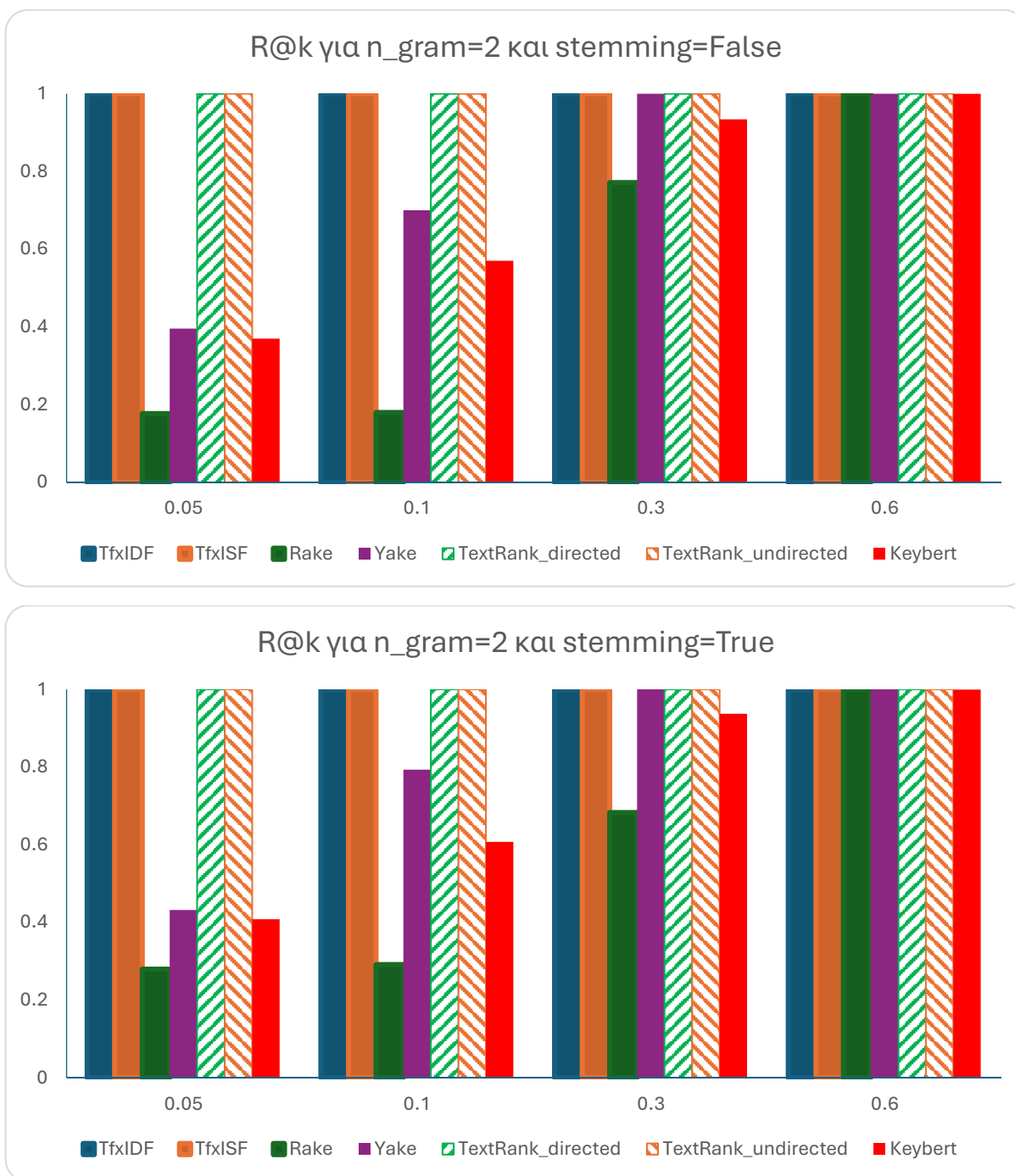
4.3.2 n_grams έως και 2

Η σύγκριση των αλγορίθμων ακολούθησε την ίδια πορεία με εκείνη που περιεγράφηκε για $n_grams=1$, με μόνη εξαίρεση τους αλγόριθμους TF οι οποίοι είχαν καλύτερη επίδοση σε σχέση με την περίπτωση των $n_grams=1$. Πιο συγκεκριμένα την καλύτερη επίδοση ως προς την ανάκληση παρουσίασαν οι αλγόριθμοι TEXTRANK και TF-IDF, TF-ISF η οποία ήταν σταθερά ίση με 1.0 σε όλα τα επίπεδα του k . Ωστόσο, κανένας από τους αλγόριθμους που εξετάστηκαν δεν ξεπέρασε την τιμή 0,2 για την μετρική της ακρίβειας, τιμή εμφανώς χαμηλότερη σε σχέση με εκείνη που παρατηρήθηκε για $n_grams=1$.

Τις καλύτερες επιδόσεις και με μεγάλη διαφορά από τους υπόλοιπους αλγόριθμους ως προς τη τιμή της μέσης ακρίβειας ($MAP > 0,7$) κατέγραψαν οι δύο υλοποιήσεις του TEXTRANK ακολουθούμενες από τους αλγόριθμους TF, ενώ οι υπόλοιποι αλγόριθμοι δεν ξεπέρασαν την τιμή 0,2. Αντίστοιχη πορεία ακολούθησαν οι αλγόριθμοι και για τις μετρικές $MnDCG$, με τις υλοποιήσεις του TEXTRANK να έχουν υψηλότερες τιμές σε σχέση με τους υπόλοιπους αλγόριθμους που εξετάστηκαν στην παρούσα μελέτη. Σε ότι αφορά στην μετρική MRR τις χειρότερες επιδόσεις ($MRR < 0,3$) κατέγραψαν οι KEYBERT και RAKE ανεξάρτητα εάν εφαρμοζόταν stemming ή όχι. Αντιθέτως οι αλγόριθμοι TEXTRANK (ανεξαρτήτως stemming), TF (ανεξαρτήτως stemming) και YAKE (χωρίς stemming) κατέγραψαν τιμές MRR κοντά στο 0,5, υποδηλώνοντας ότι η καλύτερη σειρά κατάταξης που μπορούσαν να προβλέψουν μια λέξη/φράση κλειδί ήταν στη δεύτερη θέση. Μεταξύ των υλοποιήσεων του TEXTRANK η υλοποίηση με κατευθυνόμενο γράφο φάνηκε και σε αυτή την περίπτωση να υπερέχει έναντι της υλοποίησης με μη κατευθυνόμενο γράφο. Επίσης, όπως και στην περίπτωση των $n_grams=1$, οι επιδόσεις των αλγορίθμων ήταν ελαφρώς αυξημένες στην περίπτωση του stemming σε σχέση με το όταν δεν λάμβανε χώρα stemming (μόνη εξαίρεση ο YAKE που σύμφωνα με την μετρική MRR οι επιδόσεις του ήταν καλύτερες όταν δεν εφαρμοζόταν stemming). Γενικότερα για όλους τους αλγόριθμους παρατηρήθηκε σταθερότητα ή οριακή αύξηση των επιδόσεων τους σε σχέση με την περίπτωση των $n_grams=1$. Τα αποτελέσματα των μετρήσεων φαίνονται στα διαγράμματα (σχήματα 4-5 έως και 4-8) που ακολουθούν.



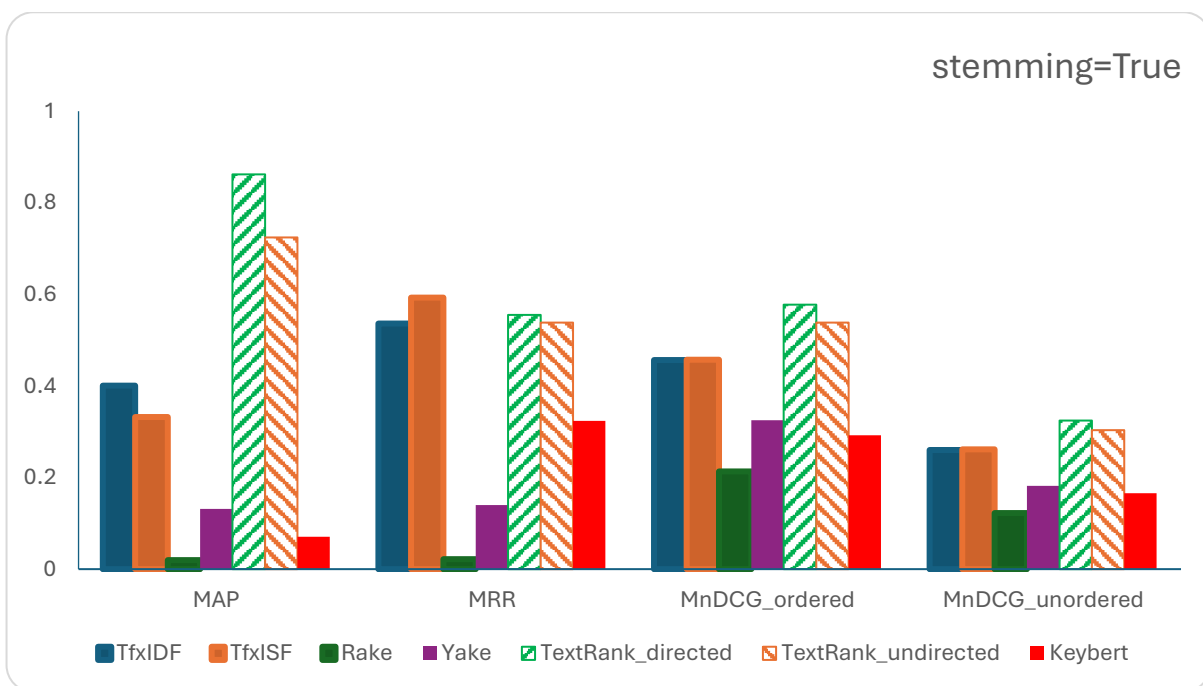
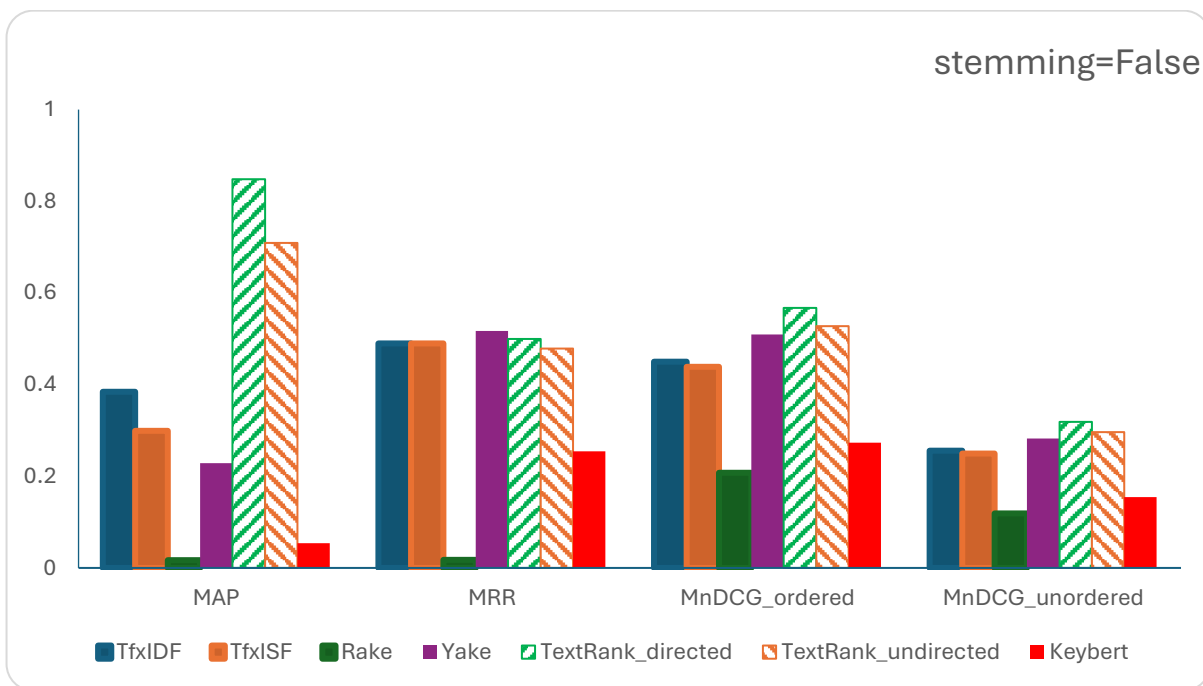
Σχήμα 4-5 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $P@k$, για $n_grams \leq 2$ με stemming και χωρίς stemming



Σχήμα 4-6 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής R@k, για n_grams ≤2 με stemming και χωρίς stemming



Σχήμα 4-7 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $F1@k$, για $n_grams \leq 2$ με stemming και χωρίς stemming

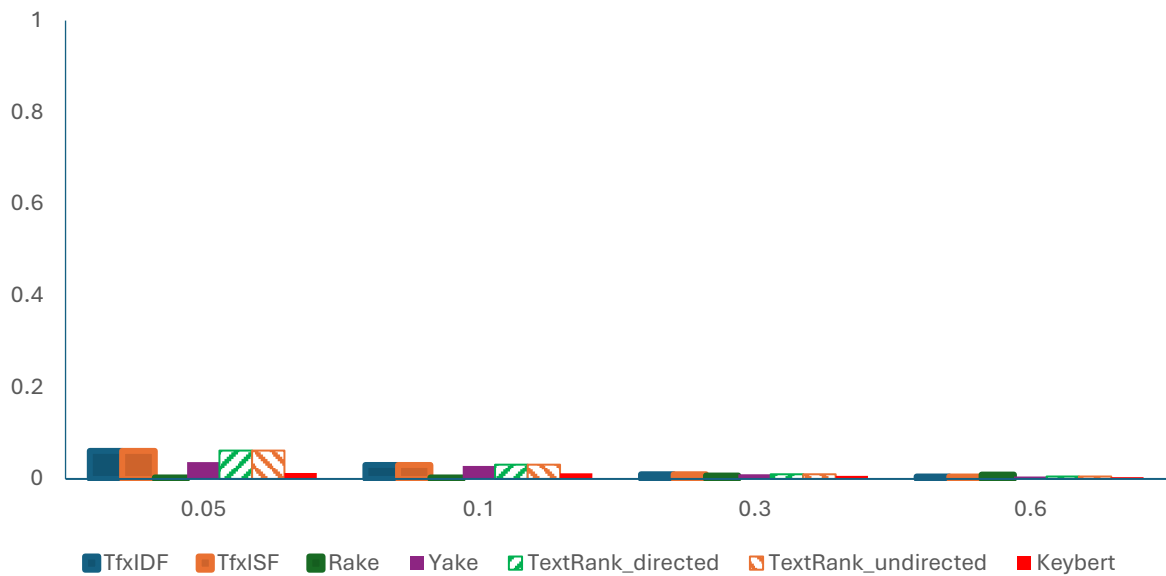


Σχήμα 4-8 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για $n_grams \leq 2$ με stemming και χωρίς stemming

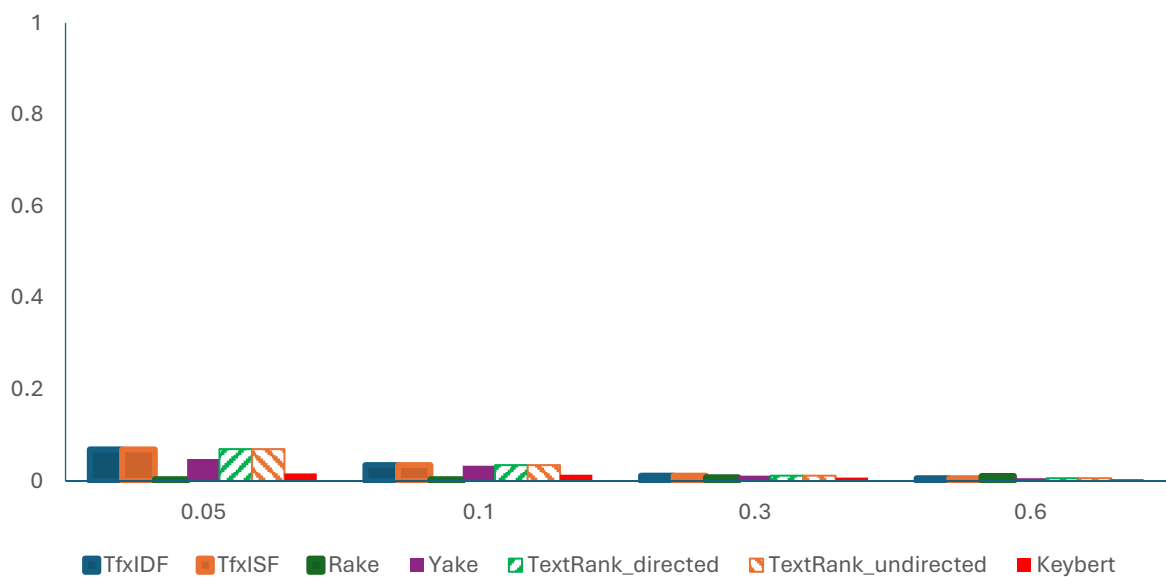
4.3.3 n_grams έως και 3

Δεν παρατηρήθηκαν κάποιες ιδιαίτερες μεταβολές στις επιδόσεις των αλγορίθμων σε σχέση με εκείνες που παρατηρήθηκαν για $n_grams=2$, με μόνη εξαίρεση την αύξηση των επιδόσεων του YAKE. Πιο συγκεκριμένα, για τον YAKE η τιμή της ανάκλασης αυξήθηκε κατά 20 μονάδες σε σχέση με την τιμή που παρατηρήθηκε για $n_grams=2$, ενώ τη μέγιστη τιμή της ($>0,94$) την έλαβε από πολύ νωρίς για τιμή του $k \geq 0.1$. Συνεπώς για $n_grams=3$, την καλύτερη επίδοση κατέγραψαν οι TEXTRANK (και οι δύο υλοποιήσεις), TF-IDF, TF-ISF και YAKE. Επίσης, όπως και στην περίπτωση των $n_grams=2$, οι επιδόσεις των αλγορίθμων ήταν ελαφρώς αυξημένες στην περίπτωση του stemming σε σχέση με το όταν δεν λάμβανε χώρα stemming, με μόνη εξαίρεση τον YAKE ο οποίος όπως και στην περίπτωση των $n_grams=2$ είχε καλύτερη επίδοση όταν δεν εφαρμοζόταν stemming ($MRR=0,51$) σε σχέση με όταν εφαρμοζόταν stemming ($MRR=0,13$). Τέλος, όπως και στις περιπτώσεις $n_grams=1$ και $n_grams=2$, οι RAKE και KEYBERT κατέγραψαν τις χαμηλότερες επιδόσεις σε σχέση με τους παραπάνω αλγόριθμους, ενώ η υλοποίηση του TEXTRANK με κατευθυνόμενο γράφο υπερείχε εκείνης με μη κατευθυνόμενο γράφο, εκτός από την καλύτερη πρόβλεψη της σειράς κατάταξης (μετρική MRR), που η υλοποίηση με μη κατευθυνόμενο γράφο ($MRR=0,46$) υπερείχε εκείνης με κατευθυνόμενο γράφο ($MRR=0,46$). Τα αποτελέσματα των μετρήσεων φαίνονται στα διαγράμματα (σχήματα 4-9 έως και 4-12) που ακολουθούν

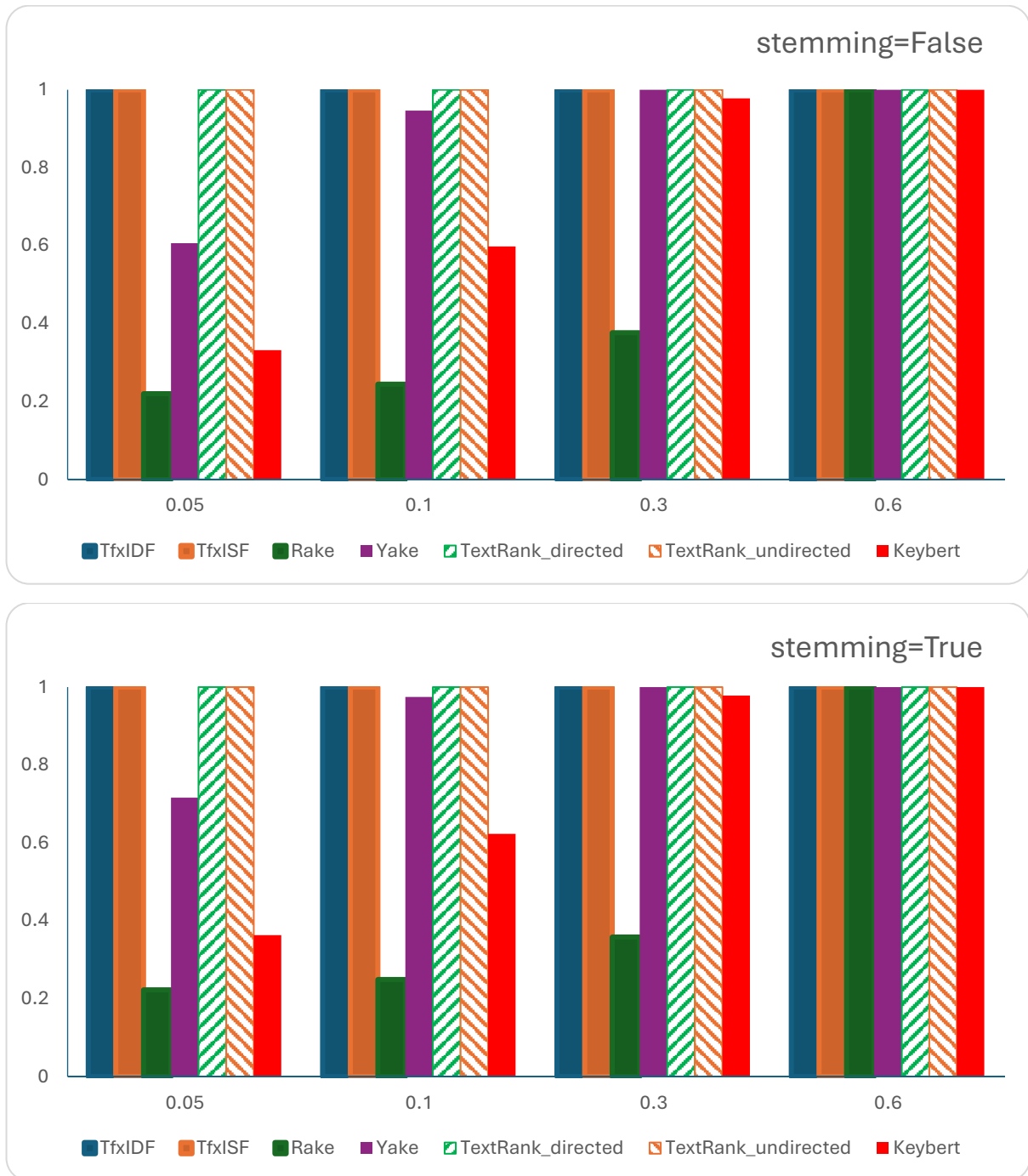
P@k για n_gram=3 και stemming=False



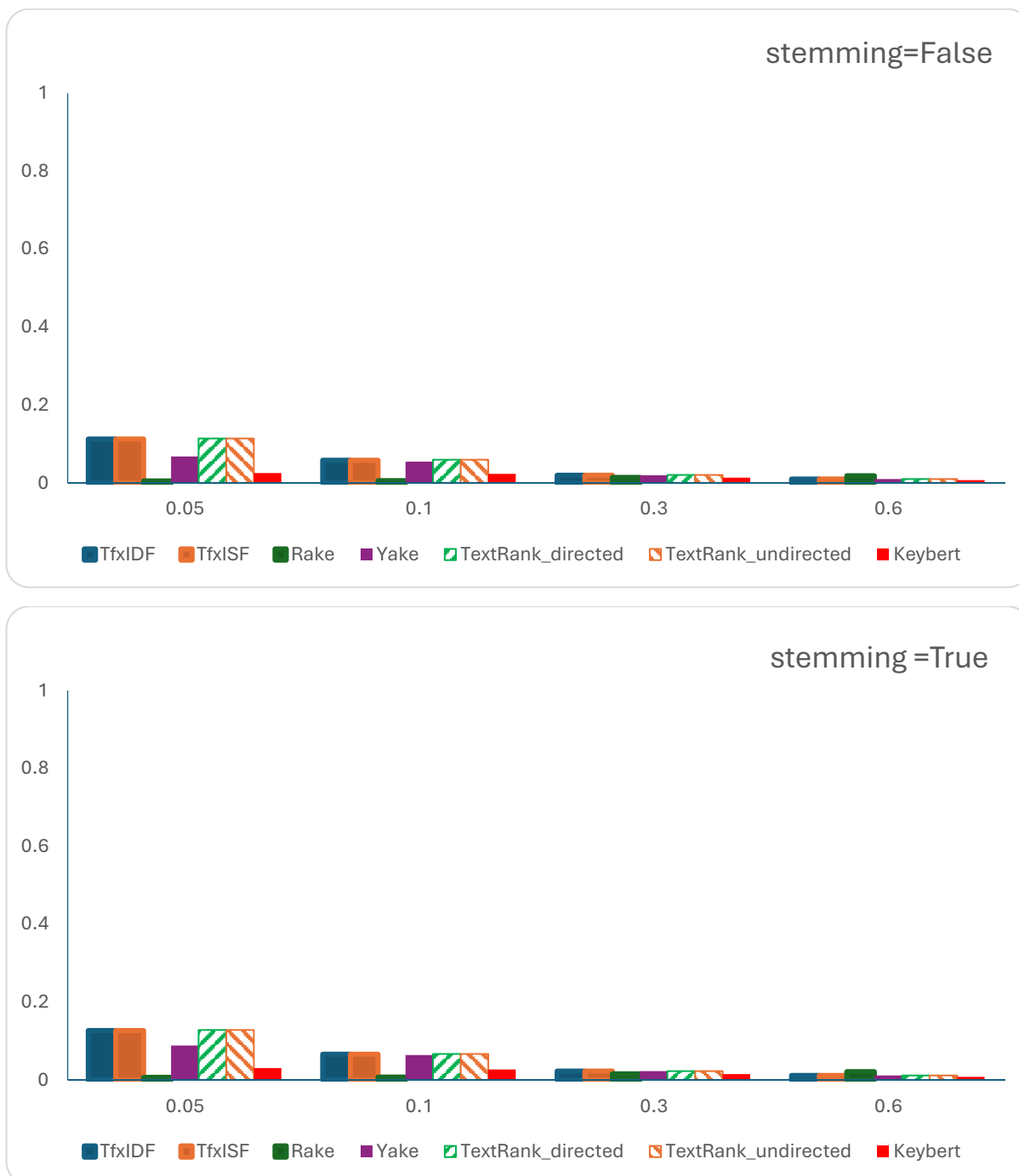
P@k για n_gram=3 και stemming=True



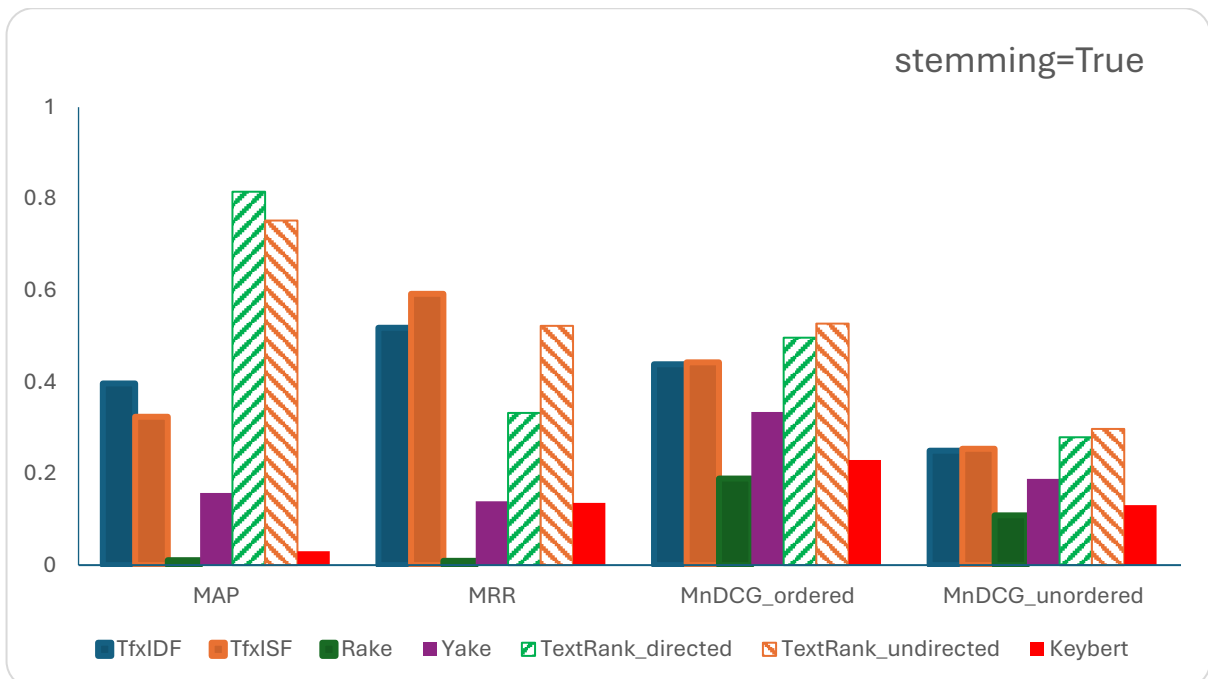
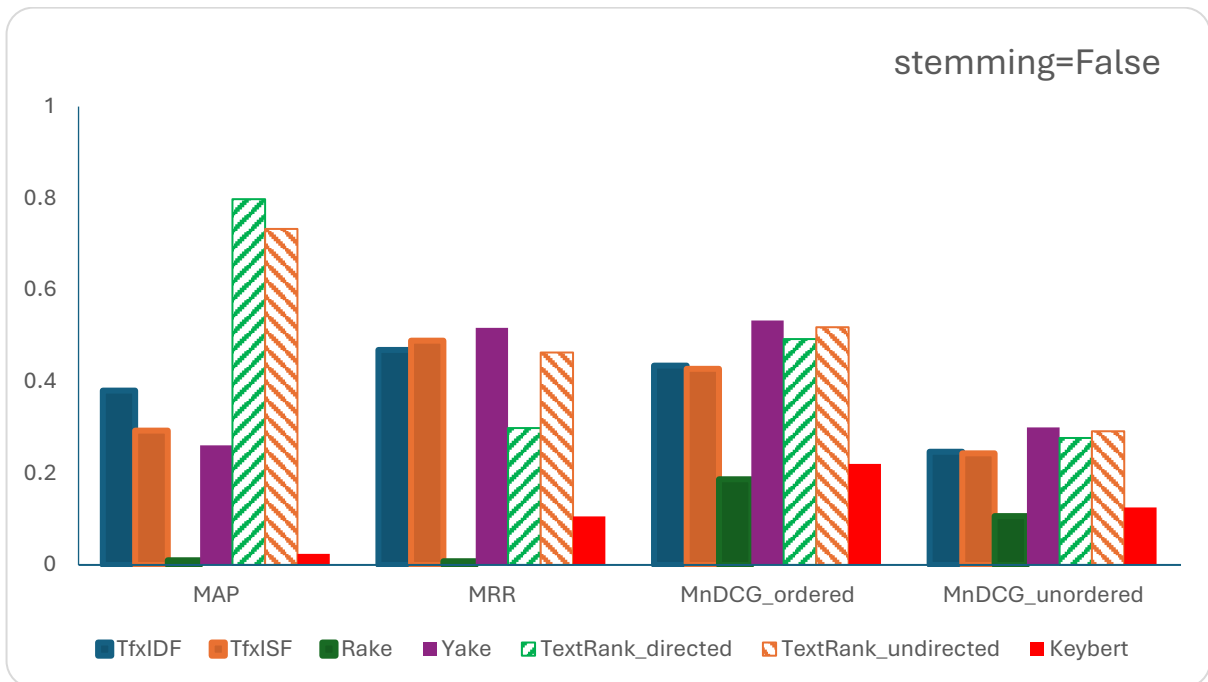
Σχήμα 4-9 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής P@k, για n_grams ≤ 3 με stemming και χωρίς stemming



Σχήμα 4-10 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $R@k$, για $n_grams \leq 3$ με stemming και χωρίς stemming



Σχήμα 4-11 Σύγκριση αλγορίθμων ως προς τις τιμές της μετρικής $F1@k$, για $n_grams \leq 3$ με stemming και χωρίς stemming



Σχήμα 4-12 Σύγκριση αλγορίθμων ως προς τις τιμές των μετρικών MAP, MRR, MnDCG (ordered / unordered), για $n_grams \leq 3$ με stemming και χωρίς stemming

4.3.4 Σύγκριση ταχύτητας εκτέλεσης των αλγορίθμων

Ως προς την ταχύτητα εκτέλεσης, ο KEYBERT ήταν ο περισσότερο αργός (5663ms και 5761 ms χωρίς και με stemming αντίστοιχα) και ο RAKE ο ταχύτερος (19ms και 43 ms χωρίς και με stemming αντίστοιχα) σε σχέση με τους υπόλοιπους αλγόριθμους. Επίσης, στους περισσότερο γρήγορους αλγόριθμους κατατάχθηκαν οι TF-ISF και η υλοποίηση του TEXTRANK με κατευθυνόμενο γράφο οι οποίοι δεν ξεπέρασαν κατά μέσο όρο τα 56ms προκειμένου να βαθμολογήσουν ένα πλήρες άρθρο της συλλογής Krapivn_2009, όταν δεν εφαρμοζόταν stemming. Σε ότι αφορά τους αλγόριθμους TF-IDF, TEXTRANK_undirected και YAKE η ταχύτητα εκτέλεσης τους ήταν τουλάχιστον 7πλάσια των προηγούμενων «γρήγορων» αλγορίθμων (RAKE, TF-ISF, TEXTRANK_directed). Με εξαίρεση τον TF-IDF, η ταχύτητα εκτέλεσης όλων των υπόλοιπων αλγορίθμων αυξήθηκε αρκετά στην περίπτωση που εφαρμοζόταν stemming. Χαρακτηριστικό παράδειγμα ο αλγόριθμος TEXTRANK_undirected όπου η μέση ταχύτητα εκτέλεσης αυξήθηκε από 56ms σε 1435ms όταν στο κείμενο εφαρμοζόταν stemming. Αντιθέτως η ταχύτητα εκτέλεσης του TF-IDF, ελαττώθηκε κατά 19% στην περίπτωση που εφαρμόστηκε stemming, σε σχέση με όταν δεν εφαρμοζόταν stemming. Οι μέσοι χρόνοι εκτέλεσης των αλγορίθμων να βαθμολογήσουν όλες τις λέξεις/φράσεις ($n_grams \leq 3$) ενός πλήρους άρθρου της συλλογής Krapivn_2009 φαίνονται στον Πίνακα 4-1.

	Μέσος χρόνος εκτέλεσης σε (ms)	
	stemming=False	stemming=True
TF-IDF	1304,25	1094,48
TF-ISF	26,57	428,51
RAKE	19,05	43,23
YAKE	441,93	762,23
TEXTRANK_directed	56,39	1435,03
TEXTRANK_undirected	1301,46	2563,29
KEYBERT	5663,39	5761,052

Πίνακας 4-1 Μέσοι χρόνοι εκτέλεσης των αλγορίθμων με stemming και χωρίς stemming για $n_grams \leq 3$

4.4 Συγκριτική αξιολόγηση αλγορίθμων στην ανάκτηση πληροφορίας

4.4.1 Μέθοδος σύγκρισης

Οι αλγόριθμοι αξιολογήθηκαν επίσης και ως προς την ικανότητα τους να ανακτούν πληροφορία, όταν χρησιμοποιούνται σε αποθηκευμένες συλλογές εγγράφων. Πιο συγκεκριμένα, αξιολογήθηκε η δυνατότητα των αλγορίθμων να προτείνουν επιστημονικά περιοδικά προς δημοσίευση όταν «διαβάσουν» ένα άρθρο προς δημοσίευση. Για το σκοπό αυτό χρησιμοποιήθηκε η συλλογή δεδομένων Aminer Citation Network (ACN), που αποτελείται από 2000 δημοσιευμένα άρθρα, και για κάθε άρθρο υπάρχουν οι παρακάτω πληροφορίες:

- ο τίτλος του
- το περιοδικό που δημοσιεύτηκε
- το κείμενο της περίληψης
- τα επιστημονικά πεδία που θεραπεύει ως λίστα πλειάδων της μορφής ('field', score), όπου score μια τιμή μεταξύ 0 και 1, που αντιπροσωπεύει την βαρύτητα κάθε πεδίου.

Μετά από κατάλληλη επεξεργασία των εγγράφων προέκυψαν 1821 επιλέξιμα άρθρα (δημοσιευμένα σε 1091 περιοδικά), καθώς 79 εγγραφές της αρχικής συλλογής δεν είχαν όλες τις παραπάνω πληροφορίες. Η συλλογή αποθηκεύτηκε στη μνήμη -υπό μορφή λεξικού της python (dictionary)- αφού πρώτα έλαβαν χώρα τα παρακάτω για κάθε επιλέξιμη εγγραφή:

- (α) Εφαρμόστηκε ο κάθε αλγόριθμος σε ένα ενιαίο κείμενο που συμπεριλάμβανε τον τίτλο και το κείμενο της περίληψης του άρθρου.
- (β) Ο κάθε αλγόριθμος εφαρμόστηκε για αριθμό `n_grams` έως 3 και με αφαίρεση των κοινότυπων λέξεων. Επίσης, ο κάθε αλγόριθμος εφαρμόστηκε με και χωρίς stemming. Με αυτό τον τρόπο το κείμενο κάθε αντιστοιχούσε σε σάκο λέξεων (BoW), όπου κάθε λέξη-φράση συνοδευόταν από την βαθμολογία που λάμβανε από κάθε αλγόριθμο.
- (γ) Για κάθε άρθρο επιλέχθηκαν τα επιστημονικά πεδία με βαθμολογία υψηλότερη του 0,45.

Ακολούθως έλαβε χώρα η αξιολόγηση των αλγορίθμων όταν διαβάσουν ένα άρθρο της συλλογής να προβλέψουν και να κατατάξουν στη καλύτερη θέση (α) το περιοδικό που είναι δημοσιευμένο και (β) τουλάχιστον ένα επιστημονικό πεδίο του άρθρου. Η αξιολόγηση των αλγορίθμων έγινε με τη χρήση της μετρικής MRR (η μέση τιμή του αντίστροφου της σειράς κατάταξης σύμφωνα με τις εξισώσεις 24 και 25), αφού κάθε άρθρο συγκρινόταν ως προς την ομοιότητα του με όλα τα υπόλοιπα άρθρα της συλλογής με χρήση ομοιότητας συνημίτονου

(cosine similarity). Αν και η κάθε άρθρο συγκρινόταν με όλα τα υπόλοιπα 1820 άρθρα της συλλογής, τα άρθρα που επιλέχθηκαν να λάβουν χώρα στις δοκιμές (test sample) ήταν εκείνα που ήταν δημοσιευμένα σε περιοδικό που είχε τουλάχιστον 2 δημοσιευμένα άρθρα (ώστε πάντα να υπάρχει το περιοδικό που είναι δημοσιευμένο το άρθρο εντός των περιοδικών που θα καταναείμουν οι αλγόριθμοι). Με βάση τον τελευταίο περιορισμό επιλέξιμα για χρήση στις δοκιμές ήταν 1069 από τα 1821 αποθηκευμένα άρθρα της συλλογής. Η κατάταξη των αποτελεσμάτων των συγκρίσεων, έγινε με φθίνουσα κατάταξη του βαθμού ομοιότητας σύμφωνα με την υπόθεση ότι δύο άρθρα με υψηλό βαθμό ομοιότητας είναι πολύ πιθανό (α) να είναι δημοσιευμένα στο ίδιο περιοδικό και (β) να έχουν τουλάχιστον ένα κοινό επιστημονικό πεδίο. Τέλος, πριν την έναρξη των μετρήσεων αξιολογήθηκε και η κατανομή των επιστημονικών πεδίων ανά περιοδικό. Από την κατανομή προέκυψε ότι το επικρατέστερο πεδίο ήταν το 'Computer science' που ανήκε σε 340 περιοδικά ενώ τα υπόλοιπα επιστημονικά πεδία κατανέμονταν μεταξύ 80 και 20 περιοδικών το κάθε ένα. Για να υπάρχει ομοιομορφία στις μετρήσεις, το επιστημονικό πεδίο 'Computer science' παραλείφθηκε από τις δοκιμές ομοιότητας επιστημονικού πεδίου.

4.4.2 Αποτελέσματα

Ως προς την ανάκτηση πληροφορίας και συγκεκριμένα ως προς τη δυνατότητα των αλγορίθμων να προβλέπουν το περιοδικό που είναι δημοσιευμένη μια εργασία οι αλγόριθμοι κατάφεραν να προβλέψουν το περιοδικό μεταξύ 10^{ης} και 37^{ης} θέσης κατά μέσο όρο. Την καλύτερη πρόβλεψη θέσης επέτυχε ο KEYBERT (MRR=0,106) ενώ η χειρότερη πρόβλεψη θέσης προήλθε από τον YAKE (MRR=0,024). Φάνηκε επίσης ότι σε ορισμένες περιπτώσεις όλοι οι αλγόριθμοι κατάφεραν να κατατάξουν το περιοδικό στην 1^η θέση (max MRR=1.0) ενώ υπήρξαν και περιπτώσεις που οι αλγόριθμοι κατέταξαν το περιοδικό στις τελευταίες θέσεις (>1000 από 1090 περιοδικά). Επίσης, η μετρική MRR δεν φάνηκε να επηρεάζεται από την εφαρμογή stemming στις περιλήψεις των περιοδικών που έλαβαν χώρα οι μετρήσεις. Τα αποτελέσματα των μετρήσεων ανά αλγόριθμο χωρίς stemming και με stemming φαίνονται στους Πίνακες 4-2 και 4-3.

Σε αντίθεση με την ικανότητα των αλγορίθμων να προβλέπουν το περιοδικό, η ικανότητα όλων των αλγορίθμων να προβλέπουν το επιστημονικό πεδίο της εργασίας ήταν κατά πολύ αυξημένη. Έτσι η χαμηλότερη τιμή της μετρικής MRR ως προς το επιστημονικό πεδίο ήταν 0,46, που αντιστοιχεί σε θέση κατάταξης μεταξύ 2 και 3 (μέση θέση = 2,17).

Αντίστοιχα με την πρόβλεψη των περιοδικών ο αλγόριθμος KEYBERT κατάφερε να προβλέψει το επιστημονικό πεδίο ενός άρθρου καλύτερα από τους υπόλοιπους αλγορίθμους κατατάσσοντας το κυρίως στη 1^η θέση και σε ορισμένες περιπτώσεις στη 2^η (MRR=0,89) κατά μέσο όρο, ενώ ο YAKE είχε τη χειρότερη επίδοση κατατάσσοντας το επιστημονικό πεδίο του περιοδικού μεταξύ 2^{ης} και 3^{ης} θέσης. Η σειρά κατάταξης των επιστημονικών πεδίων από τους αλγόριθμους δεν φάνηκε να επηρεάζεται από το εάν εφαρμόστηκε stemming ή όχι στο κείμενο των περιλήψεων. Τα αποτελέσματα των μετρήσεων ανά αλγόριθμο χωρίς stemming και με stemming φαίνονται στους Πίνακες 4-4 και 4-5.

	Μετρική MRR		
	Μέση τιμή	Ελάχιστη τιμή	Μέγιστη τιμή
TF-IDF	0,049	> 0,001	1,0
TF-ISF	0,067	> 0,001	1,0
RAKE	0,083	> 0,001	1,0
YAKE	0,027	> 0,001	1,0
TEXTRANK_directed	0,085	> 0,001	1,0
TEXTRANK_undirected	0,086	> 0,001	1,0
KEYBERT	0,106	> 0,001	1,0

Πίνακας 4-2 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων χωρίς stemming να προβλέπουν το περιοδικό δημοσίευσης μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)

	Μετρική MRR		
	Μέση τιμή	Ελάχιστη τιμή	Μέγιστη τιμή
TF-IDF	0,048	> 0,001	1,0
TF-ISF	0,068	> 0,001	1,0
RAKE	0,076	> 0,001	1,0
YAKE	0,024	> 0,001	1,0
TEXTRANK_directed	0,071	> 0,001	1,0
TEXTRANK_undirected	0,078	> 0,001	1,0
KEYBERT	0,104	> 0,001	1,0

Πίνακας 4-3 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων με stemming να προβλέπουν το περιοδικό δημοσίευσης μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)

Μετρική MRR			
	Μέση τιμή	Ελάχιστη τιμή	Μέγιστη τιμή
TF-IDF	0,602	> 0,01	1,0
TF-ISF	0,796	> 0,01	1,0
RAKE	0,737	> 0,01	1,0
YAKE	0,481	> 0,01	1,0
TEXTRANK_directed	0,776	> 0,01	1,0
TEXTRANK_undirected	0,791	> 0,01	1,0
KEYBERT	0,881	> 0,01	1,0

Πίνακας 4-4 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων χωρίς stemming να προβλέπουν το επιστημονικό πεδίο μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)

Μετρική MRR			
	Μέση τιμή	Ελάχιστη τιμή	Μέγιστη τιμή
TF-IDF	0,596	> 0,01	1,0
TF-ISF	0,753	> 0,01	1,0
RAKE	0,729	> 0,01	1,0
YAKE	0,461	> 0,01	1,0
TEXTRANK_directed	0,750	> 0,01	1,0
TEXTRANK_undirected	0,778	> 0,01	1,0
KEYBERT	0,889	> 0,01	1,0

Πίνακας 4-5 Μέγιστη, ελάχιστη και μέση τιμή της μετρικής MRR στην ικανότητα των αλγορίθμων με stemming να προβλέπουν το επιστημονικό πεδίο μιας εργασίας από την περίληψη της (μέγεθος δείγματος 1069 περιλήψεις)

5 Ανάπτυξη εφαρμογής συστάσεων περιοδικών

5.1 Γενική περιγραφή της εφαρμογής

Σε αυτό το μέρος θα παρουσιαστεί η ανάπτυξη μιας εφαρμογής που θα κάνει επίδειξη της ικανότητας των αλγορίθμων που παρουσιάστηκαν στο προηγούμενο μέρος να προβλέπουν το περιοδικό προς δημοσίευση ενός άρθρου. Συγκεκριμένα ο χρήστης θα ανεβάζει στην εφαρμογή το κείμενο του άρθρου (.docx, .pdf, .txt) και με τη χρήση ενός από τους παραπάνω αλγόριθμους η εφαρμογή θα προτείνει στο χρήστη κατάλληλα περιοδικά στα οποία θα μπορούσε να δημοσιευτεί το άρθρο του. Τα περιοδικά καθώς και οποιαδήποτε σχετική πληροφορία θα είναι αποθηκευμένα σε βάση δεδομένων (ΒΔ), ενώ η εφαρμογή θα κάνει χρήση του ίδιου αλγορίθμου, τόσο για την ανάκτηση πληροφορίας από τη βάση, όσο και για εγγραφή κάποιας πληροφορίας σε αυτή (π.χ. δημοσιευμένες εργασίες περιοδικών). Η εφαρμογή έχει αναπτυχθεί ώστε να λειτουργεί με έναν μόνο αλγόριθμο από αυτούς που παρουσιάστηκαν στο προηγούμενο μέρος, συνεπώς η επιλογή του αλγόριθμου θα πρέπει να γίνει από τον προγραμματιστή πριν αρχίσει να λαμβάνει χώρα η αποθήκευση δεδομένων περιοδικών/δημοσιευμένων εργασιών στη ΒΔ. Ωστόσο, ο προγραμματιστής -εκτός εφαρμογής- μπορεί μέσω των παραμέτρων της εφαρμογής (αρχείο config.json) να επιλέξει τον αλγόριθμο που επιθυμεί και ακολούθως -μέσω της εφαρμογής- να αρχίσει να αποθηκεύει πληροφορία στην κενή ΒΔ. Αλλαγή του αλγορίθμου ενώ η ΒΔ δεν είναι κενή -αν και μπορεί να γίνει τεχνικά μέσω απευθείας αλλαγής των στοιχείων του config.json- απαγορεύεται γιατί η εφαρμογή ενδέχεται να δίνει λανθασμένα αποτελέσματα.

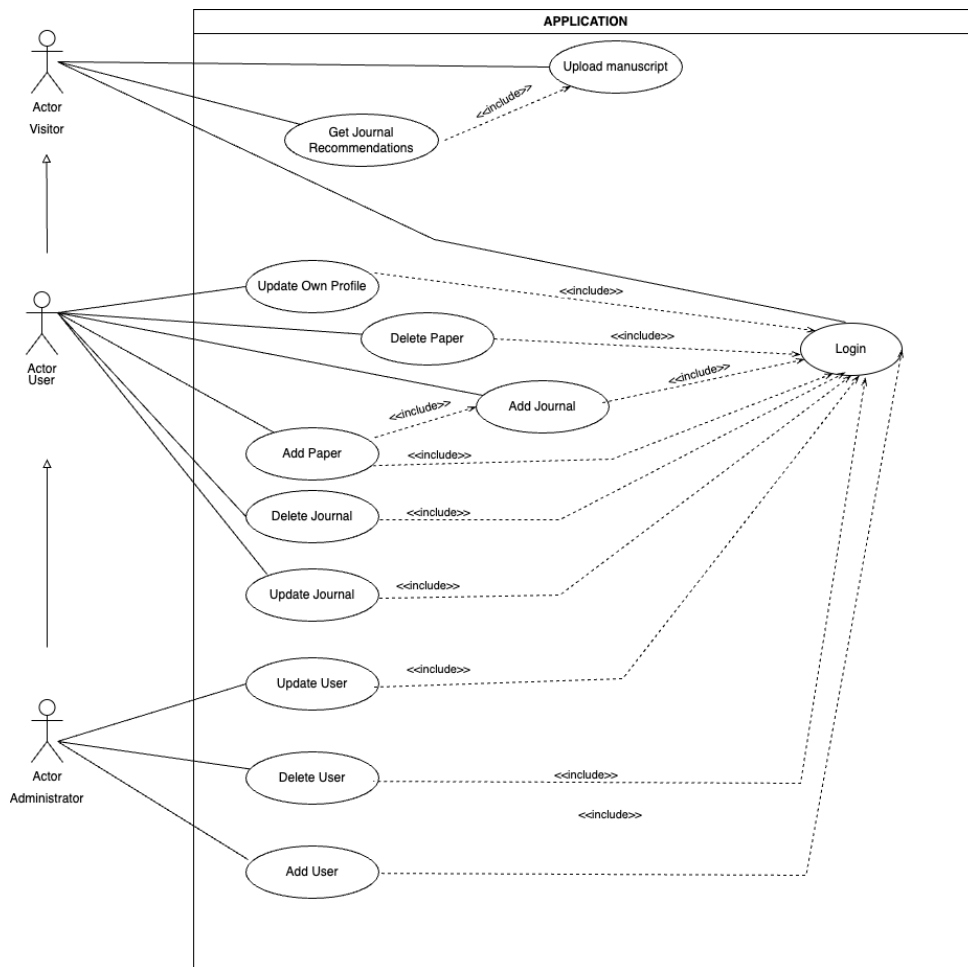
5.2 Περιγραφή του πλαισίου ανάπτυξης (framework) της εφαρμογής

Η εφαρμογή αναπτύχθηκε σε γλώσσα προγραμματισμού Python v.3.13 και με τη χρήση του πλαισίου ανάπτυξης (framework) ανοιχτού κώδικα Streamlit v.1.54.0 (<https://docs.streamlit.io>). Το Streamlit είναι ένα framework που έχει σχεδιαστεί για τη δημιουργία διαδραστικών εφαρμογών με έμφαση στην επιστήμη της μηχανικής μάθησης. Επιτρέπει στους προγραμματιστές να μετατρέπουν τα αρχεία python σε πλήρως λειτουργικές web εφαρμογές χωρίς να απαιτείται η γνώση χρήσης front-end εφαρμογών, μειώνοντας αισθητά το χρόνο υλοποίησης σε σχέση με τα παραδοσιακά web frameworks. Επίσης η βάση δεδομένων υλοποιήθηκε σε SQLITE v.3 η οποία είναι απόλυτα συμβατή με τη Python v.3.13.

5.3 Περιγραφή της αρχιτεκτονικής της εφαρμογής

5.3.1 Διάγραμμα περιπτώσεων χρήσης

Η εφαρμογή αποτελείται από μια περίπτωση χρήσης (ΠΧ) με όνομα “Application”. Σε αυτή την ΠΧ διακρίνονται 3 χρήστες με διαβαθμισμένη πρόσβαση ο κάθε ένας στις δυνατότητες της εφαρμογής. Ο κατώτερος χρήστης είναι ο επισκέπτης (visitor) που μπορεί να ανεβάσει τις εργασίες του (manuscripts) και να λάβει τις συστάσεις για περιοδικά προς δημοσίευση. Ένας visitor αφού συνδεθεί στην εφαρμογή έχει και επιπλέον δυνατότητες. Εάν συνδεθεί ως εγγεγραμμένος χρήστης (registered user), μπορεί επιπλέον να ενημερώσει τη ΒΔ ανεβάζοντας δημοσιευμένες εργασίες ή να ενημερώσει το προφίλ του. Εάν ο visitor συνδεθεί ως διαχειριστής (administrator), τότε επιπλέον των δυνατοτήτων του εγγεγραμμένου χρήστη μπορεί επίσης, να δημιουργήσει ένα νέο χρήστη από τη ΒΔ, να διαγράψει ένα υφιστάμενο χρήστη από τη ΒΔ, να ενημερώσει το προφίλ κάποιου χρήστη και να διαγράψει ένα περιοδικό από τη ΒΔ. Το διάγραμμα της περίπτωσης χρήσης “Application” φαίνεται στο παρακάτω σχήμα (Σχήμα 5-1).



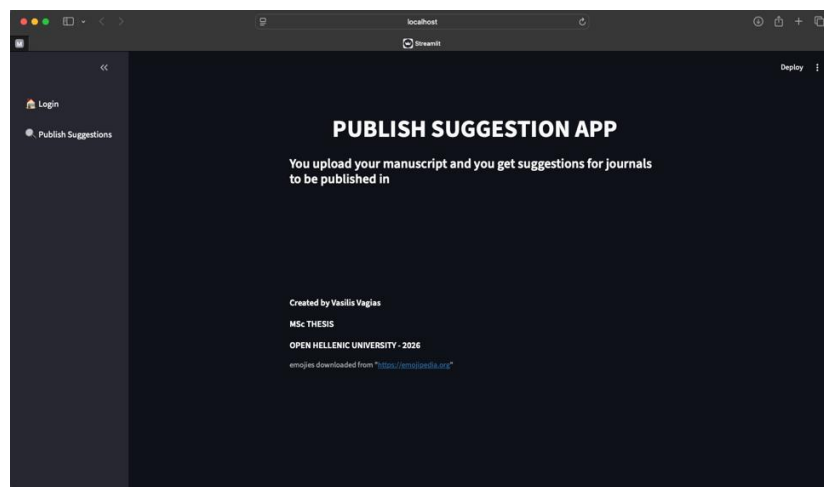
Σχήμα 5-1 Διάγραμμα Περιπτώσεων Χρήσης Application

5.3.2 Οθόνες (Views)

Το διάγραμμα περιπτώσεων χρήσης του σχήματος 1, αποτυπώνεται μέσω των οθονών διεπαφής του χρήστη με την εφαρμογή.

5.3.2.1 Οθόνη καλωσορίσματος (όλοι οι χρήστες)

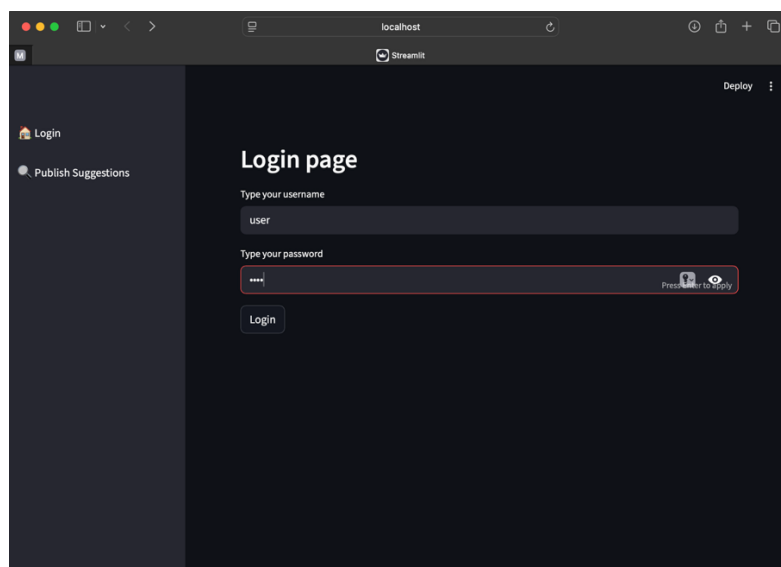
Η συγκεκριμένη οθόνη είναι η οθόνη καλωσορίσματος (landing page) και η πρώτη οθόνη που βλέπει ο επισκέπτης προκειμένου να πλοηγηθεί στην εφαρμογή. Η μπάρα πλοήγησης έχει δύο κουμπιά. Την σύνδεση του επισκέπτη με την εφαρμογή και την χρήση της εφαρμογής ως προς το να λαμβάνει συστάσεις περιοδικών για να δημοσιεύσει τις εργασίες του.



Σχήμα 5-2 Οθόνη καλωσορίσματος

5.3.2.2 Οθόνη Login (όλοι οι χρήστες)

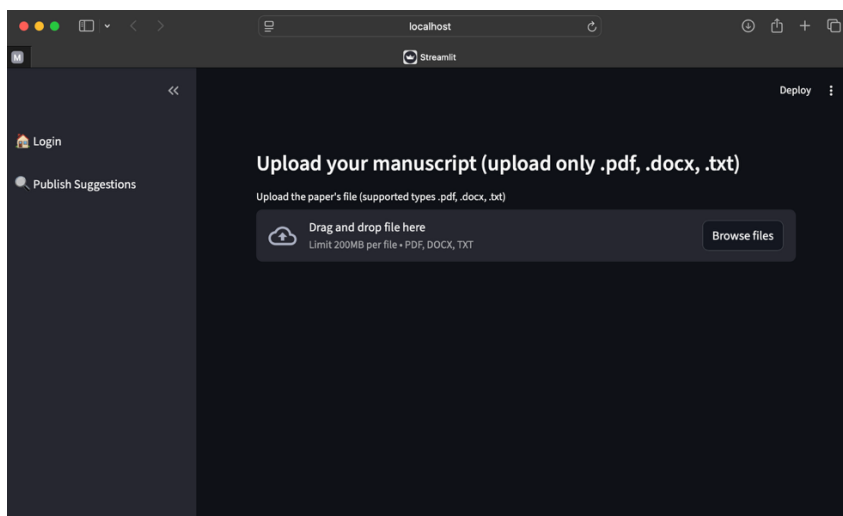
Σε αυτή την οθόνη ο χρήστης βάζει το username και το password και ανάλογα τα στοιχεία του ανατίθεται ο ρόλος. Ανάλογα με το εάν είναι εγγεγραμμένος χρήστης ή όχι θα δει κάποιο σχετικό μήνυμα, ενώ η μπάρα πλοήγησης θα έχει τα αντίστοιχα κουμπιά πλοήγησης (δες παρακάτω οθόνες) .



Σχήμα 5-3 Οθόνη εγγραφής χρήστη

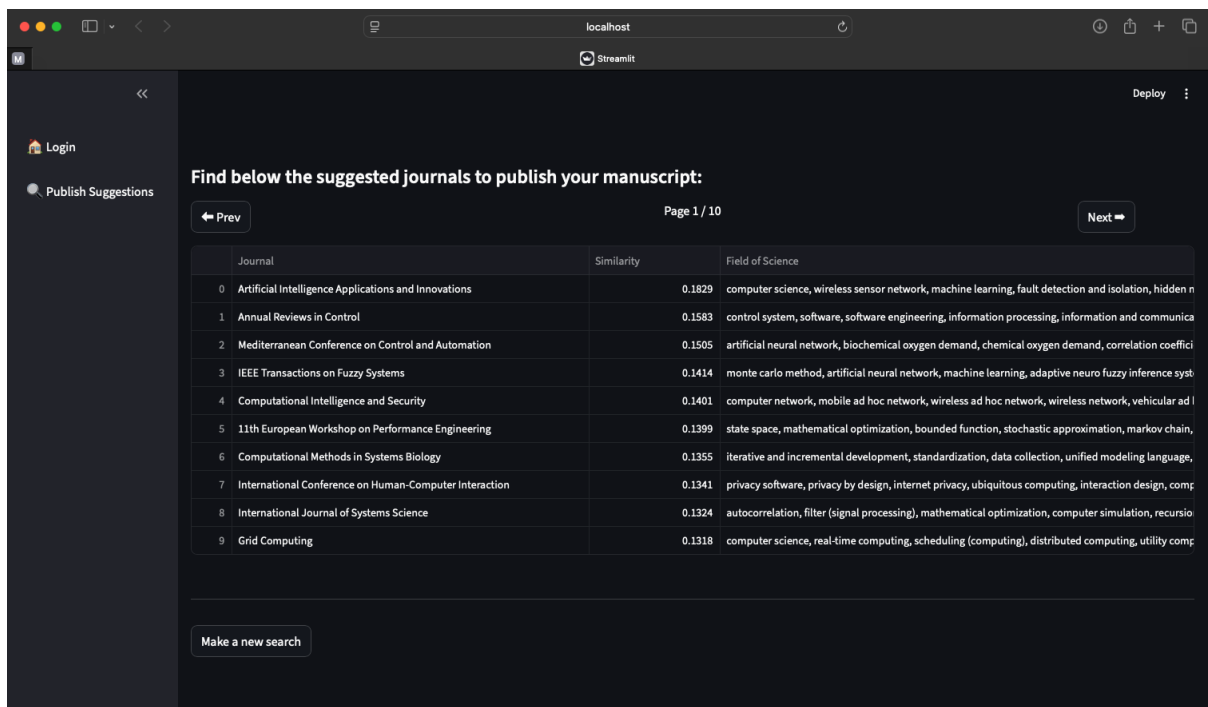
5.3.2.3 Οθόνη Ανάκτησης Συστάσεων περιοδικών για δημοσίευση (όλοι οι χρήστες)

Σε αυτή την οθόνη, ο χρήστης ανεβάζει την εργασία του, η οποία διαβάζεται από το σύστημα και βαθμολογείται ως προς την συνάφεια της με τα αποθηκευμένα άρθρα στη βάση



Σχήμα 5-4 Οθόνη προτροπής του χρήστη να ανεβάσει το άρθρο του

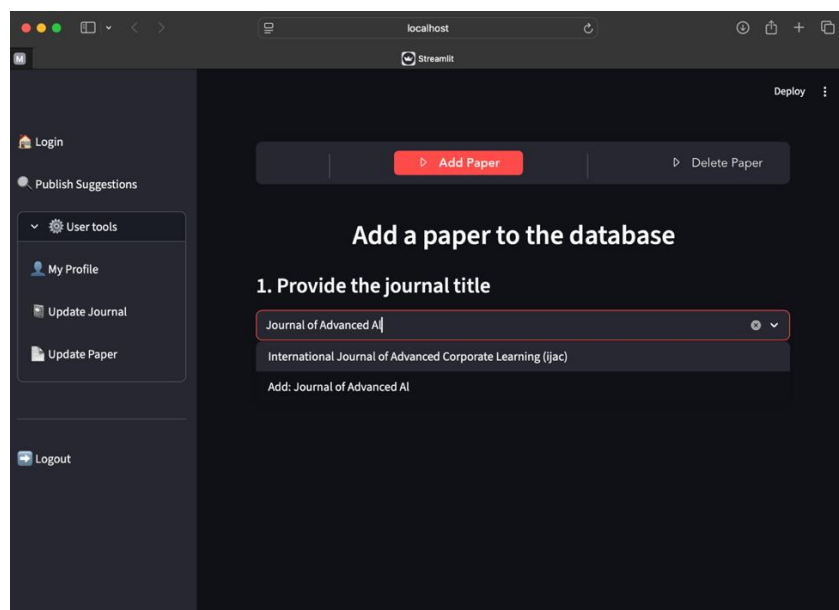
Ακολουθώντας τα πιο σχετικά περιοδικά και τα επιστημονικά πεδία που θεραπεύει κάθε ένα από αυτά παρουσιάζονται στην οθόνη του χρήστη κατά φθίνουσα σειρά εγγύτητας ως προς την εργασία που ανέβασε ο χρήστης.



Σχήμα 5-5 Οθόνη παρουσίασης στον χρήστη των σχετικών περιοδικών

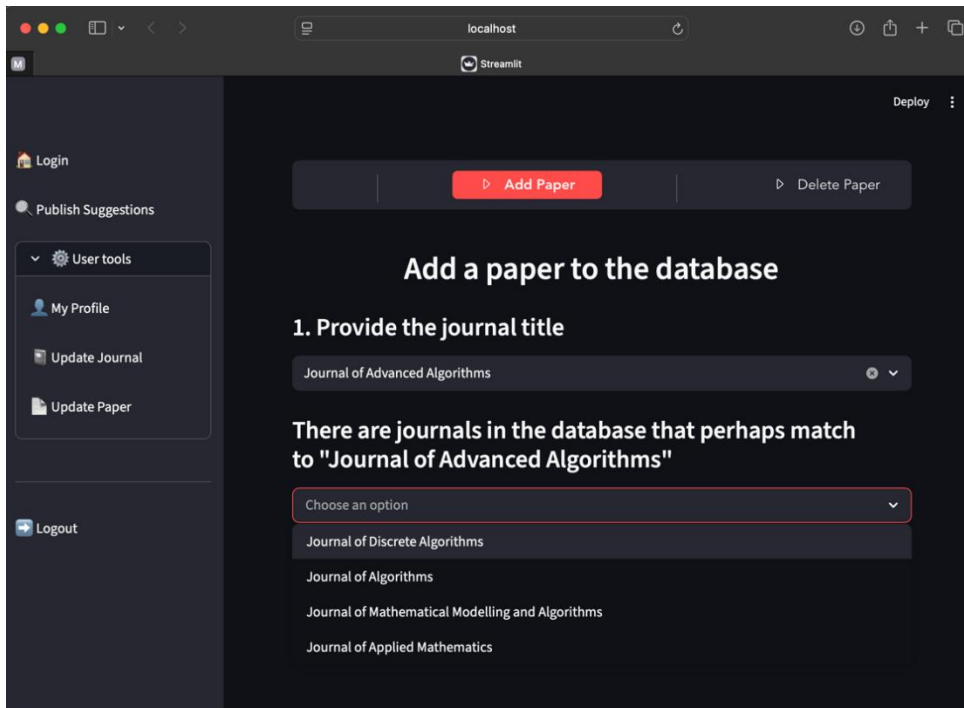
5.3.2.4 Οθόνη Προσθήκης άρθρου (εγγεγραμμένος χρήστης και διαχειριστής)

Η συγκεκριμένη οθόνη είναι διαθέσιμη από το κουμπί “Update Paper” της μπάρας πλοήγησης. Σε αυτή την οθόνη ο χρήστης καλείται πρώτα να εισάγει το περιοδικό που είναι δημοσιευμένο το άρθρο. Το σύστημα μέσω live select box φιλτράρει τα περιοδικά που είναι αποθηκευμένα στη ΒΔ ώστε να τον διευκολύνει να επιλέξει κάποιο περιοδικό που υπάρχει ήδη στη ΒΔ



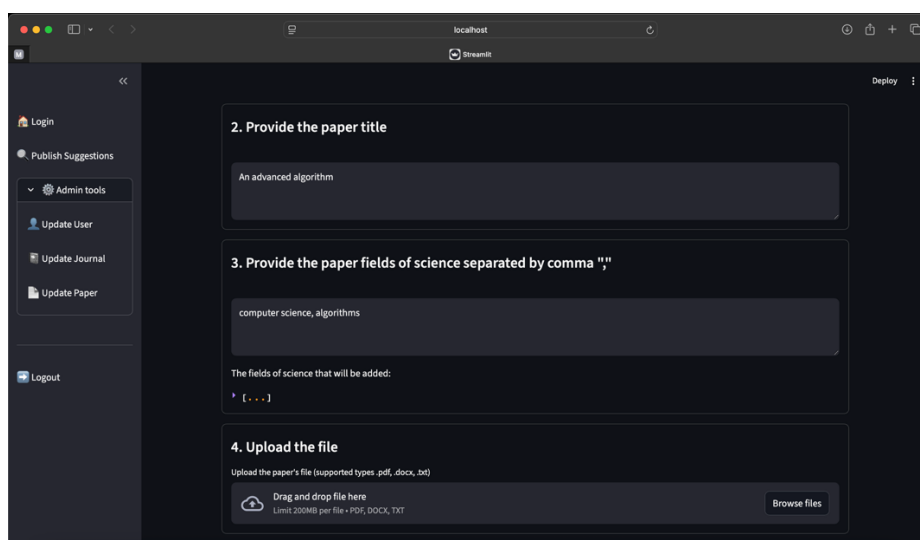
Σχήμα 5-6 Οθόνη εισαγωγής νέου άρθρου στη βάση – Εισαγωγή περιοδικού

Στην περίπτωση που ο χρήστης δώσει κάποιο περιοδικό που δεν υπάρχει στη ΒΔ, και για την αποφυγή λανθασμένης εισόδου του χρήστη το σύστημα προειδοποιεί το χρήστη εάν στη ΒΔ περιέχει ήδη το περιοδικό που δήλωσε με κάποιο άλλο όνομα.



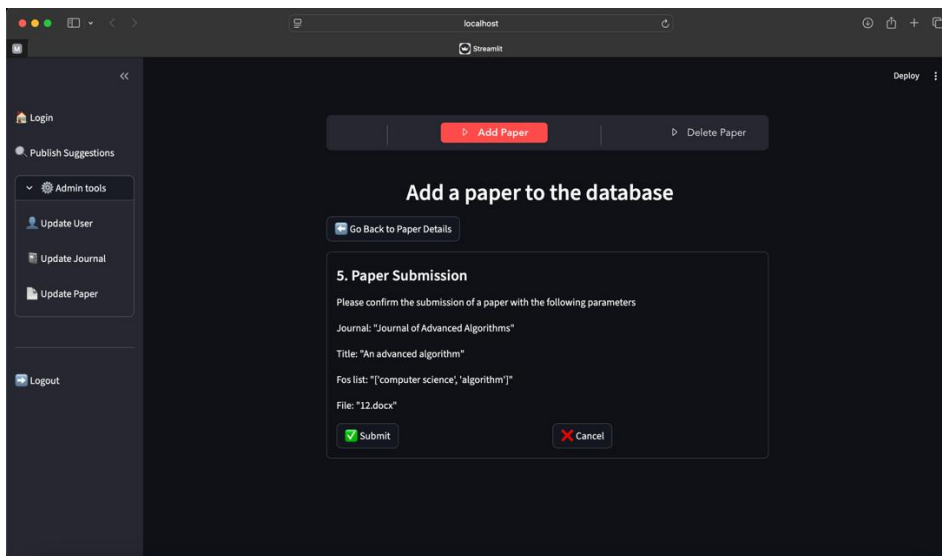
Σχήμα 5-7 Οθόνη εισαγωγής νέου άρθρου στη βάση – Έλεγχος ύπαρξης περιοδικού

Ακολούθως η οθόνη μεταφέρεται στην επιλογή του τίτλου του άρθρου, των επιστημονικών πεδίων που θεραπεύει και στο ανέβασμα του αρχείου κειμένου του στις υποστηριζόμενες μορφές (.pdf, .docx, .txt).



Σχήμα 5-8 Οθόνη εισαγωγής νέου άρθρου στη βάση – Προσθήκη στοιχείων άρθρου

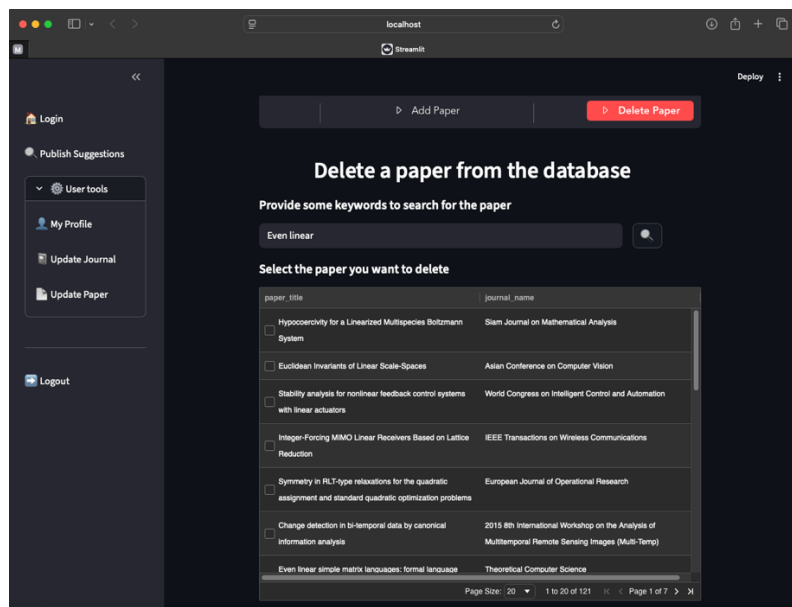
Τέλος, το άρθρο αποθηκεύεται στο σύστημα μετά από σχετικό μήνυμα επιβεβαίωσης.



Σχήμα 5-9 Οθόνη εισαγωγής νέου άρθρου στη βάση – Επιβεβαίωση

5.3.2.5 Οθόνη Διαγραφής άρθρου (εγγεγραμμένος χρήστης και διαχειριστής)

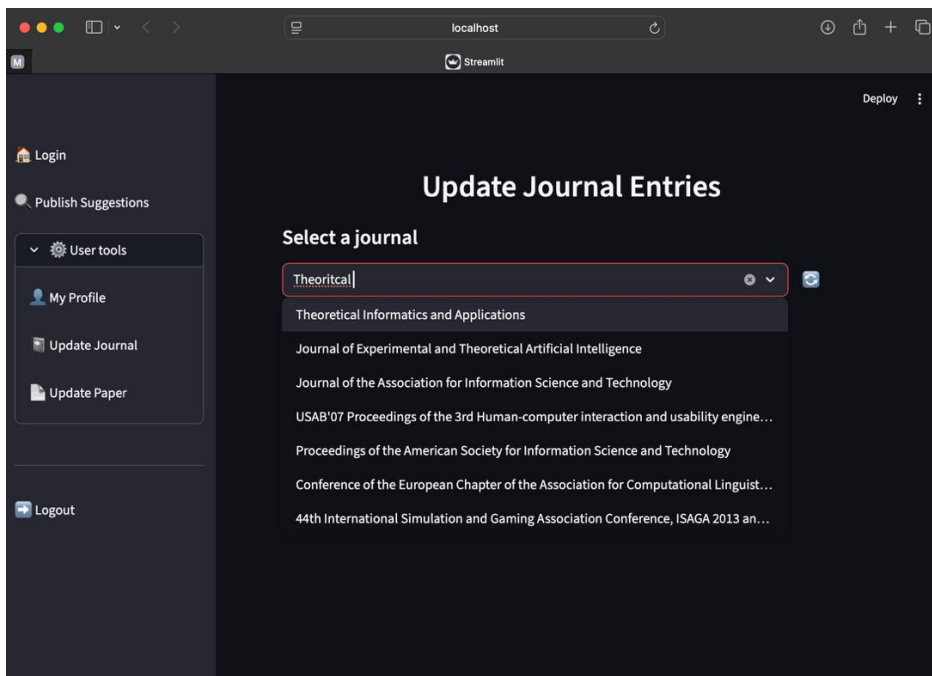
Η συγκεκριμένη οθόνη είναι επίσης διαθέσιμη από το κουμπί “Update Paper” της μπάρας πλοήγησης. Σε αυτή την οθόνη ο χρήστης δίνει κάποιες λέξεις κλειδιά και η ΒΔ ανασύρει τα πιο σχετικά περιοδικά κατά φθίνουσα σειρά, προκειμένου διευκολυνθεί ο χρήστης να επιλέξει το περιοδικό που επιθυμεί να διαγράψει από τη ΒΔ, μετά από σχετικό μήνυμα επιβεβαίωσης.



Σχήμα 5-10 Οθόνη διαγραφής περιοδικού από τη βάση

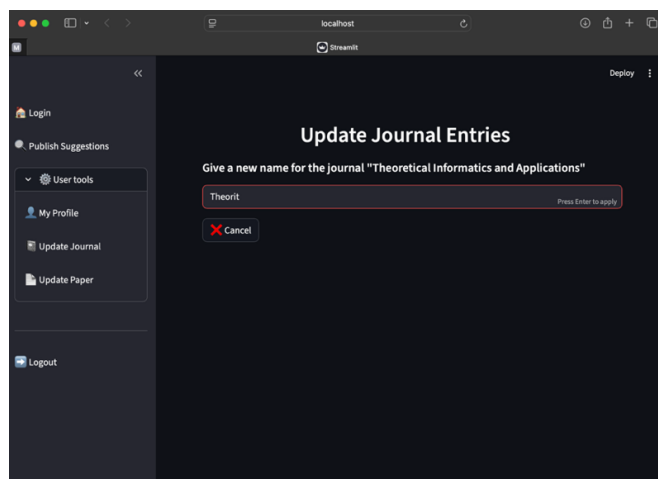
5.3.2.6 Οθόνη Επικαιροποίησης Περιοδικού (εγγεγραμμένος χρήστης και διαχειριστής)

Η συγκεκριμένη οθόνη είναι διαθέσιμη από το κουμπί “Update Journal” της μπάρας πλοήγησης και αρχικά προτρέπει το χρήστη να επιλέξει ένα περιοδικό μέσω live select box.

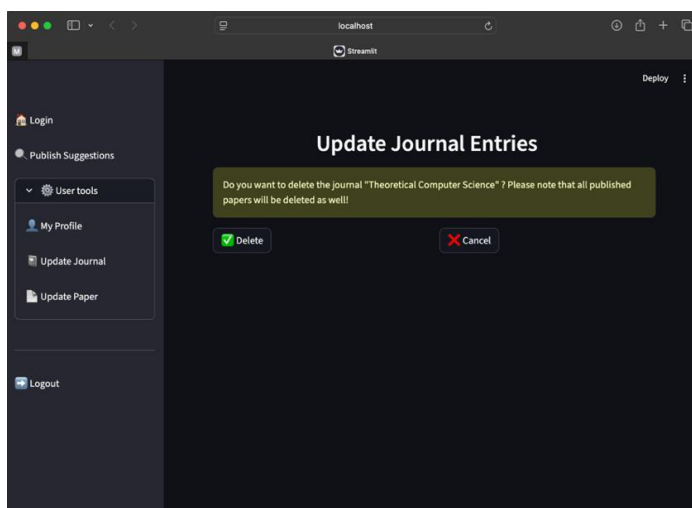


Σχήμα 5-11 Οθόνη επικαιροποίησης περιοδικού από τη βάση

Αφού επιλέξει το περιοδικό ο χρήστης, το σύστημα ακολούθως του δίνει δύο επιλογές. Να αλλάξει το όνομα ενός περιοδικού ή να το διαγράψει από τη βάση (μετά από σχετικό μήνυμα επιβεβαίωσης).



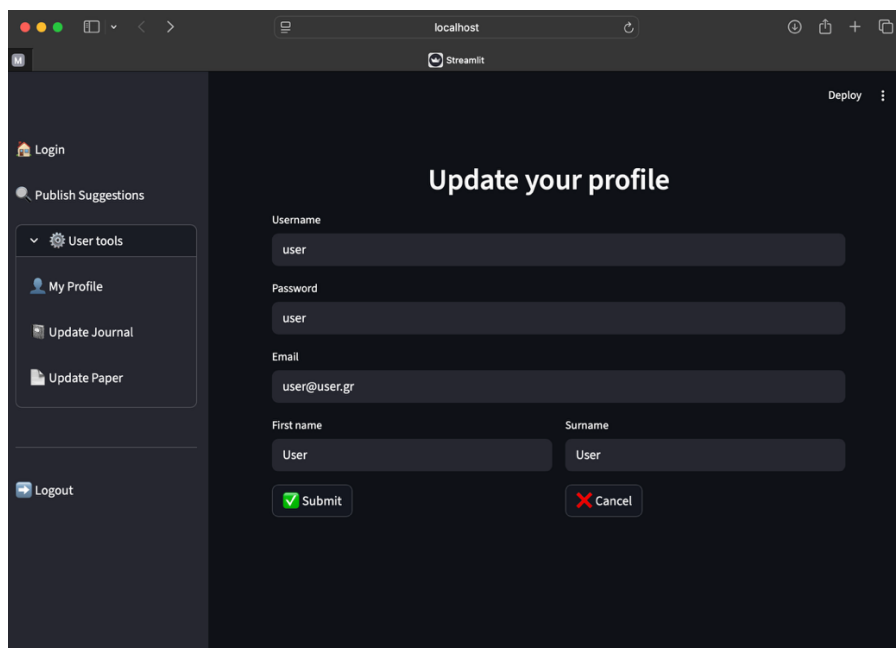
Σχήμα 5-12 Οθόνη επικαιροποίησης περιοδικού από τη βάση – Αλλαγή ονόματος



Σχήμα 5-13 Οθόνη επικαιροποίησης περιοδικού από τη βάση – Επιβεβαίωση

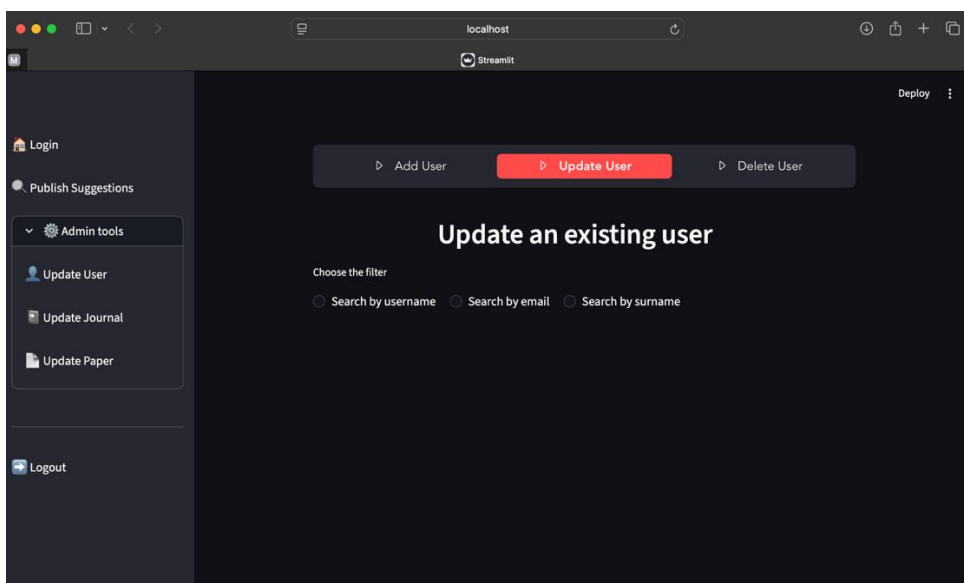
5.3.2.7 Οθόνη Επικαιροποίησης στοιχείων χρήστη (εγγεγραμμένος χρήστης και διαχειριστής)

Παρακάτω είναι η οθόνη που ο εγγεγραμμένος χρήστης μπορεί να επικαιροποιήσει τα στοιχεία του (προσβάσιμη μέσω του κουμπιού “My Profile”)



Σχήμα 5-14 Οθόνη επικαιροποίησης στοιχείων χρήστη – Εγγεγραμμένος χρήστης

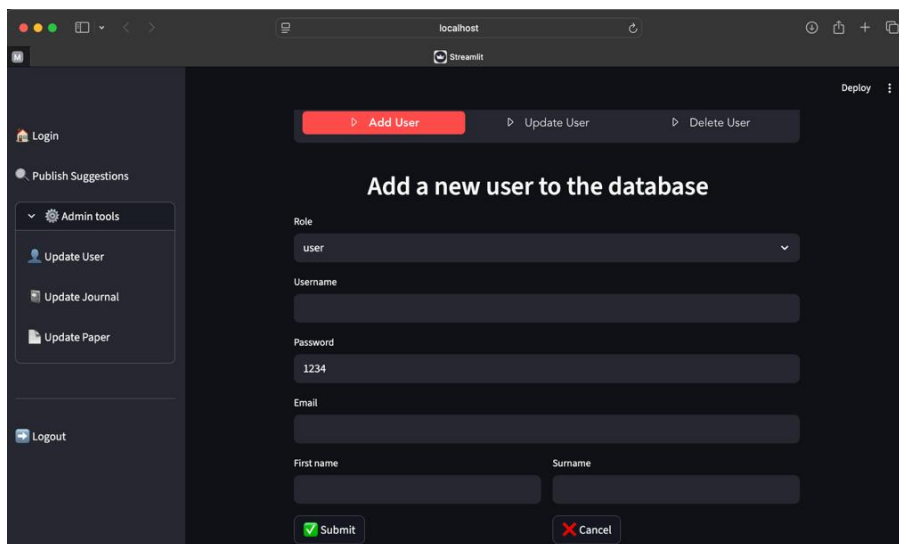
Ενώ ο διαχειριστής, ο οποίος έχει τη δυνατότητα να επικαιροποιήσει τα στοιχεία οποιαδήποτε χρήστη, μεταβαίνει στην επικαιροποίησης στοιχείων κάποιου χρήστη, αφού πρώτα επιλέξει το φίλτρο αναζήτησης κάποιου χρήστη (username, email, επώνυμο).



Σχήμα 5-15 Οθόνη επικαιροποίησης στοιχείων χρήστη – Διαχειριστής

5.3.2.8 Οθόνη Προσθήκης χρήστη (διαχειριστής)

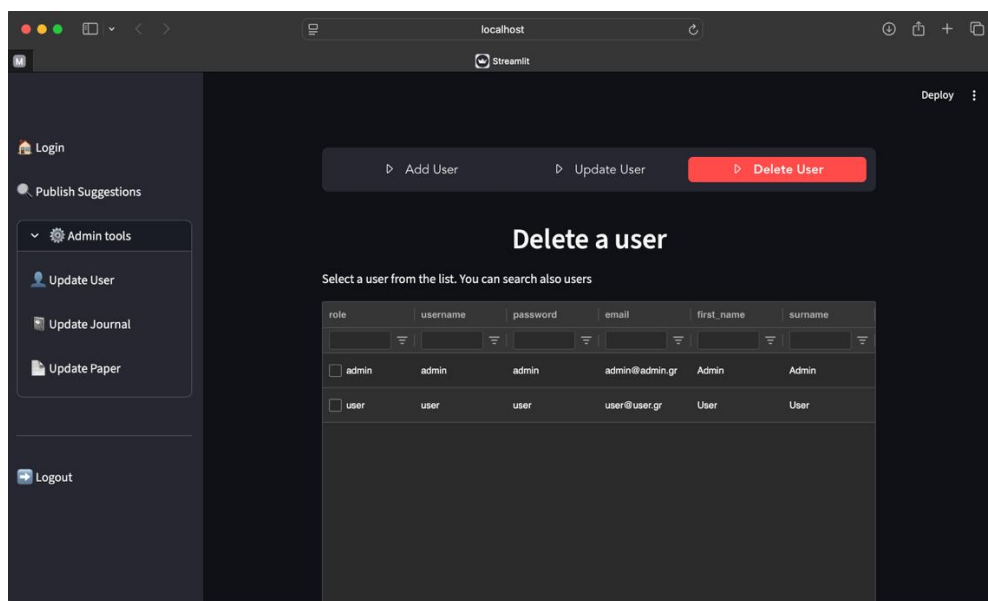
Σε αυτή την οθόνη, ο διαχειριστής μπορεί να εγγράψει έναν νέο χρήστη και να του δώσει επίσης και δικαιώματα.



Σχήμα 5-16 Οθόνη δημιουργίας νέου χρήστη

5.3.2.9 Οθόνη Διαγραφής χρήστη (διαχειριστής)

Σε αυτή την οθόνη ο διαχειριστής μπορεί να διαγράψει ένα χρήστη αφού τον εντοπίσει με διάφορα φίλτρα.

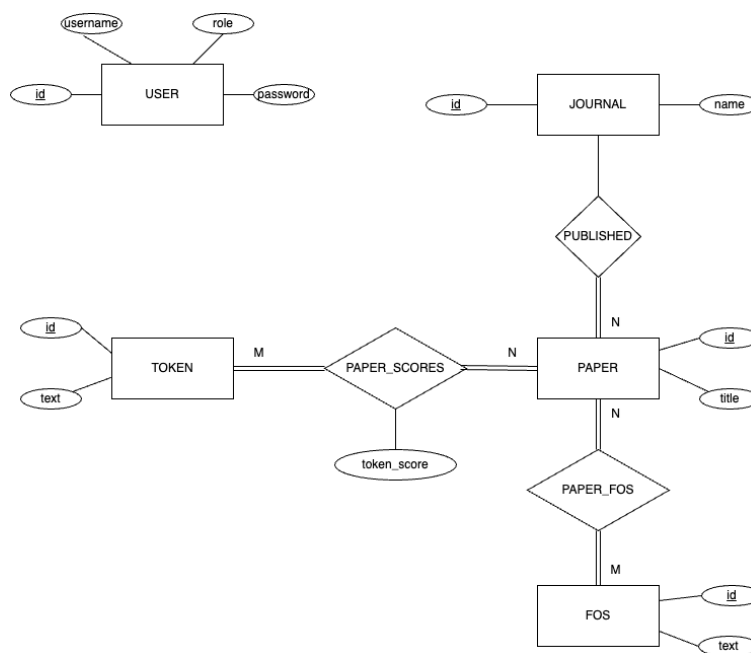


Σχήμα 5-17 Οθόνη διαγραφής χρήστη

5.3.3 Σχήμα Βάσης Δεδομένων – Περιγραφή σχέσεων οντοτήτων

Χρησιμοποιήθηκε σχεσιακή ΒΔ που να περιέχει αποθηκευμένα τα περιοδικά και τα άρθρα του κάθε περιοδικού. Το κάθε άρθρο είναι αποθηκευμένο ως λέξεις-φράσεις σε μορφή αντεστραμένου δείκτη. Συνεπώς διακρίνονται αρχικά τρεις οντότητες. Η οντότητα περιοδικό (JOURNAL), η οντότητα άρθρο (PAPER) και η οντότητα λέξη-φράση (TOKEN). Η σχέση μεταξύ JOURNAL και PAPER είναι της μορφής 1xN, μοντελοποιώντας τη σχέση «ένα άρθρο είναι δημοσιευμένο σε ένα περιοδικό, ενώ ένα περιοδικό έχει δημοσιευμένα τουλάχιστον ένα άρθρα». Η σχέση μεταξύ PAPER και TOKEN (και κατ' επέκταση η σχέση μεταξύ JOURNAL και TOKEN) είναι της μορφής NxN μοντελοποιώντας τη σχέση « ένα άρθρο (περιοδικό) περιέχει περισσότερες από μία λέξεις-φράσεις ενώ μια λέξη-φράση μπορεί να υπάρχει σε περισσότερα του ενός άρθρα (περιοδικά)». Καθώς υπάρχει έστω και μια συσχέτιση NxN, τότε είναι αναγκαίο να δημιουργηθεί και μια ξεχωριστή συνδυαστική οντότητα (PAPERxTOKEN) που να μετατρέπει την σχέση της μορφής NxN σε δύο σχέσεις της μορφής 1xN. Η νέα συνδυαστική οντότητα μοντελοποιεί τη συσχέτιση «έναν μοναδικός συνδυασμός «άρθρο x λέξη-φράση» σε μία συλλογή άρθρων τα οποία περιέχουν περισσότερες από μια λέξεις-φράσεις, οι οποίες μπορεί να είναι κοινές μεταξύ των άρθρων». Στην υλοποίηση της ΒΔ της συγκεκριμένης εφαρμογής η οντότητα PAPER x TOKEN ονομάστηκε PAPER_SCORES αντιστοιχίζοντας για κάθε άρθρο συγκεκριμένη βαθμολογία για τις λέξεις-φράσεις του. Η οντότητα PAPER συσχετίζεται επίσης και με την οντότητα Επιστημονικό Πεδίο (FOS) με σχέση NxN, μοντελοποιώντας τη σχέση «ένα άρθρο μπορεί να θεραπεύει περισσότερα του ενός

επιστημονικά πεδία, ενώ ένα επιστημονικό πεδίο μπορεί να ανήκει σε περισσότερα του ενός άρθρα». Καθώς και σε αυτή την περίπτωση πρόκειται για συσχέτιση NxN δημιουργήθηκε η συνδυαστική οντότητα PAPER_FOS κατά αναλογία της PAPER_SCORES που αναφέρθηκε παραπάνω. Επίσης, μια άλλη οντότητα που υπάρχει στη δομή της ΒΔ είναι η οντότητα χρήστη (USER) που περιέχει αποθηκευμένα στοιχεία για κάθε χρήστη. Η USER δεν συσχετίζεται με καμία από τις παραπάνω οντότητες που αναφέρθηκαν. Το Διάγραμμα Οντοτήτων Συσχετίσεων φαίνεται στο Σχήμα 5-18.



Σχήμα 5-18 Διάγραμμα Οντοτήτων Συσχετίσεων της ΒΔ

Επίσης, από το παραπάνω διάγραμμα προκύπτει το παρακάτω σχήμα της ΒΔ

USER:

<u>id</u>	username	password	role
-----------	----------	----------	------

JOURNAL:

<u>id</u>	name
-----------	------

PAPER:

<u>id</u>	title	JOURNAL.id(*)
-----------	-------	---------------

TOKEN

<u>id</u>	text
-----------	------

FOS

<u>id</u>	text
-----------	------

PAPER_SCORES

PAPER.id(*)	TOKEN.id(*)	token	score
-------------	-------------	-------	-------

PAPER_FOS

PAPER.id(*)	FOS.id(*)
-------------	-----------

5.3.4 Διαγράμματα κλάσεων

5.3.4.1 Πως λειτουργεί το Streamlit

Για να γίνει πιο κατανοητή η λειτουργία της κάθε κλάσης της εφαρμογής, θα πρέπει πρώτα να κατανοηθεί πως λειτουργεί το Streamlit, καθώς οι κλάσεις που θα αναφερθούν παρακάτω δημιουργήθηκαν ώστε να εξυπηρετούν τις μη λειτουργικές απαιτήσεις της εφαρμογής που σχετίζονται με το Streamlit.

Εγκατάσταση – εκτέλεση αρχείου πηγαίου κώδικα

Κατ' αρχάς το streamlit εγκαθίσταται στο virtual environment της python μέσω της εντολής κονσόλας:

```
pip install streamlit
```

Μετά την εγκατάσταση του, το streamlit «εισέρχεται» στο αρχείο .py που περιέχει τον πηγαίο κώδικά της εφαρμογής ως εξής:

```
import streamlit ή import streamlit as st (για συντόμηση)
```

Ακολούθως ο πηγαίος κώδικας (αρχείο .py) έχει πρόσβαση σε όλες τις μεθόδους του streamlit και μπορεί να τις καλεί ως:

```
streamlit.όνομα_εντολής(παράμετροι) ή st.όνομα_εντολής(παράμετροι).
```

Για να τρέξει το αρχείο που περιέχει τον πηγαίο κώδικα του streamlit, απαιτείται εντός του virtual environment της python η εκτέλεση της εντολής κονσόλας:

```
streamlit run όνομα_αρχείου_πηγαίου_κώδικα.py
```

Τα στοιχεία widget του streamlit και το streamlit.session_state

Μια διαδικτυακή εφαρμογή που εκτελεί το streamlit, περιέχει στατικά στοιχεία (π.χ. κείμενο φωτογραφίες κλπ) και δυναμικά στοιχεία, με τα οποία ο χρήστης μπορεί να αλληλοεπιδρά. Τα δυναμικά στοιχεία ονομάζονται widgets και περιλαμβάνουν στοιχεία όπως να πατήσει ο χρήστης ένα κουμπί, να γράψει κάποιο κείμενο, να επιλέξει από κάποιο μενού, να ανεβάσει ένα αρχείο για να διαβαστεί από το σύστημα κλπ. Όταν ξεκινά μια εφαρμογή streamlit ή κάθε φορά που ο χρήστης αλληλοεπιδρά με κάποιο widget (π.χ. πάτημα ενός κουμπιού, εισαγωγή κειμένου, ανέβασμα αρχείου κλπ) δημιουργείται μια συνεδρία (session) που προκαλεί την

εκτέλεση του πηγαίου κώδικα από την αρχή. Μια εφαρμογή streamlit θυμίζει ένα βρόγχο, οποίος δημιουργεί ένα νέο session το οποίο επανακτελεί τον πηγαίο κώδικα από την αρχή κάθε φορά που ο χρήστης αλληλοεπιδράσει με ένα widget. Συνεπώς όλες οι μεταβλητές που περιέχει ο πηγαίος κώδικας είναι τοπικές μεταβλητές που κρατάνε την τρέχουσα τιμή τους μόνο κατά τη διάρκεια ενός δεδομένου session. Όταν ένα widget αλλάζει κατάσταση, δημιουργείται ένα νέο session και όλες οι μεταβλητές του πηγαίου κώδικα γυρίζουν στην αρχική προεπιλεγμένη τιμή τους. Για παράδειγμα ο χρήστης γράφει κείμενο σε ένα widget της μορφής `input_text`, το οποίο θέλει να αποθηκεύσει σε μια μεταβλητή. Εάν πατήσει ένα widget της μορφής `button` (κουμπί) ώστε να κάνει κάποια ενέργεια με αυτή τη μεταβλητή, τότε με το πάτημα του κουμπιού, θα δημιουργηθεί ένα νέο session, ο πηγαίος κώδικας θα τρέξει από την αρχή, ο χρήστης θα δει ένα κενό `input_text` και κατά συνέπεια η τιμή της μεταβλητής θα γίνει `None` ή ίση με μια προεπιλεγμένη τιμή. Παρόλα αυτά το streamlit δίνει τη δυνατότητα στον προγραμματιστή να αποθηκεύσει τις τιμές κάποιων μεταβλητών σε ένα λεξικό (`streamlit.session_state`), με αποτέλεσμα οι μεταβλητές να κρατάνε τις τρέχουσες τιμές τους, μεταξύ των sessions. Έτσι εάν στο προηγούμενο παράδειγμα ο χρήστης είχε αποθηκεύσει στο λεξικό `session_state` την τιμή που επιστρέφει το widget `input_text` ως:

```
streamlit.session_state['my_input'] = streamlit.text_input(**kargs)
```

τότε θα μπορούσε να καλέσει τη μεταβλητή `session_state['my_input']` και να κάνει κάτι με αυτή ανεξάρτητα της δημιουργίας νέου session. Βέβαια ο σωστός τρόπος χρήσης των `session_state` εντολών για το παραπάνω παράδειγμα είναι ο παρακάτω (υπό την προϋπόθεση ότι υπάρχουν αποθηκευμένες οι μεταβλητές `'text_input_key'` και `'my_input'` στο λεξικό `streamlit.session_state`):

```
streamlit.session_state['text_input_key'] = streamlit.session_state['my_input']  
streamlit.session_state['my_input'] = streamlit.text_input(key='text_input_key', label='Give  
some text')
```

καθώς με την εντολή της πρώτης γραμμής η τιμή του `text_input` widget γίνεται ίση με αυτή που είχε πληκτρολογήσει ο χρήστης πριν την λήξη του προηγούμενου session, και ακολούθως η τιμή της μεταβλητής `st.session_state['my_input']` γίνεται ίση με εκείνη που πληκτρολογεί ο χρήστης στο τρέχον session. Εάν δεν υπήρχε η πρώτη γραμμή κώδικα, τότε με την επανεκκίνηση εκτέλεσης του πηγαίου κώδικα η τιμή `st.session_state['my_input']`, θα έπαιρνε την τιμή `None`, καθώς το `text input` widget δεν θα είχε αρχικοποιηθεί σε κάποια αποθηκευμένη τιμή και θα εμφάνιζε ένα κενό κείμενο.

Οι μεταβλητές `streamlit.session_state` μπορούν να αποθηκεύσουν οποιαδήποτε τύπο μεταβλητής (π.χ όλοι οι αριθμητικοί τύποι, λογικές μεταβλητές, λίστες, λεξικά, αντικείμενα κλπ). Επίσης το περιεχόμενο του λεξικού `streamlit.session_state` διαγράφεται όταν ο χρήστης κλείσει το παράθυρο της εφαρμογής, ή πατήσει refresh (F5) στον φυλλομετρητή.

Πρακτικά προβλήματα - Προκλήσεις

Από τα παραπάνω γίνεται αντιληπτό ότι ο προγραμματιστής μιας εφαρμογής `streamlit`, έρχεται αντιμέτωπος με τις παρακάτω προκλήσεις:

- Όλες οι μεταβλητές έχουν τοπικό χαρακτήρα ανά `session`, εκτός και εάν έχουν αποθηκευτεί ως κλειδιά στο λεξικό `session_state`
- Η δήλωση μιας `session_state` μεταβλητής είναι πολύπλοκη, καθώς η επιλογή του ονόματος του κλειδιού που θα αντιστοιχεί σε μια δεδομένη μεταβλητή θα πρέπει να είναι μοναδική.
- Κάθε widget εκτός των κουμπιών θα πρέπει να είναι συνδεδεμένο με μια μεταβλητή `session_state`. Συνεπώς για ένα π.χ. `input_text` widget απαιτούνται δύο διαφορετικές μεταβλητές, μια για την τρέχουσα τιμή του και μια άλλη ώστε να αποθηκευτεί η τιμή του σε αυτή.

Λόγω των παραπάνω χαρακτηριστικών η συγγραφή του κώδικα είναι δύσκολη, επιρρεπής σε λάθη και δύσκολη στην αποσφαλμάτωση. Ο προγραμματιστής κάθε φορά που θα αναφέρεται σε μια μεταβλητή θα πρέπει να την καλεί ως τιμή του `session_state` λεξικού. Επίσης, θα πρέπει να επιλέγει κλειδιά που δεν έχει επιλέξει νωρίτερα (σε διαφορετική τιμή το νέο κλειδί θα εγγράψει την τιμή του στο παλαιότερο). Σε πολύπλοκες εφαρμογές με πολλές οθόνες ο περιγραφικός χαρακτήρας του ονόματος ενός κλειδιού μπορεί να γίνει ιδιαίτερα εκτενής προκειμένου να είναι μοναδικός (π.χ. `“view1_subview_2_sb_journals_first_input”`, `“view1_subview_2_sb_journals_first_input_value”`, για ένα ζεύγος keys που θα χρειαστούν για ένα widget (κείμενο – αποθήκευσης τιμής του – αρχικοποίησης τιμής του).

Επίσης, ένα άλλο επίσης πολύ σημαντικό θέμα για τον προγραμματιστή είναι ότι δεν μπορεί να εφαρμόσει άμεσα τους κανόνες του αντικειμενοστραφούς προγραμματισμού ώστε να μοντελοποιήσει τις διαδικασίες του, καθώς κάθε φορά που θα ξεκινά ένα `session`, τα αντικείμενα των κλάσεων του θα δημιουργούνται από την αρχή.

5.3.4.2 Αντικειμενοστραφής προσέγγιση της παρούσας εφαρμογής

Προκειμένου να εξαλειφθεί το πρόβλημα των πολλών και διαφορετικών κλειδιών αλλά και η εφαρμογή να έχει αντικειμενοστραφή σχεδίαση, δημιουργήθηκαν οι παρακάτω κλάσεις:

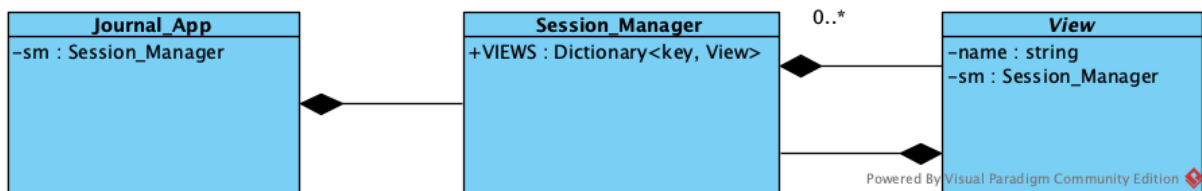
- Journal_App
- Session_Manager
- View (αφηρημένη κλάση)

Η Journal_App είναι η κλάση που θα περιέχεται στο αρχείο του πηγαίου κώδικα. Στον κατασκευαστή της περιέχει δύο ιδιότητες ένα αντικείμενο Session_Manager και την κεντρική μπάρα πλοήγησης στην κάθε οθόνη της εφαρμογής. Κάθε φορά που θα ξεκινά ένα νέο session, ο πηγαίος κώδικας θα δημιουργεί ένα νέο αντικείμενο αυτής της κλάσης δηλαδή ένα νέο αντικείμενο Session_Manager και μια νέα μπάρα πλοήγησης. Ο κώδικας της Journal_App παρουσιάζεται στο Παράρτημα 10

Η κλάση Session_Manager, αποτελεί το κεντρικό σημείο αποθήκευσης των session_state μεταβλητών οι οποίες θα μπορούν και να διαμοιράζονται μεταξύ των οθονών (για παράδειγμα το username ενός χρήστη ή ο ρόλος του ώστε να του είναι διαθέσιμη μια οθόνη ή όχι κλπ). Η συγκεκριμένη κλάση περιέχει επίσης αποθηκευμένες τις διάφορες οθόνες ως αντικείμενα σε session_state μεταβλητές. Σε κάθε νέο session, καλείται ο κατασκευαστής του Session_Manager και δημιουργείται ένα νέο αντικείμενο του. Ο κατασκευαστής του Session_Manager ελέγχει πρώτα εάν μια μεταβλητή session_state υπάρχει στο λεξικό session_state. Στην περίπτωση που η μεταβλητή session_state δεν υπάρχει (η εφαρμογή τρέχει για πρώτη φορά) τότε την δημιουργεί και αποθηκεύει την τιμή της σε μια ιδιότητα του Session_Manager, σε διαφορετική περίπτωση (νέο session) η τιμή της συγκεκριμένης ιδιότητας του Session_Manager γίνεται ίση με την τιμή της μεταβλητής session_state. Συνεπώς καλώντας την ιδιότητα ενός αντικείμενου Session_Manager ισοδυναμεί με το να καλεί κανείς την αντίστοιχη session_state μεταβλητή. Για ορισμένες ιδιότητες του ο Session_Manager έχει και τους ανάλογους setters ώστε να αλλάζουν την τιμή της αντίστοιχης session_state μεταβλητής. Με αντίστοιχο τρόπο είναι αποθηκευμένες και όλες οι οθόνες, ως αντικείμενα View, στον Session_Manager. Ο κώδικας της Session_Manager παρουσιάζεται στο Παράρτημα 11

Η View είναι μια αφηρημένη κλάση από την οποία μέσω κληρονομικότητας δημιουργούνται οι πραγματικές οθόνες που αλληλοεπιδρά ο χρήστης. Η View περιέχει ως δική της ιδιότητα το αντικείμενο Session_Manager, ώστε τα αντικείμενα της να έχουν πρόσβαση στις ιδιότητες (δηλαδή αναφορά στις αντίστοιχες session_state μεταβλητές) και τις μεθόδους

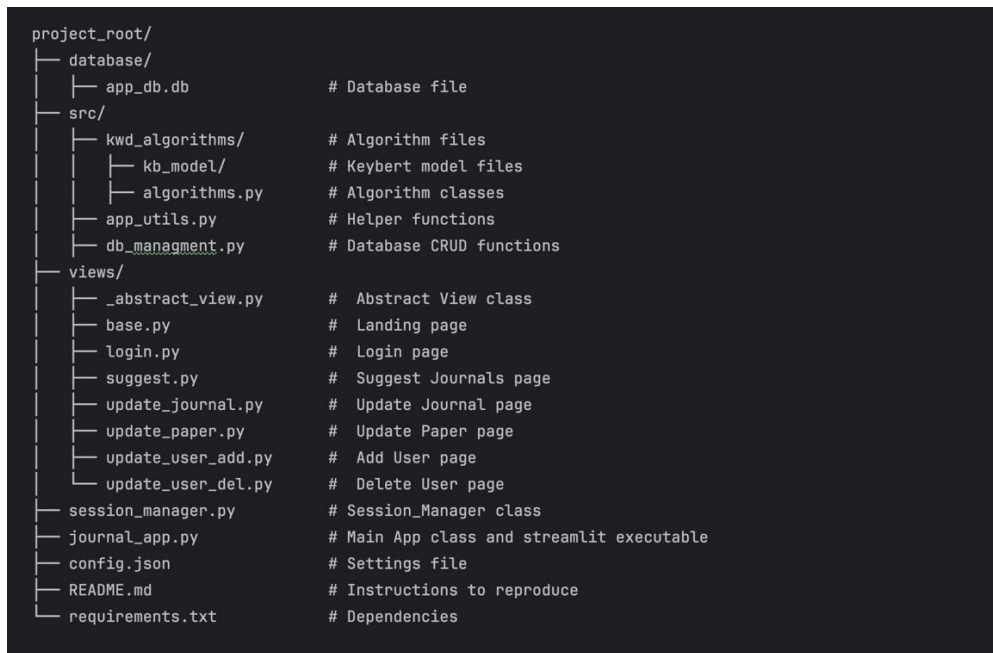
του. Επίσης, η View περιέχει ιδιότητες και μεθόδους προκειμένου να δημιουργεί μοναδικά κλειδιά για τα δικά της widgets, να διαγράφει τα μοναδικά κλειδιά κλπ. Τέλος, η κλάση View περιέχει την αφηρημένη μέθοδο `self.render()`, η οποία υπερφορτώνεται από τις θυγατρικές κλάσεις της ώστε να υλοποιούν τη λογική της αντίστοιχης οθόνης που βλέπει ο χρήστης. Ο κώδικας της View παρουσιάζεται στο Παράρτημα 12, ενώ οι συσχετίσεις μεταξύ των κλάσεων της εφαρμογής φαίνεται στο βασικό διάγραμμα κλάσεων που ακολουθεί (Σχήμα 5-19):



Σχήμα 5-19 Βασικό Διάγραμμα κλάσεων Journal_App, Session_Manager και View

5.3.4.3 Δομή και οργάνωση της εφαρμογής

Η εφαρμογή έχει αναπτυχθεί με διαχωρισμό επιπέδων, ώστε να διασφαλίζεται η επεκτασιμότητα αλλά και η συντήρηση του κώδικα. Η δομή των φακέλων επιτρέπει τον σαφή διαχωρισμό της διεπαφής χρήστη, της λογικής επεξεργασίας και της διαχείρισης δεδομένων. Η εφαρμογή υλοποιεί την παρακάτω δομή φακέλων (Σχήμα 5-20)



Σχήμα 5-20 Δομή φακέλων της εφαρμογής

Επίσης, το περιεχόμενο του αρχείου με τις παραμέτρους της εφαρμογής (config.json), παρουσιάζεται παρακάτω, με επεξήγηση για κάθε παράμετρο:

```
{
  "SUPPORTED_ALGORITHMS": ["tf-idf", "tf-isf", "rake", "yake", "textrank_directed",
    "textrank_undirected", "keybert"],
  "ALGORITHM": "tf-isf", # The selected algorithm
  "MAX_FETCHED_JOURNALS": 100, # The maximum number of suggested journals to the user
  "JOURNALS_PER_PAGE": 10, # The number of suggested journals per pagination page
  "STEMMER": true, # Stemming or not stemming
  "TOP_N": 1, # a value between 0 and 1; percentage of the tokens of a paper added to the db
  "N_GRAMS": 3, # up to ngrams
  "ROLES": ["admin", "user"],
  "FUZZY_THRESHOLD": 65, # fuzzy match threshold; used for possible journal name matching
  "JOURNAL_ABBREVIATIONS": {
    "j": "journal",
    "&": "and",
    "jrn": "journal",
    "rev": "review",
    "proc": "proceedings",
    "trans": "transactions",
    "ann": "annals",
    "bull": "bulletin",
    "res": "research",
    "surv": "survey",
    "mag": "magazine",
    "appl": "applied",
    "theor": "theoretical",
    "theoret": "theoretical",
    "exp": "experimental",
    "clin": "clinical",
    "adv": "advances",
    "dev": "development",
    "anal": "analysis"
  } # common journal abbreviations; used for journal name matching
  "STOP_WORDS": [..365 elements], # The stopwords used by the algorithms
}
```

5.4 Υλοποίηση Οθονών (Views)

Όπως αναφέρθηκε παραπάνω η κάθε οθόνη κληρονομεί την View και υπερσκελίζει (overrides) κάποιες από τις αφηρημένες μεθόδους της. Ωστόσο, πριν γίνει περιγραφή της λειτουργικότητας των κλάσεων των οθονών θα προηγηθεί περιγραφή της αφηρημένης κλάσης View προκειμένου να επεξηγηθούν οι κοινές λειτουργίες αλλά και οι λειτουργίες των αντικειμένων των θυγατρικών κλάσεων μετά την υπερσκελίση των αφηρημένων ή και άλλων μεθόδων της.

5.4.1 Η αφηρημένη κλάση View

Ιδιότητες:

- `self.name`: το όνομα της οθόνης (string)
- `self.sm`: το (τρέχον ανά session) αντικείμενο `Session_Manager`
- `self._id_counter`: Ένας θετικός ακέραιος αριθμός. Χρησιμοποιείται για την δημιουργία keys των widgets
- `self._keys`: Μια λίστα αλφαριθμητικών με τα τρέχοντα keys των widgets της οθόνης

Κατασκευαστής:

- `def __init__(self, name, session_manager)`

Παράμετροι: `name`: το όνομα της οθόνης

`session_manager`: αντικείμενο `Session_Manager`

Η μέθοδος δημιουργεί ένα αντικείμενο `View` αποθηκεύοντας στις ιδιότητες `self.name` και `self.sm` το όνομα της οθόνης και ένα αντικείμενο `Session_Manager` αντίστοιχα.

Αφηρημένες μέθοδοι

- `def _constructor(self)`

Η μέθοδος θα καλείται στον κατασκευαστή αντικειμένων των θυγατρικών κλάσεων (οθονών) και θα αρχικοποιεί τις επιπλέον ιδιότητες τους (π.χ. μεταβλητές, σημαίες κατάστασης, χαρακτηριστικά widgets κλπ).

- `def render(self)`

Η μέθοδος θα υλοποιεί την λειτουργία και την επιχειρησιακή λογική της κάθε οθόνης δημιουργώντας και αποτυπώνοντας στην οθόνη τα στατικά και δυναμικά στοιχεία (widgets) ανάλογα με το τρέχον session.

Μη αφηρημένες μέθοδοι

- `def _make_wdg_key(self)`

Η μέθοδος δημιουργεί ένα νέο μοναδικό key (για να χρησιμοποιηθεί ως `session_state` μεταβλητή στην εφαρμογή), ως `όνομα_οθόνης_αυξων_αριθμός`, χρησιμοποιώντας την τρέχουσα τιμή της μεταβλητής `self._id_counter`. Επίσης, η τιμή του key εισέρχεται στη λίστα `self._keys`, προκειμένου ο προγραμματιστής να έχει διαθέσιμα και αποθηκευμένα τα keys που έχουν δημιουργηθεί.

- `def _sync_widget (self, widget_specs, var)`

Η μέθοδος εξετάζει εάν υπάρχει μεταβλητή `session_state[widget_specs['key']]` και σε περίπτωση που δεν υπάρχει την αρχικοποιεί με το όρισμα `var`. Σε περίπτωση που υπάρχει την αφήνει να δείχνει στην τρέχουσα τιμή της. Με τη συγκεκριμένη μέθοδο, κάθε φορά η μεταβλητή `var` είναι συνδεδεμένη με το εν λόγω `widget`.

Η χρήση της μεθόδου στην `self.render()` είναι η εξής:

```
self._sync_widget( self. widget_text_input_specs, self.user_input)
self.user_input = self.streamlit.text_input (** widget_text_input_specs)
```

Με την παραπάνω χρήση, η μεταβλητή `self.user_input` θα είναι πάντοτε εκείνη που θα έχει το `widget text_input`, με χαρακτηριστικά `widget_text_input_specs`

Εάν η μέθοδος `self._sync_widget` κληθεί μετά τη δημιουργία του αντίστοιχου `widget`, το `Streamlit` προκαλεί εξαίρεση.

- `def _force_set_widget(self, widget_specs, var)`

Η συγκεκριμένη μέθοδος, έχει ανάλογη λειτουργία με εκείνη της `self._sync_widget(widget_specs, var)` με τη μόνη διαφορά ότι θέτει την τιμή ενός `widget` σε μια τιμή και είναι ανεξάρτητη των `sessions`. Για παράδειγμα ο παρακάτω κώδικας:

```
self._force_set_widget( self. widget_text_input_specs, 'Hello')
self.user_input = self.streamlit.text_input (** widget_text_input_specs)
```

Θα δείχνει συνέχεια `Hello` ανεξάρτητα την τιμή που θα έχει πληκτρολογήσει ο χρήστης στο προηγούμενο `session`. Η εν λόγω μέθοδος θα πρέπει να χρησιμοποιείται με πολύ προσοχή και μόνο σε ειδικές περιπτώσεις (π.χ όταν θα πρέπει να γίνει `reset` μιας φόρμας στις αρχικές τιμές της). Η μέθοδος προκαλεί επίσης εξαίρεση όταν κληθεί μετά τη δημιουργία του `widget`.

- `def _remove_key(self, key)`

Η συγκεκριμένη μέθοδος διαγράφει ένα κλειδί από το `streamlit.session_state` λεξικό και από την λίστα `self._keys`. Πρόκειται για πολύ εξειδικευμένη μέθοδο και καλείται όταν θέλουμε να προκαλέσουμε `reset` σε ορισμένα `widgets` και όχι σε όλα.

- `def _clear_keys(self)`

Η μέθοδος διαγράφει όλα τα υπάρχοντα `keys` του αντικειμένου από το `streamlit.session_state` λεξικό και επίσης, διαγράφει τα περιεχόμενα της λίστας `self._keys`. Για την αποφυγή τυχόν διπλοεγγραφών η μέθοδος δεν μηδενίζει την τιμή

του self._counter. Συνεπώς, κάθε φορά που δημιουργείται ένα νέο key με την self._make_wdg_key() αυτό θα είναι πάντοτε μοναδικό.

- `def _handle_db_exception(self, e)`

Η μέθοδος ενημερώνει τον χρήστη με μηνύματα στην οθόνη, όποτε προκαλείται κάποια εξαίρεση από τις λειτουργίες της βάσης δεδομένων. Η παράμετρος `e` είναι η προκληθείσα εξαίρεση.

- `def reset(self)`

Η μέθοδος καλεί διαδοχικά την `self._clear_keys()` και την `self._constructor()`, επαναφέροντας πρακτικά την οθόνη στις αρχικές μεταβλητές. Αφού διαγραφούν όλα τα κλειδιά των widgets της οθόνης, θα δημιουργηθούν widgets με τα ίδια χαρακτηριστικά αλλά με διαφορετικό κλειδί. Ανάλογα με το πως επιθυμεί ο προγραμματιστής να γίνεται η επαναφορά, η μέθοδος ενδέχεται να υπερσκελίζεται σε αντικείμενα θυγατρικών κλάσεων.

- `def refresh_session_manager(self, new_sm)`

Η μέθοδος δέχεται ως όρισμα ένα αντικείμενο `Session_Manager` και αναθέτει στην ιδιότητα `self.sm` μια αναφορά σε αυτό το αντικείμενο. Αν και πρόκειται για μια απλή ανάθεση (`self.sm = new_sm`), η συγκεκριμένη ενέργεια γίνεται με τη μορφή μεθόδου, ώστε να υπάρχει η δυνατότητα υπερσκέλισης της σε θυγατρικές κλάσεις και ιδιαίτερα σε εκείνες που περιέχουν ως ιδιότητες τους αντικείμενα `View`.

5.4.2 Αλληλοεπίδραση αντικειμένων View με την εφαρμογή

Συνοψίζοντας την λειτουργικότητα της εφαρμογής, όταν η εφαρμογή τρέξει για πρώτη φορά ή όποτε υπάρξει ένα νέο session, εκτελούνται οι παρακάτω γραμμές κώδικα:

```
# 1. Create an app object
app = Journal_App()

# 2. Refresh the views with the current sm
app.sm.apply_to_views()

# 3. Get the view from the navbar
app._render_navbar()

# 4. render that view
app.sm.CURRENT_VIEW.render()
```

Η πρώτη γραμμή δημιουργεί ένα νέο αντικείμενο της `Journal_App` και συνεπώς ένα νέο αντικείμενο `Session_Manager` στον κατασκευαστή της. Το αντικείμενο `Session_Manager` την

πρώτη φορά που θα τρέξει την εφαρμογή θα δημιουργήσει και θα αποθηκεύσει σε μια `session_state` μεταβλητή όλα τα αντικείμενα `View` (δηλαδή τις οθόνες). Η ιδιότητα του `Session_Manager` που δείχνει στην `session_state` μεταβλητή που είναι αποθηκευμένα τα αντικείμενα `View` είναι ένα λεξικό `self.VIEWS`, που έχει ως κλειδιά τα ονόματα των οθονών. Όποτε υπάρχει νέο `session`, τα αντικείμενα `View` θα παραμένουν αποθηκευμένα στην τρέχουσα κατάσταση τους όπως είχε διαμορφωθεί στο τελευταίο `session`, και θα είναι προσπελάσιμα μέσω του λεξικού `self.VIEWS` του `Session_Manager`.

Η δεύτερη γραμμή, ενημερώνει όλα τα αντικείμενα `View` ώστε να δείχνουν στο νέο αντικείμενο `Session_Manager` που δημιουργήθηκε στο τρέχον `session`, καθώς μέχρι πρότινος τα αντικείμενα `View` έδειχναν στον `Session_Manager` του προηγούμενου `session`. Η εντολή `app.sm.apply_to_all_views()`, καλείται από τον `Session_Manager` σε όλα τα αντικείμενα που δείχνει η `self.VIEWS` καλώντας για κάθε αντικείμενο `View` τη μέθοδο `refresh_session_manager(new_sm)`.

Η τρίτη γραμμή λαμβάνει από την μπάρα πλοήγησης την οθόνη που επιθυμεί ο χρήστης και ενημερώνει την ιδιότητα `self.CURRENT_VIEW` του `Session_Manager`. Η ιδιότητα `self.CURRENT_VIEW` είναι επίσης μια αναφορά σε μια `session_state` μεταβλητή, η οποία δείχνει το συγκεκριμένο αντικείμενο `View` της `self.VIEWS` που αντιστοιχεί στη τρέχουσα οθόνη. Με άλλα λόγια η λειτουργικότητα της γραμμής πλοήγησης είναι να θέτει την τιμή στη μεταβλητή `self.CURRENT_VIEW` του `Session_Manager`.

Η τέταρτη γραμμή καλεί τη μέθοδο `render()` του επιλεγμένου αντικειμένου `View` με αποτέλεσμα στο τρέχον `session` να εκτελείται ο κώδικας `render()` της επιλεγμένης οθόνης. Συνεπώς σε κάθε `session`, για το επιλεγμένο αντικείμενο `View` εκτελείται μόνο ο κώδικας της μεθόδου `render()` από την αρχή μέχρι το τέλος του `session`.

Από την παραπάνω λειτουργικότητα, προκύπτει ότι κάθε οθόνη που κληρονομεί τη `View` θα πρέπει να έχει και επιπλέον συγκεκριμένες ιδιότητες που να υλοποιούν την επιχειρησιακή λογική της, όπως για παράδειγμα, σημαίες που να καθορίζουν ποια `widgets` θα εμφανιστούν στο χρήστη, χαρακτηριστικά που να αποθηκεύουν ό,τι εισάγει ο χρήστης (κείμενο, αρχεία κλπ), μεθόδους που να επεξεργάζονται τις εισόδους του χρήστη, μεθόδους για επικοινωνία με τη ΒΔ και ανάκτηση-εγγραφή από-σε αυτή κλπ. Οι επιπλέον ιδιότητες θα πρέπει να αρχικοποιούνται στην μέθοδο `self._constructor()` η οποία θα καλείται στη `self.__init__()` της κάθε κληρονομούμενης `View`.

Μια άλλη ομάδα ιδιοτήτων που θα πρέπει να έχουν οι `View` κλάσεις, είναι τα χαρακτηριστικά των `widgets`. Στο `streamlit` κάθε `widget` δημιουργείται με `**kwargs` ορίσματα

που επιτελούν κάποια λειτουργικότητα. Τα πιο σημαντικά ορίσματα είναι τα `key` και `label`, που αντιστοιχούν στο κλειδί του widget και στην ετικέτα που βλέπει ο χρήστης στην οθόνη. Για παράδειγμα, η εντολή `streamlit.button(label= 'Press me')` δημιουργεί ένα κουμπί που αναγράφει `Press me`. Επειδή τα widgets επαναδημιουργούνται σε κάθε session, θα ήταν ανώφελο να αποθηκευτούν ως ιδιότητες των κλάσεων `View`. Αντί αυτού, αποθηκεύονται τα χαρακτηριστικά τους στην `self._constructor()` υπό μορφή λεξικού ως ιδιότητες της κλάσης και στην `self.render()` απλά δηλώνεται το αντίστοιχο streamlit widget που φέρει ως όρισμα τα χαρακτηριστικά του. Για παράδειγμα το μοναδικό κλειδί και η ετικέτα ενός κουμπιού τύπου `submit` μπορούν να δηλωθούν στον `self._constructor()` ως ένα λεξικό της μορφής:

```
self.btn_submit = {'key':self._make_wdg_key(), 'label': 'Submit'}
```

και στην `self.render()` να εξετάζεται σε κάθε session εάν πατήθηκε το συγκεκριμένο κουμπί με την παρακάτω γραμμή κώδικα:

```
if streamlit.button(**self.btn_submit):
```

Με την ίδια λογική επιτυγχάνεται και ο συγχρονισμός των text widgets με το κείμενο που εισάγει ο χρήστης. Για παράδειγμα η τιμή του κειμένου που εισάγει ο χρήστης αποθηκεύεται σε μια ιδιότητα της κλάσης πχ `self.input` και τα χαρακτηριστικά του `text_input` widget σε ένα λεξικό της μορφής:

```
self.text_input_widget = {'key':self._make_wdg_key(), 'label': 'Give some text'}
```

Κατά τη κλήση της `self.render()` οι παρακάτω γραμμές κώδικα επιτυγχάνουν (α) το κείμενο που εισάγει ο χρήστης κάθε φορά να αποθηκεύεται στην μεταβλητή `self.input` και (β) το `text_input` widget να δείχνει το κείμενο που πληκτρολόγησε ο χρήστης στο προηγούμενο session όταν ξεκινά ένα νέο session:

```
self.sync_widget(self.text_input_widget, self.input)
```

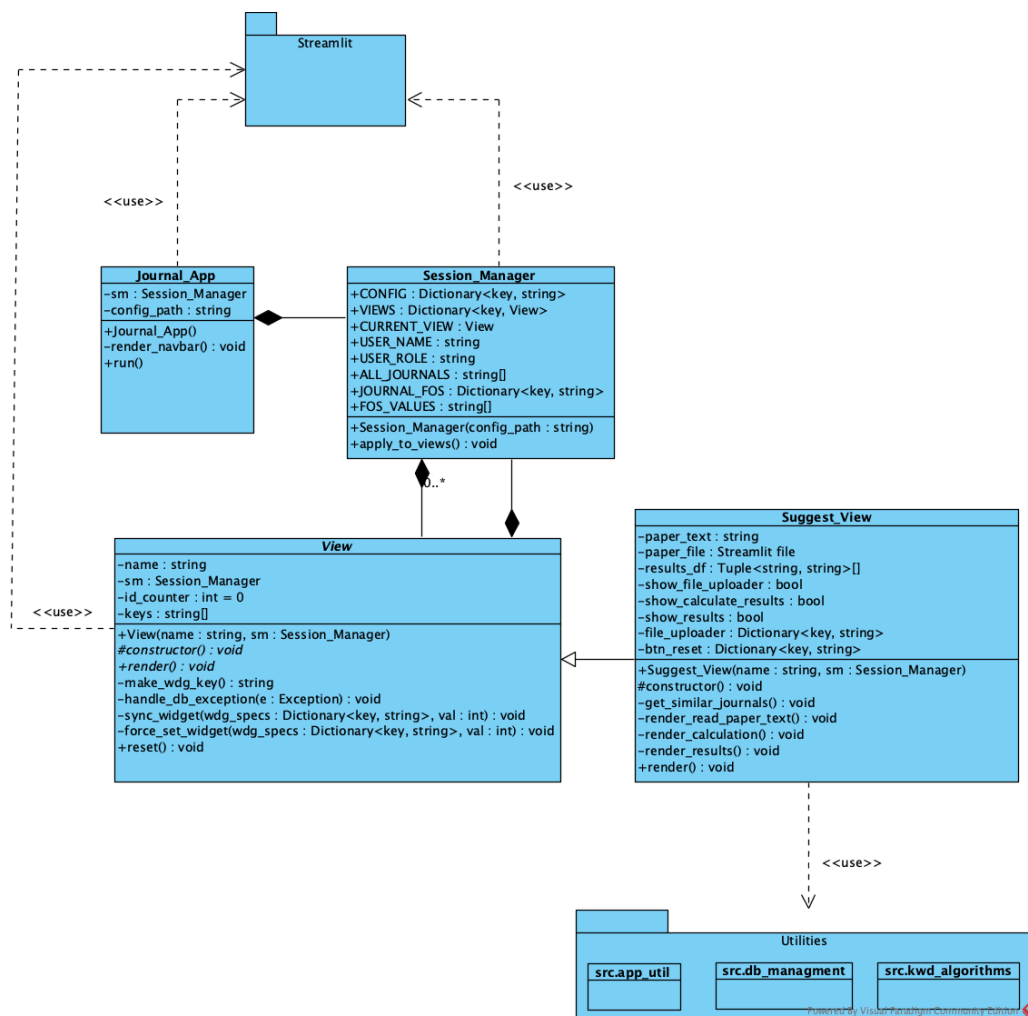
```
self.input = streamlit.text_input(** self.text_input_widget)
```

Επίσης, όπως προαναφέρθηκε τα αντικείμενα `View` έχουν επίσης πρόσβαση στις ιδιότητες και μεθόδους του `Session_Manager` και εμμέσως στις `session_state` μεταβλητές της εφαρμογής, ώστε να μπορούν να τις προσπελαίνουν ή και να τις τροποποιούν, μέσω των αντίστοιχων `setters`.

5.4.3 Η κλάση `SuggestView(View)`

Η παρούσα εφαρμογή είναι πολύπλοκη και υλοποιεί για κάθε οθόνη μια ξεχωριστή κλάση που κληρονομεί την `View`, καθώς κάθε οθόνη έχει τις δικές της ιδιαιτερότητες. Καθώς όλες οι θυγατρικές κλάσεις `View` υλοποιούν την λειτουργικότητα της αφηρημένης κλάσης `View`, με

ξεχωριστές ιδιότητες η κάθε μία, θα ήταν υπερβολή να παρατεθεί ο κώδικας για όλες αυτές. Ενδεικτικά, στο Παράρτημα 13 παρατίθεται ο κώδικας για την κλάση SuggestView, που κληρονομεί την View και υλοποιεί τον κύριο σκοπό ανάπτυξης της εφαρμογής δηλαδή την ανάκτηση συστάσεων των σχετικών περιοδικών. Επίσης, στα Παραρτήματα 14 και 15 παρουσιάζεται αποσπασματικά ο κώδικας των βοηθητικών συναρτήσεων/μεθόδων (src/util.py) και των ερωτημάτων στη ΒΔ (src/db_management.py) που χρησιμοποιήθηκαν σε αυτή την κλάση, αντίστοιχα. Τέλος, στο Σχήμα 5-21 φαίνεται ένα εκτεταμένο διάγραμμα UML στο οποίο παρουσιάζονται περισσότερο αναλυτικά τα χαρακτηριστικά και οι μέθοδοι των κλάσεων (εκτός των getters και setters).



Σχήμα 5-21 Εκτεταμένο Διάγραμμα κλάσεων Journal_App, Session_Manager, View και SuggestView

6 Συμπεράσματα – Συζήτηση

Στην παρούσα ΔΕ συγκρίθηκαν και αξιολογήθηκαν διάφορες αλγοριθμικές προσεγγίσεις ως προς την εξαγωγή και βαθμολογική κατάταξη ΛΦΚ από επιστημονικά άρθρα γραμμένα στην αγγλική γλώσσα, με σκοπό οι αλγοριθμικές προσεγγίσεις να χρησιμοποιηθούν σε εφαρμογές ΑΠ. Ειδικότερα, και μέσω της χρήσης συλλογών κατάλληλων δεδομένων (δημοσιευμένα άρθρα) οι αλγόριθμοι της παρούσας ΔΕ αξιολογήθηκαν αρχικά στο να κατατάσσουν υψηλότερα τις λέξεις κλειδιά του συγγραφέα ενός άρθρου και ακολούθως στο να προβλέπουν το περιοδικό και το επιστημονικό πεδίο του περιοδικού που ένα συγκεκριμένο άρθρο είναι δημοσιευμένο.

Σε ότι αφορά στην ικανότητα των αλγορίθμων να κατατάσσουν υψηλότερα τις ΛΦΚ των κειμένων, φάνηκε ότι η προσέγγιση της αναπαράστασης ενός εγγράφου σε γράφο των όρων του (υλοποιήσεις TEXTRANK) να υπερτερεί έναντι των υπολοίπων προσεγγίσεων που εξετάστηκαν. Συγκεκριμένα και μέσω της μετρικής της μέσης ακρίβειας φάνηκε ότι οι δύο υλοποιήσεις του TEXTRANK να κατατάσσουν τις ΛΦΚ του συγγραφέα σε περισσότερο υψηλές θέσεις σε σχέση με τους υπόλοιπους αλγόριθμους, ενώ οι αλγόριθμοι TF-IDF και TF-ISF φάνηκε να υπερτερούν σε σχέση με τους RAKE, YAKE και KEYBERT. Αυτή η τάση επιβεβαιώθηκε και μέσω της μετρικής της ανάκλασης, αν και με τη χρήση της εν λόγω μετρικής φάνηκε ότι όλοι οι αλγόριθμοι, με εξαίρεση τον YAKE, κατέταξαν στις υψηλότερες θέσεις (εντός του 10% των σημαντικότερων λέξεων-φράσεων του εγγράφου που βαθμολόγησαν) τουλάχιστον το 70% των ΛΦΚ του συγγραφέα. Επίσης, όλοι οι αλγόριθμοι κατάφεραν να κατατάξουν το σύνολο των ΛΦΚ του συγγραφέα στο 60% των σημαντικότερων λέξεων-φράσεων του εγγράφου, ανεξάρτητα του αριθμού n_grams (όταν $n \leq 3$) ή του εάν εφαρμόστηκε stemming. Επίσης, η παραλλαγή του TF-IDF (αλγόριθμος TF-ISF) φάνηκε να δίνει παρόμοια αποτελέσματα με τον TF-IDF, στην παρούσα ΔΕ. Το συγκεκριμένο εύρημα -εάν επιβεβαιωθεί και με τη χρήση και άλλων συνόλων δεδομένων- είναι πολύ σημαντικό καθώς ο αλγόριθμος TF-ISF δεν έχει τα μειονεκτήματα του TF-IDF όπως ανάγκη ύπαρξης ενός μεγάλου εύρους συλλογής εγγράφων (Manning et al, 2008) καθώς και το υπολογιστικό κόστος της παραμέτρου IDF.

Μια άλλη παράμετρος που αξιολογήθηκε στην παρούσα ΔΕ ήταν η ταχύτητα εκτέλεσης των αλγορίθμων. Εδώ φάνηκε καθαρά ότι ο αλγόριθμος KEYBERT ήταν μακράν ο πλέον αργός αλγόριθμος, γεγονός που επιβεβαιώνεται και από την βιβλιογραφία (Nadim et al, 2023). Επίσης, φάνηκε ότι το stemming καθυστέρησε σε πολύ σημαντικό βαθμό όλους τους

αλγόριθμους, με εξαίρεση τον TF-IDF. Η αύξηση στην ταχύτητα υλοποίησης όταν εφαρμόζεται stemming είναι αναμενόμενη καθώς το stemming απαιτεί επιπλέον υπολογιστικό κόστος για να εφαρμοστεί. Το γεγονός ότι η καθυστέρηση λόγω stemming δεν ήταν αισθητή στον TF-IDF μπορεί πιθανόν να αποδοθεί στο ότι το σύνολο του «λεξικού» της συλλογής των εγγράφων είναι πολύ μικρότερο όταν εφαρμοστεί stemming (και κατά συνέπεια οι απαιτήσεις σε υπολογιστικό κόστος της παραμέτρου IDF να είναι μειωμένες) σε σχέση με το όταν δεν εφαρμοστεί stemming. Παρόλα αυτά και ανεξάρτητα του εάν εφαρμόστηκε stemming ή όχι, ο αλγόριθμος TF-ISF ήταν κατά πολύ ταχύτερος του TF-IDF καθώς στις μετρήσεις μας απαιτούσε κατά μέσο όρο μόλις 27 ms (428 ms με stemming) για να κατατάξει τις ΛΦΚ ενός πλήρους άρθρου, σε αντίθεση με τον TF-ISF που απαιτούσε κατά μέσο όρο 1304 ms (1095 ms με stemming).

Όταν οι αλγόριθμοι συγκρίθηκαν μεταξύ τους ως προς το να προβλέπουν το περιοδικό που είναι δημοσιευμένο ένα άρθρο, φάνηκε μια υπεροχή του KEYBERT έναντι των υπολοίπων αλγορίθμων. Πιο συγκεκριμένα ο KEYBERT προέβλεπε το περιοδικό που ήταν δημοσιευμένο ένα άρθρο κατά μέσο όρο στην 9^η θέση επί ενός συνόλου 1092 περιοδικών, ενώ οι υπόλοιποι αλγόριθμοι προέβλεπαν το περιοδικό του άρθρου κατά μέσο όρο μεταξύ 10^{ης} και 37^{ης} θέσης, με τους αλγόριθμους TEXTRANK και RAKE να υπερτερούν σε σχέση με τους TF-IDF, TF-ISF και YAKE. Το γεγονός ότι ο KEYBERT έδωσε καλύτερα αποτελέσματα, πιθανότατα να οφείλεται στην ικανότητα του να βαθμολογεί το νόημα ενός κειμένου, μέσω της σημασιολογικής αναπαράστασης του κειμένου, σε αντίθεση με τους υπόλοιπους αλγόριθμους οι οποίοι βαθμολογούν ομοιότητα λεκτικών. Κατά συνέπεια, ο KEYBERT μπορεί να ανακαλύψει εάν υπάρχει νοηματική συνάφεια μεταξύ κειμένων τα οποία διαφέρουν ακόμα και σε μεγάλο βαθμό ως προς τα λεκτικά τους, κάτι το οποίο είναι πολύ σύνηθες σε επιστημονικά άρθρα.

Μελέτες συμπεριφοράς χρηστών, όπως αυτή των Joachims et al, (2007), δείχνουν ότι όταν οι χρήστες κάνουν μια αναζήτηση, επικεντρώνονται κυρίως στα πρώτα αποτελέσματα της αναζήτησης και σπάνια εξετάζουν αποτελέσματα πέρα από αυτά. Συνεπώς, μία κατάταξη ΑΠ θεωρείται ικανοποιητική όταν τα σχετικά αποτελέσματα εμφανίζονται εντός των πρώτων θέσεων (π.χ. πρώτες 10 θέσεις), ενώ η βέλτιστη απόδοση επιτυγχάνεται όταν αυτά εντοπίζονται στις πρώτες θέσεις της κατάταξης (μεταξύ 1^{ης} και 3^{ης} θέσης). Το γεγονός ότι όλοι οι αλγόριθμοι της παρούσας ΔΕ κατάφεραν να κατατάξουν το περιοδικό ενός συγκεκριμένου άρθρου στην 9^η ή και χαμηλότερες θέσεις, δεν σημαίνει ότι θα είναι αναποτελεσματικοί όταν

χρησιμοποιηθούν σε μια εφαρμογή ΑΠ που προτείνει περιοδικά προς δημοσίευση για τους εξής λόγους:

- (α) τα άρθρα που χρησιμοποιήθηκαν στις μετρικές είναι ήδη δημοσιευμένα σε περιοδικά κατόπιν προσωπικής επιλογής του συγγραφέα τους,
- (β) ένα άρθρο που είναι δημοσιευμένο σε ένα περιοδικό θα μπορούσε κάλλιστα να είναι δημοσιευμένο και σε ένα άλλο περιοδικό του ίδιου ή παρόμοιου επιστημονικού αντικειμένου και
- (γ) ο σκοπός της εφαρμογής ΑΠ δεν είναι να προτείνει στο χρήστη το ένα και μοναδικό περιοδικό για να δημοσιεύσει την εργασία του, αλλά μια ομάδα περιοδικών, τα οποία θα είναι συναφή με το επιστημονικό πεδίο της εργασίας.

Συνεπώς η πρακτική αξιολόγηση των αλγορίθμων θα πρέπει να εστιάζει στο κατά πόσο οι αλγόριθμοι είναι σε θέση να «διαβάσουν» ένα άρθρο, να «αντιληφθούν» τα επιστημονικά πεδία που θεραπεύει μέσα από μια συλλογή γνώσης επιστημονικών πεδίων από άλλα άρθρα που έχουν «διαβάσει» προηγουμένως και ακολούθως να αξιοποιήσουν την υπάρχουσα «γνώση» τους ώστε να προτείνουν μια ομάδα περιοδικών που θα θεραπεύουν το επιστημονικό πεδίο της προς δημοσίευση εργασίας. Στην παρούσα ΔΕ, φάνηκε ότι όλοι οι αλγόριθμοι, ανεξάρτητα εάν εφαρμόστηκε stemming ή όχι, προέβλεψαν μεταξύ 1^{ης} και 2^{ης} θέσης τουλάχιστον ένα από τα επιστημονικά πεδία του περιοδικού που είναι δημοσιευμένο ένα άρθρο το οποίο «διάβασαν», με τον KEYBERT να υπερτερεί ελαφρώς των υπόλοιπων αλγορίθμων ως προς την μετρική *MRR*.

Μια εφαρμογή ΑΠ θα πρέπει να κατατάσσει στις υψηλότερες θέσεις την ανακτημένη πληροφορία, χρησιμοποιώντας ταυτόχρονα τους λιγότερους δυνατούς υπολογιστικούς πόρους (Manning et al., 2008). Από τα αποτελέσματα της παρούσας ΔΕ, δεν φάνηκε ένα ξεκάθαρο προβάδισμα κάποιου από τους αλγόριθμους που εξετάστηκαν για χρήση σε μια εφαρμογή συστάσεων περιοδικών προς δημοσίευση. Για παράδειγμα ο KEYBERT φάνηκε να υπερτερεί έναντι των υπολοίπων αλγορίθμων ως προς την ικανότητα του να ανακτά την σωστή πληροφορία, παρόλα αυτά ήταν κατά πολύ αργότερος (τουλάχιστον 4 φορές) από τον πιο αργό από τους υπόλοιπους αλγόριθμους. Επίσης, το stemming καταναλώνει σημαντικούς υπολογιστικούς πόρους, αλλά στον αντίποδα ελαττώνει τις απαιτήσεις αποθήκευσης σε τέτοιο βαθμό που στην παρούσα ΔΕ αντιστάθμισε ακόμα και το υπολογιστικό κόστος του TF-IDF. Ένας άλλος παράγοντας που αυξάνει τις απαιτήσεις σε αποθηκευτικούς πόρους είναι τα n-grams. Στην παρούσα ΔΕ ο αριθμός των n-grams (μέχρι trigrams) δεν φάνηκε να επηρεάζει την ικανότητα των αλγορίθμων να προβλέπουν τις σημαντικές ΛΦΚ, συνεπώς η χρήση μόνο

unigrams θα μπορούσε να είναι μια λύση που συνδυάζει ταυτόχρονα χαμηλές υπολογιστικές και αποθηκευτικές απαιτήσεις. Ωστόσο, μια τέτοια πρακτική θα πρέπει να επιλέγεται με προσοχή, καθώς σύμφωνα με τους Manning et al, (2008) τα n-grams διαδραματίζουν πολύ σημαντικό ρόλο στην ανάκτηση πληροφορίας και ειδικότερα σε ερωτήματα τύπου wildcard. Μια άλλη παράμετρος που θα μπορούσε να αξιοποιήσει ορθότερα τους αποθηκευτικούς πόρους -και κατ' επέκταση να ελαττώσει το υπολογιστικό κόστος- είναι αριθμός των ΛΦΚ που αναπαριστούν κάθε έγγραφο στην μνήμη. Στις μετρικές της παρούσας ΔΕ κάθε έγγραφο αναπαραστάθηκε στην μνήμη από το σύνολο των n-gram όρων του, εκτός των κοινότυπων όρων. Ωστόσο σε μια πραγματική εφαρμογή που η συλλογή της θα αποτελείται από δεκάδες ή και εκατοντάδες χιλιάδες έγγραφα, το κάθε έγγραφο καλό θα είναι να αναπαρίσταται από ένα ποσοστό των περισσότερων αντιπροσωπευτικών όρων του και όχι καθολικά. Όπως φάνηκε από τις μετρικές της παρούσας ΔΕ ένα ποσοστό μεταξύ 10 και 60% των όρων του εγγράφου είναι ικανοποιητικό προκειμένου να αναπαραστήσει ένα επιστημονικό άρθρο σε μια εφαρμογή ΑΠ. Είναι προφανές από τα παραπάνω ότι η επιλογή του κατάλληλου αλγόριθμου είναι μια πολυπαραγοντική διαδικασία που θα πρέπει να προσαρμόζεται ανάλογα με τις απαιτήσεις της εφαρμογής. Για τον λόγο αυτό, η εφαρμογή ΑΠ που αναπτύχθηκε στην παρούσα ΔΕ, δημιουργήθηκε με τους κανόνες της αντικειμενοστραφούς προσέγγισης προκειμένου ο προγραμματιστής αλλά και σε μεγάλο βαθμό ο χρήστης (διαχειριστής) να είναι σε θέση να την προσαρμόσει ανάλογα. Η παρούσα εφαρμογή έχει σχεδιαστεί ώστε να λειτουργεί για οποιονδήποτε από τους παραπάνω αλγορίθμους, για οποιοδήποτε αριθμό n_grams και να λειτουργεί με ή χωρίς stemming. Επίσης, ο διαχειριστής μπορεί να επιλέξει εάν επιθυμεί οι αλγόριθμοι να επιστρέφουν την βαθμολογία όλων των όρων ενός κειμένου, ή κάποιο ποσοστό των πλέον επικρατέστερων από τους όρους, και κατά συνέπεια να την προσαρμόζει στις απαιτήσεις του.

Προτάσεις για περαιτέρω έρευνα.

Στην παρούσα ΔΕ φάνηκε ο αλγόριθμος TF-ISF να δίνει παρόμοια αποτελέσματα με τον TF-IDF κάτι που θα πρέπει να επιβεβαιωθεί και με τη χρήση άλλων συνόλων δεδομένων και σε κείμενα που ανήκουν σε διαφορετικά πεδία (π.χ. νομικά κείμενα, ιατρικά κείμενα, λογοτεχνία κλπ). Επίσης, η μέτρηση της ΑΠ έγινε σε περιλήψεις άρθρων. Προτείνεται για την επιβεβαίωση των αποτελεσμάτων η μέτρηση της ΑΠ να γίνει με χρήση αντίστοιχων συλλογών δεδομένων που έχουν και το πλήρες κείμενο του άρθρου. Στην παρούσα ΔΕ χρησιμοποιήθηκε ο κάθε αλγόριθμος ξεχωριστά. Μία πρόταση για μελλοντική έρευνα είναι η χρήση μιας

συνδυασμένης αλγοριθμικής τεχνικής ΑΠ που να βασίζεται στα πλεονεκτήματα κάθε αλγορίθμου. Τέλος, με τη χρήση των κατάλληλων κοινότυπων όρων, θα μπορούσε επίσης να εξεταστεί εάν η εν λόγω εφαρμογή ΑΠ μπορεί να χρησιμοποιηθεί και σε κείμενα εκτός της αγγλικής ή σε μικτά κείμενα (π.χ. διπλωματικές εργασίες, διδακτορικές διατριβές κλπ).

Βιβλιογραφία

- Anandarajan, M., Hill, C., Nolan, T. (2019). *Practical Text Analytics: Maximizing the value of text data*. Switzerland: Springer.
- Brin S. and Page., L. (1998.) The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, Vol 30, pp 1–7.
- Campos, R. , Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C., Jatowt, A. (2020). YAKE! Keyword Extraction from Single Documents using Multiple Local Features. *Information Sciences*. Vol 509, pp 257-289.
- Crestani, F., Lalmas, M., Rijsbergen, C., Campbell, I. (2001). Is This Document Relevant? ...Probably: A Survey of Probabilistic Models in Information Retrieval. *ACM Computing Surveys*. Vol 30, pp.528-552.
- Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.
- Friedman, N, Moises G. (1996). Building classifiers using Bayesian networks. In *Proc. National Conference on Artificial Intelligence*, pp. 1277–1284.
- Grootendorst, M. (2020) “KeyBERT: Minimal keyword extraction with BERT,” Zenodo, Tech. Rep., 2020. (<https://doi.org/10.5281/zenodo.4461265>)
- Gupta T.E. (2017) Keyword Extraction: a Review. *International Journal of Engineering Applied Sciences and Technology*, Vol. 2 (4), pp 215-220.
- Herbrich, R and Graepel T., (2010) *Handbook of Natural Language Processing*, 2nd edition, New York. Chapman and Hall/CRC.
- Joachims, T., Granka, L., Pan, B., Hembrooke, H., Gay, G. (2007). Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *ACM Transactions on Information Systems*, 25(2), pp 6-es
- Liddy, E.D. (2001). *Natural Language Processing. In Encyclopedia of Library and Information Science*, 2nd Ed. NY. Marcel Decker, Inc.
- Mahedi Hasan, H. M., Sanyal F., Chaki D., Ali, M. H. (2017) An empirical study of important keyword extraction techniques from documents. *1st International Conference on Intelligent Systems and Information Management (ICISIM)*, Aurangabad, India, 2017, pp. 91-94.

- Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge: MIT Press.
- Manning, C.D., Raghavan, P. Schütze, H. (2008), *Introduction to Information Retrieval*, Cambridge University Press.
- Mihalcea R. and Tarau, P. (2004). Textrank: Bringing order into text. Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, pp. 404-411.
- Nadim, M., Akopian D., Matamoros, A. (2023) A Comparative Assessment of Unsupervised Keyword Extraction Tools, *IEEE Access*, Vol 11, pp. 144778-144798.
- Pan, S., Li, Z., Dai, J. (2019). An improved TextRank keywords extraction algorithm. ACM TURC '19: Proceedings of the ACM Turing Celebration Conference - China. pp. 1-7.
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, San Mateo, CA
- Porter, M. F. (1980). An algorithm for suffix stripping. Program Electronic Library and Information Systems, Vol 14(3), pp130–137.
- Rajput, S., Gahoi, A., Reddy, M., Sharma, D.,M. (2020). N-Grams TextRank : A Novel Domain Keyword Extraction Technique. Proceedings of the 17th International Conference on Natural Language Processing: TermTraction 2020 Shared Task, pp. 9–12
- Reimers, N., Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 3982–3992.
- Robertson, S. and Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval. Vol 3(4), pp 333–38
- Rose S., Engel, D., Cramer, N., Cowley, W. (2010). Automatic Keyword Extraction from Individual Documents. *In Text Mining: Applications and Theory*. Edited by Michael W. Berry and Jacob Kogan. John Wiley and Sons Ltd
- Salton,G. and Lesk, M.A. (1968). Computer evaluation of indexing and text processing. Journal of the ACM (JACM), Vol 15.1, pp. 8–36.
- Sharma, P. and Li, Y. (2019), Self-Supervised Contextual Keyword and Keyphrase Retrieval with Self-Labeling. Preprints, 2019080073. Ανακτήθηκε Μάρτιος 2026 από <https://doi.org/10.20944/preprints201908.0073.v1>
- Siddiqi S., Sharan A., (2015). Keyword and Keyphrase Extraction Techniques: A Literature Review. International Journal of Computer Applications, Vol 109(2), pp 18-23.

- Spärck J., Walker, K.S., Robertson, S., E. (2000). A probabilistic model of information retrieval: Development and comparative experiments. *Information Processing and Management*. Vol 36(6), pp.779–808, 809–840.
- Tan, P.N., Steinbach, M., Kumar. V. (2018). Εισαγωγή Στην Εξόρυξη Δεδομένων, (2^η έκδοση). Εκδόσεις Τζιόλα
- Valcarce, D., Bellogín, A., Parapar, J., Castells, P. (2020). Assessing ranking metrics in top-N recommendation. *Information Retrieval Journal*. Vol 23, pp. 411-448.
- Vaswani, A., Shazeer, A., Parmar, N., Uszkoreit, J., Jones. L., Gomez, A.N., Kaiser, L. Polosukhin, I., (2017). Attention is all you need. *Advances in Neural Information Processing Systems*. Vol (30), pp 5998–6008.
- Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*. Vol (33), pp. 5776-5788.
- Witten, I., Paynter, G., Frank, E., Gutwin, C., Nevill-Manning, C. (1999). KEA: Practical Automatic Keyphrase Extraction. *ACM DL*. 254-255. Ανακτήθηκε Μάρτιος 2026 από <https://doi.org/10.48550/arXiv.cs/9902007>
- Zipf, G.K. (1932), *Selected studies of the principle of relative frequency in language*. Reprint 2014. Cambridge, MA, US: Harvard University Press.

ΠΑΡΑΡΤΗΜΑ: Υλοποίηση κλάσεων της παρούσας ΔΕ

1. Κοινότυποι όροι της αγγλικής γλώσσας που χρησιμοποιήθηκαν στην παρούσα μελέτη

#UNION NLTK - SKLEARN

['side', 'please', 'cry', 'they'd', 'herein', 'de', 'eight', 'ours', 'two', 'already', 't', 'again', 'hasnt', 'latter', 'empty', 'throughout', 'has', 'four', 'across', 'thick', 'fifty', 'isn', 'because', 'it's', 'didn', 'up', 'with', 'enough', 'these', 'much', 'whatever', 'the', 'aren', 'always', 'take', 'while', 'further', 'whereupon', 'he', 'either', 'whole', 've', 'd', 'don't', 'whither', 'mustn', 'fill', 'sometime', 'latterly', 'into', 'both', 'they'll', 'one', 'themselves', 'you'll', 'but', 'get', 'elsewhere', 'should', 'too', 'hence', 'now', 'yours', 'full', 'down', 'someone', 'whoever', 'whereafter', 'own', 'otherwise', 'i', 'detail', 'can', 'you'd', 'everything', 'm', 'wouldn't', 'neither', 'only', 'system', 'nobody', 'myself', 'have', 'wouldn', 'did', 'over', 'becomes', 'i'm', 'sometimes', 'inc', 'mightn't', 'bill', 'him', 'an', 'which', 'they've', 'couldnt', 'been', 'shan't', 'thereafter', 'aren't', 'it', 'don', 'during', 'see', 'interest', 'beside', 'she', 'serious', 'hereupon', 'show', 'being', 'call', 'would', 'i'd', 'therein', 'wasn', 'nothing', 'un', 'you've', 'below', 'for', 'ever', 'as', 'indeed', 'had', 'keep', 'didn't', 'made', 'hasn't', 'll', 'wherever', 'just', 'y', 'himself', 'afterwards', 'first', 'around', 'few', 'towards', 'therefore', 's', 'whence', 'whenever', 'might', 'else', 'put', 'toward', 'even', 'twelve', 'whereas', 'meanwhile', 'cannot', 'whom', 'here', 'front', 'under', 'yet', 'them', 'although', 'about', 'namely', 'our', 'con', 'bottom', 'amongst', 'so', 'couldn', 'you', 'co', 'something', 'mustn't', 'anyone', 'together', 'within', 'o', 'until', 'six', 'should've', 'thereupon', 'eleven', 'others', 'anyway', 'how', 'isn't', 'never', 'shouldn't', 'nine', 'nevertheless', 'ourselves', 'somehow', 'must', 'this', 'hereafter', 'third', 'ie', 'we', 'became', 'do', 'via', 'moreover', 'other', 'move', 'rather', 'each', 'above', 'nowhere', 'he's', 'anything', 'mightn', 'part', 'doing', 'everywhere', 'than', 'before', 'becoming', 'me', 'last', 'won', 'theirs', 'onto', 'may', 'upon', 'beforehand', 'hundred', 'in', 'after', 'ain', 'needn't', 'very', 'fifteen', 'anywhere', 'sixty', 'behind', 'where', 'whether', 'also', 'out', 'most', 'amount', 'back', 'against', 'all', 'from', 'us', 'cant', 'who', 'seemed', 'what', 'sincere', 'of', 'why', 'it'd', 'are', 'you're', 'yourself', 'found', 'anyhow', 'due', 'will', 'thin', 'go', 'was', 'less', 'then', 'every', 'more', 'once', 'several', 'no', 'amongst', 'there', 'i've', 'we'd', 'her', 're', 'be', 'ma', 'forty', 'mill', 'they're', 'were', 'noone', 'i'll', 'name', 'his', 'that'll', 'perhaps', 'we're', 'does', 'she'd', 'herself', 'itself', 'if', 'seems', 'though', 'or', 'among', 'my', 'three', 'when', 'least', 'that', 'wasn't', 'twenty', 'such', 'thence', 'she'll', 'between', 'many', 'their', 'thus', 'he'd', 'thereby', 'hers', 'another', 'is', 'weren', 'she's', 'not', 'per', 'any', 'couldn't', 'almost', 'five', 'yourselves', 'shouldn', 'they', 'your', 'often', 'done', 'doesn't', 'find', 'am', 'at', 'we've', 'wherein', 'everyone', 'none', 'since', 'its', 'weren't', 'through', 'and', 'those', 'having', 'he'll', 'eg', 'by', 'former', 'alone', 'somewhere', 'except', 'give', 'without', 'to', 'hasn', 'describe', 'thru', 'a', 'it'll', 'formerly', 'off', 'besides', 'beyond', 'needn', 'mostly', 'shan', 'doesn', 'haven't', 'hadn't', 'top', 'ten', 'hadn', 'same', 'haven', 'however', 'along', 'next', 'on', 'mine', 'seem', 'become', 'still', 'fire', 'ltd', 'etc', 'nor', 'seeming', 'hereby', 'some', 'we'll', 'whose', 'well', 'whereby', 'won't', 'could']

2. Κώδικας της κλάσης Keyword

```
class Keyword:
    def __init__(self, txt):
        self.txt= txt
        self.length = len(txt.split(' '))
        self.score=0

        # used for tf algorithms
        self.count=0 # occurencies; used also in Yake
        self.sent_in=set() # it is a set with the indx of sentences the kwd is in; used also in Yake
        self.tf=0 # term frequency

        # Used for Yake scratch implementation
        self.start_capital = 0
        self.is_uppercase = 0 # if string is uppercase
        self.positions=[] # it is a list with the positions the kwd has in the sentences
        self.r_neighbors=[] # list of strings => right neighbors of the kwd
        self.l_neighbors=[] # list of strings => left n
```

3. Κώδικας της κλάσης Algorithm

```
from nltk.stem import PorterStemmer
from nltk.util import ngrams

class Algorithm:
    """ NLP PUBLIC STATIC METHODS """
    @staticmethod
    def tokenize_with_newlines(text, stopwords, lowercase):
        """
        Tokenizes text into words, punctuation, and newlines.
        - Words => \w+,
        - Punctuation => [^\w\s],
        - Newlines => \n,
        """
        import re
        # Use regex to match words, punctuation, or newlines and keep '-' inside the word (e.g co-operation)
        tokens = re.findall(r'\w+(?:-\w+)*| [^\w\s]|\n', text)
        if not lowercase:
            return [t for t in tokens if t.lower() not in stopwords]
            return [t.lower() for t in tokens if t.lower() not in stopwords]

        @staticmethod
        def make_sentences(tokenized_text):
            """
            The method returns the sentences of a tokenized text as list of lists.
            Each sublist is a list of tokens belonging to the same sentence
            """
```

```
# string.punctuation: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
# string.whitespace: '\t\n\x0b\x0c\r '
'''
import string
final_list=[]
sentence=[]
for t in tokenized_text:
    if t in string.punctuation:
        if len(sentence)>0:
            final_list.append(sentence)
            sentence=[]
    elif t not in string.whitespace:
        sentence.append(t)
if len(sentence)>0:
    final_list.append(sentence)

return final_list

@staticmethod
def extract_all_n_grams(tokens_list, n_gram_size, min_n=1):
    ''' It returns a list of lists
        each sublist is a list of relative n-grams
        params: tokens_list the list of tokens (should be considered as unigrams) to get their n_grams
               n_gram_size: number of n (unigrams=1, bigrams=2, trigrams=3 ...)
               min_n: min number of n (e.g min_n=3 => trigrams and above are returned)
    '''
    real_min_n=min_n
    if real_min_n<1:
        real_min_n=1

    all_n_grams=[]
    for i in range(n_gram_size):
        n_grams=list(ngrams(tokens_list, i+1)) # they are returned as a list of tuples (each tuple is the
                                                unigrams)

        if len(n_grams)>0: # if i_grams cannot be created (e.g 'john', 'deer' can not create trigrams) -> it
                           returns empty list
            n_gram_strings=[' '.join(ng) for ng in n_grams]
            all_n_grams.append(n_gram_strings)
    return all_n_grams[real_min_n-1:]

@staticmethod
def porter_stem(phrase):
    ''' It returns a phrase stemmed by Porter's stemmer '''
    stemmer=PorterStemmer()
    return " ".join(stemmer.stem(word) for word in phrase.split())

def __init__(self, text):
    '''
        :param text it might be a single string or a list of strings
    '''
    self._kwd_dict_indx = {} # a dict with key, value -> token_str, index of Keyword obj in kwd_list
    self._kwd_list = []
```

```
self._text=self._text_to_single_string(text) # the text will be always a single string
self._default_state=None # It will be used only for the metrics as a store point of the default behavior

''' PROTECTED METHODS '''
def _add_kwd_to_list(self, kwd_text):
    ''' The following method adds a Keyword object to the kwd_list and returns it for further processing
        If the Keyword object exists, it just returns it
    '''
    indx = self._kwd_dict_indx.get(kwd_text)
    if not indx is None: # kwd object exists
        return self._kwd_list[indx]
    self._kwd_dict_indx[kwd_text]=len(self._kwd_list)
    self._kwd_list.append(Keyword(kwd_text))
    return self._kwd_list[len(self._kwd_list)-1]

def _stem_results(self):
    '''
        The following method returns a list [Token list, indx_dict] of stemmed keywords
        If two or more keywords give the same stemmed version,
        the max of their scores will be considered as the score of the stemmed version
    '''
    new_kwd_dict_indx = {}
    new_kwd_list = []
    for kwd in self._kwd_list:
        # print(kwd.score)
        stemmed = Algorithm.porter_stem(kwd.txt)
        stemmed_indx = new_kwd_dict_indx.get(stemmed)
        if stemmed_indx is None: # stemmed shown for 1st time
            new_kwd_dict_indx[stemmed] = len(new_kwd_list) # add to indx_dict
            new_kwd = Keyword(stemmed) # create the stemmed kwd and give the score of the parent
            new_kwd.score = kwd.score
            new_kwd_list.append(new_kwd) # append to the stemmed kwd list

        else: # stemmed already exists; It will get the score of the heavier parent kwd
            if new_kwd_list[stemmed_indx].score < kwd.score:
                # old_score= new_kwd_list[stemmed_indx].score
                new_kwd_list[stemmed_indx].score = kwd.score
    return [new_kwd_list, new_kwd_dict_indx]

def _text_to_single_string(self, text_list):
    '''
        The following method returns the text argument as a single string
        if the text is single string it just returns it
        if the text is a list of strings (e.g. list paragraphs) it joins it to a single string
        Raises exception if the argument is not a string or list of strings
    '''
    if isinstance(text_list, (list, tuple, set)):
        return ' '.join(text_list)
    if isinstance(text_list, str):
        return text_list
    raise Exception('argument is not string or list of strings')
```

```
def _text_to_list_string(self, text):  
    """  
    The following method returns the text argument as list of strings (e.g. list of paragraphs)  
    if the text is a list of strings it returns that list  
    if the text is single string it splits it based on '\n'  
    Raises exception if the argument is not a string or list of strings  
    """  
    if isinstance(text, (list, tuple, set)):  
        return text  
    if isinstance(text, str):  
        return text.split('\n')  
    raise Exception('argument is not string or list of strings')  
  
    """ PUBLIC METHODS """  
  
def get_kwd_list(self, descending=True, top_n=None):  
    """  
    The following function returns the sorted kwds as a list of tuples (kwd_string, score)  
    :param top_n the fraction of the kwd_list to be returned; it must be between 0 and 1  
    """  
    sorted_list = sorted(self._kwd_list, key=lambda token: token.score, reverse=descending)  
    if top_n==None or top_n<=0 or top_n>=1:  
        # Some algorithms (e.g Keybert) may give negative scores for tokens with opposite meaning;  
        # we ignore those tokens  
        return [(x.txt, x.score) for x in sorted_list if x.score>0]  
  
    # if top_n is a real fraction  
    sorted_list = [kwd for kwd in sorted_list if kwd.score > 0]  
    top_items=int(len(sorted_list) * top_n)  
    if top_items==0:  
        top_items=1  
  
    top_list= sorted_list[:top_items]  
    remaining_list = sorted_list[top_items:]  
    ties=[] # these are items that perhaps have the same value with the last item of the top_list  
    for element in remaining_list:  
        if abs(element.score - top_list[len(top_list)-1].score) < 0.001:  
            ties.append(element)  
        else:  
            break  
    return [(x.txt, x.score) for x in top_list+ties]  
  
def get_kwd_list_normalized(self, descending=True, top_n=None):  
    """ The following function returns the sorted kwds as a list of tuples (kwd_string, normalized_score) """  
    kwd_tuples = self.get_kwd_list(descending, None)  
  
    #Calculate the sqrt of scores' squared sum  
    sq_sum=0  
    for kwd_t in kwd_tuples:  
        sq_sum+=(kwd_t[1] * kwd_t[1])
```

```
sq_sum_root = math.sqrt(sq_sum)

# Normalize all scores
normalized_list=[]
for kwd_t in kwd_tuples:
    normalized_list.append((kwd_t[0], kwd_t[1]/sq_sum_root))

# return the whole list of tuples
if top_n==None or top_n<=0 or top_n>=1:
    return normalized_list

# if top_n is a real fraction return the real fraction including ties
top_items=int(len(normalized_list) * top_n)
if top_items==0:
    top_items=1
top_list = normalized_list[:top_items]
remaining_list = normalized_list[top_items:]
ties = [] # these are items that perhaps have the same value with the last item of the top_list
for element in remaining_list:
    if abs(element[1] - top_list[len(top_list) - 1][1]) < 0.001:
        ties.append(element)
    else:
        break
# append the ties to the top_list
return top_list + ties
```

4. Κώδικας της κλάσης myRake (Algorithm)

```
from rake_nltk import Rake # pip install rake-nltk

class myRake(Algorithm):
    def __init__(self, text, n_grams, stop_words, n_gram_score='default', stemmer=False):
        super().__init__(text)
        if n_gram_score is None or n_gram_score.lower() not in ['default', 'normalize']:
            raise Exception(f'n_gram_score must be either default or normalizwe')

        # Initialize RAKE
        rake = Rake(
            stopwords=stop_words,
            min_length=1,
            max_length=30, # Rake internally ignores sentences >max_length. So I put 30 here to ensure
                           # unigrams and actual n_grams will be handled afterwards
            include_repeated_phrases=True,
        )

        # Extract keywords -make the Rake run
        rake.extract_keywords_from_text(self._text)

        # If I want to tweak further I can get RAKE score for each unigram as score= degree/frequency
        degree = rake.degree # co-ccurence value per unigram
        freq = rake.frequency_dist # frequency of each unigram
        self._default_state=rake # Store the algorithm that run for 1st time

        # Get raw phrases and scores; phrases will be up to max_length;
        raw_phrases = rake.get_ranked_phrases_with_scores() # list of tuples (score, phrase)

        # Get also the n_grams of each raw_phrase (phrases may be repeated) # it will be stored as set
        raw_n_grams = set()
        for s, phrase in raw_phrases:
            n_gram_list = self.extract_all_n_grams(phrase.split(' '), n_grams, min_n=2)
            raw_n_grams.update([x for p in n_gram_list for x in p])

        # Now add only the unigrams and their scores (I will use degree/frequency)
        for t, value in degree.items():
            try:
                score=value/freq[t]
                added = self._add_kwd_to_list(t)
                added.score=score
            except:
                print(f'Exception for adding {t} with degree {value}')
                pass

        #Finally add the n-grams and their scores (default is sum of unigrams)
        if n_gram_score.lower()=='default':
            for phrase in raw_n_grams:
```

```
self.__sum_rake_n_grams(phrase)

if n_gram_score.lower() == 'normalize':
    weights = [n_grams - i for i in range(n_grams)] # [3, 2, 1]
    total = sum(weights)
    weights = [w / total for w in weights] # normalize so the sum of weights =1 => [0.5, 0.33, 0.166]
    for phrase in raw_n_grams:
        self.__normalize_rake_n_grams(phrase, weights)

if stemmer:
    self._kwd_list, self._kwd_dict_idx = self._stem_results()

''' PRIVATE METHODS '''
def __sum_rake_n_grams(self, n_gram_string):
    ''' Even it is destined for phrases, it works also if n_gram_string is unigram
        The score of the unigram (if exists) is added to 0
    '''
    unigrams = n_gram_string.split(' ')
    n_gram_score = 0
    for unigram in unigrams:
        try:
            n_gram_score += self._kwd_list[self._kwd_dict_idx[unigram]].score
        except:
            pass
    if n_gram_score > 0: # if i do have a score
        added = self._add_kwd_to_list(n_gram_string)
        added.score = n_gram_score

def __normalize_rake_n_grams(self, n_gram_string, weights):
    ''' The weights is a normalized weights (sum of weights =1)
        based on position of a unigram in the n-gram string'''
    unigrams = n_gram_string.split(' ')
    n_gram_score = 0
    for pos, unigram in enumerate(unigrams):
        try:
            n_gram_score += weights[pos] * self._kwd_list[self._kwd_dict_idx[unigram]].score
        except:
            pass
    if n_gram_score > 0: # if i do have a score
        added = self._add_kwd_to_list(n_gram_string)
        added.score = n_gram_score
```

5. Κώδικας της κλάσης myYake (Algorithm)

```
class myYake(Algorithm):
    def __init__(self, text, n_grams, stop_words, stemmer=False):
        super().__init__(text)

        #Document specific statistics
        self.doc_word_counts=[]
        self.doc_word_max_count=0
        self.doc_mean_counts=0
        self.doc_stdv_counts=0

        # Read and make the list of sentences; each sentence is a list of n_grams
        # Stemming occurred when reading the text
        self.n_gram_sents=self._preprocess(self._text, stop_words, n_grams, stemmer)

        # Yake applies to unigrams, so only the 1st element of the self.n_gram_sents will be used for now
        for sent_indx, sent in enumerate(self.n_gram_sents):
            for uni_indx, unigram in enumerate(sent[0]):
                kwd = self._add_kwd_to_list(unigram.lower()) # Create/update the kwd object
                kwd.count+=1 # update count
                kwd.sent_in.add(sent_indx) # update sentences in (the indx of a sentence containing the kwd)
                kwd.positions.append(uni_indx+1) # update the index (position) of the kwd in a particular
                                                #sentence

                if unigram.isupper(): # update the casing status
                    kwd.is_uppercase+=1
                elif unigram[0].isupper():
                    kwd.start_capital+=1 # starts with capital?
                #Update the neighbors
                kwd.l_neighbors+= [sent[0][x].lower() for x in range(uni_indx)] # words before the unigram
                kwd.r_neighbors+= [sent[0][x].lower() for x in range(uni_indx+1, len(sent[0]))] # words after the
                                                #unigram

        # Now run again the unigrams to get the document statistics
        for kwd in self._kwd_list:
            self.doc_word_counts.append(kwd.count)
            if kwd.count > self.doc_word_max_count:
                self.doc_word_max_count=kwd.count
            self.doc_mean_counts=statistics.mean(self.doc_word_counts)
            self.doc_stdv_counts=statistics.stdev(self.doc_word_counts)

        # Run again to calculate score for the unigrams; Yake the lower the score the better
        for kwd in self._kwd_list:
            kwd.score = self._run_Yake_unigrams(kwd)

        # Run again to the bigrams etc so that calculate their score based on their unigrams;
        # 1st calculate the counts of each phrase; we need the count to feed the n_gram calculation formula
        for sent in self.n_gram_sents:
            for i in range(1, len(sent)):
                for n_gram in sent[i]: # Now bi-grams and higher are considered
```

```
kwd_phrase = self._add_kwd_to_list(n_gram.lower())
kwd_phrase.count+=1

#2nd calculate the scores of the n-grams; Yake the lower the score the better
for kwd in self._kwd_list:
    if kwd.length>1:
        kwd.score = self._run_Yake_n_phrase(kwd_phrase)

# 3 Finally inverse the scores to be aligned with the descending order;
#Yake scores significant kwds with lower score
for kwd in self._kwd_list:
    try:
        kwd.score = 1/kwd.score
    except:
        kwd.score=0

def _preprocess(self, text, stopwords, n_gram_size, stemmer):
    """ The following method returns the sentences of the text
    as a list. Stemming also occurs in that method.
    Each sentence list has length up to n_gram_size
    Each sentence sublist corresponds to its different n-gram lists

    e.g. "My name is Bill. What is your name?" for n_gram_size=3

    sent1 = [['my', 'name', 'is', 'bill'], ['my name', 'name is', ..], ['my nane is', 'name is Bill]]
    sent2 = [['what', 'is', 'your', 'name'], ['what is', 'is your' ..], ['what is your, 'is your name']]

    returns: [sent1, sent2]
    """

    tokens = Algorithm.tokenize_with_newlines(text, stopwords, lowercase=False) # tokenize
    sentences = Algorithm.make_sentences(tokens) # sentences as list of unigram list
    n_gram_sents=[] # the returned list of n-gram sentences

    for sent in sentences:
        n_grams=[]
        if stemmer==False:
            n_grams=Algorithm.extract_all_n_grams(sent, n_gram_size, 1)
        else:
            stemmed=[Algorithm.porter_stem(t) for t in sent]
            n_grams = Algorithm.extract_all_n_grams(stemmed, n_gram_size,1)
        n_gram_sents.append(n_grams)

    return n_gram_sents

""" Calculating Yake's score components and final score """

def __calculate_casing(self, kwd):
```

```

"""
    max(count_start_capital, count_is_acronym)
    casing = -----
                total word count
"""

nominator = max(kwd.start_capital/kwd.count, kwd.is_uppercase/kwd.count)
try:
    denominator = kwd.count
except:
    denominator = 1
return nominator / denominator

def __calculate_positional(self, kwd):
    """
    positional = ln [ln(3+median(keyword_positions))] """

    try:
        return math.log(math.log(3 + statistics.median(kwd.positions)))
    except:
        print('exception raised during calculating positional in ', kwd.txt)
        return 0

def __calculate_word_frequency(self, kwd):
    """
        count_of_word
        frequency = -----
                mean_counts + stdev_counts
    """

    try:
        return kwd.count / (self.doc_mean_counts + self.doc_stdv_counts)
    except:
        return 0

def __calculate_relatedness_to_context(self, kwd):
    """
        count_of_word
        relatedness = 1+ (WR+WL) * ----- + PL+ PR
                        max_count

    where:
        number of unique words in right(left)
        WR(WL) = -----
                total number of words in right(left)

        total number of words in right (left)
        PR(PL) = -----
                max_count
    """

    try:
        pl = len(kwd.l_neighbors) / self.doc_word_max_count
    except:

```

```
pl = 0

try:
    pr = len(kwd.r_neighbors) / self.doc_word_max_count
except:
    pr = 0

try:
    wl = len(set(kwd.l_neighbors)) / len(kwd.l_neighbors)
except:
    wl = 0

try:
    wr = len(set(kwd.r_neighbors)) / len(kwd.r_neighbors)
except:
    wr = 0

res = 1 + pl + pr
res += (wl + wr) * kwd.count / self.doc_word_max_count
return res

def __calculate_different_sentence(self, kwd):
    """
        number of sentences the keyword occurs in
        different_sentence_score = -----
                                total number of sentences
    """

    return len(kwd.sent_in) / len(self.n_gram_sents)

def _run_Yake_unigrams(self, kwd):
    """
        relatdness_to_context * word_positional
        yake_single_score = -----
                        word_frequency + word_different_sentence
                        casing + -----
                                relatdness_to_context
    """

    relatedness = self.__calculate_relatedness_to_context(kwd)
    positional = self.__calculate_positional(kwd)
    casing = self.__calculate_casing(kwd)
    frequency = self.__calculate_word_frequency(kwd)
    difference = self.__calculate_different_sentence(kwd)

    return (relatedness * positional) / (casing + ((frequency + difference) / relatedness))

def _run_Yake_n_phrase(self, kwd_phrase):
    """
        Π (single_word_score)
        YAKE_keyphrase_score = -----
                        kwd_phrase_count * (1+ Σ(single_word_score) )
    """
```

```
'''  
components= kwd_phrase.txt.split(' ')  
c_product = 1  
c_sum=0  
for c in components:  
    c_product*=self._kwd_list[self._kwd_dict_indx[c]].score  
    c_sum+=self._kwd_list[self._kwd_dict_indx[c]].score  
  
return c_product/(kwd_phrase.count*(1+c_sum))
```

6. Κωδικας της κλάσης myYake από έτοιμο πακέτο της Python (yake)

```
import yake #pip install yake

class myYake_from_pip (Algorithm):
    def __init__(self, text, n_grams, stop_words, stemmer=False):

        super().__init__(text)
        yake_extr = yake.KeywordExtractor(
            lan='en',
            n=n_grams,
            stopwords=stop_words,
            top=1000000000000000, # initially extract many, we will filter/sort manually
            features=None
        )

        # Extract keywords (score: lower is better) - Make Yake runs
        keywords = yake_extr.extract_keywords(self._text)

        #Looping through keyword tuples
        for phrase, score in keywords:
            inv_score=0
            if score!=0:
                inv_score=1/score
            cleaned = self.__clean_yake_key_phrase(phrase, stop_words).lower() # clean and get the lower
            added=self._add_kwd_to_list(cleaned) # return the added
            if added.score==None or added.score<inv_score:
                added.score=inv_score

        if stemmer:
            self._kwd_list, self._kwd_dict_indx = self._stem_results()

    def __clean_yake_key_phrase(self, phrase, stop_words):
        """ The method gets a phrase from yake and returns it cleaned from stopwords if it contains"""
        cleared=""
        for str in phrase.split(' '):
            if str.lower() not in stop_words:
                cleared+=str+' '
        return cleared[:len(cleared)-1] # omit the last space
```

7. Κώδικας της κλάσης myTextRank (Algorithm)

```
import networkx as nx # Graph creator for TextRank: pip install networkx

class myTextRank(Algorithm): # Using the nx.pagerank
    def __init__(self, text, n_grams, stop_words, directed,
                  n_gram_score='default',
                  n_gram_weights=[1,4,6], stemmer=False, window_size=4):
        if n_gram_score is None or n_gram_score.lower() not in ['default', 'weighted']:
            raise Exception(f'n_gram_score must be either default or weighted')

        super().__init__(text)
        #Preprocess: tokenize, remove stopwords and create n_grams
        candidates = self._preprocess(self._text, stop_words, n_grams)

        #Create the graph of the text
        graph = self._build_graph(candidates, directed, window_size)

        #Run the algorithm on the graph
        scores = nx.pagerank(graph) # it is a dict

        self._default_state = scores

        # Fill in the kwd_list/kwd_dict with 'default' values (those returned by pagerank)
        for kwd, val in scores.items():
            added = self._add_kwd_to_list(kwd)
            added.score=val

        """ If we must follow the Rajput et al 2020, N-Grams TextRank : A Novel Domain Keyword Extraction
            Technique approach the scores dict must be updated, so bigrams are scored 4-fold their weight
            and tri-grams 6 fold their weight
        """

        if n_gram_score == 'weighted':
            self._weight_textrank_results(weights=n_gram_weights)

        if stemmer:
            self._kwd_list, self._kwd_dict_idx = self._stem_results()

    def _preprocess(self, text, stopwords, n_gram_size):
        """ The following method returns a list of tokenized sentences (lowercased)
            with stopwords removed. Also, each sentence list contains the unigrams, bigrams trigrams etc
            of that sentence
        """

        tokens = Algorithm.tokenize_with_newlines(text, stopwords, lowercase=True) # tokenize
        sentences=Algorithm.make_sentences(tokens) # sentences as list of unigram list
        if n_gram_size>1:
            n_gram_sentences=[]
            for sent in sentences:
```

```
n_grams = Algorithm.extract_all_n_grams(sent, n_gram_size, min_n=1)
n_gram_list=[]
for n_gram in n_grams:
    n_gram_list+=n_gram # I want each sentence to be unigrams, bigrams, trigrams
n_gram_sentences.append(n_gram_list)
return n_gram_sentences
return sentences

def _build_graph(self, candidates, directed, window_size):
    """ Graph is build based on the approach followed
    by Rajput et al 2020, N-Grams TextRank : A Novel Domain Keyword Extraction Technique
    where n-grams were incorporated to the graph as single nodes.
    N-grams where created for each sentence of the text. So each sentence was
    a list [unigrams, bigrams, trigrams ...]
    """
    graph = nx.Graph()
    if directed:
        graph = nx.DiGraph()
    for sent in candidates:
        for i, node in enumerate(sent):
            for j in range(i + 1, i + window_size):
                if j >= len(sent):
                    break
                graph.add_edge(node, sent[j])
    return graph

def _weight_textrank_results(self, weights):
    """ The method scores with a different weight the n-grams based on their length
    param weights-> a list of weights for each n-gram.
    If n> len(weights) the higher n will get the score of the last weight item
    e.g. weights=[1,4,6] =>
        unigrams =1*default_score,
        bigrams =4*default_score,
        trigrams and more = 6* default_score
    if an empty list is provided, the scores will not change """
    if len(weights)>0:
        for kwd in self._kwd_list:
            if kwd.length>len(weights):
                kwd.score *=weights[len(weights) -1]
            else:
                kwd.score *=weights[kwd.length -1]
```

8. Κώδικας της κλάσης myKeyBert (Algorithm)

```
"""
KEYBERT => always load the model once and pass it as parameter to myKeyBert objects !!!
"""

from keybert import KeyBERT # from pip install keybert and pip install sentence-transformers

def load_keybert_model(model_name, local_path):
    """
    Tries to load KeyBERT model from Hugging Face.
    If it fails, loads from a local folder.
    The method returns the keybert model
    """
    import socket
    def __get_local_model(local_path, e):
        print(f"Failed to load Keybert online model: {e}")
        print(f"Falling back to Keybert local model at '{local_path}'...")
        if not os.path.exists(local_path):
            raise FileNotFoundError(
                f"Local model folder '{local_path}' does not exist. "
                "You need to download the model at least once."
            )
        kw_model = KeyBERT(model=local_path)
        print("Keybert local model was loaded successfully!")
        return kw_model

    # Establish connection with huggingface.co
    try:
        socket.create_connection(("huggingface.co", 443), timeout=1.0)
    except OSError as e:
        print('No connection can be established with huggingface.co')
        return __get_local_model(local_path, e)

    try:
        # Try online Hugging Face model
        print(f"Trying to load Keybert online model '{model_name}'...")
        kw_model = KeyBERT(model_name)
        print("Keybert online model was loaded successfully!")
        return kw_model
    except Exception as e:
        return __get_local_model(local_path, e)

class myKeyBert(Algorithm):

    def __init__(self, text, n_grams, stop_words, keybert_model, stemmer=False):
        """
        :params keybert_model the model returned from load_keybert_model(model_name, local_path)
        """
        super().__init__(text)
```

```
# Extract keywords using the keybert model
keywords = keybert_model.extract_keywords(self._text, stop_words=stop_words,
                                         keyphrase_ngram_range=(1, n_grams),
                                         top_n=50000000000)

for kwd, val in keywords:
    if val>0: # Sometimes the algorithm gives negative values for totally unrelated tokens
        added = self._add_kwd_to_list(kwd)
        added.score = val

if stemmer:
    self._kwd_list, self._kwd_dict_idx = self._stem_results()
```

9. Κώδικας της κλάσης InvertSentFreq (Algorithm)

```
class InvertSentFreq(Algorithm):
    def __init__(self, text, n_grams, stop_words, stemmer=False):
        super().__init__(text)
        self._n_gram_sents = self._preprocess(self._text, stop_words, n_grams, stemmer)
        self._total_sents = [0 for x in range(n_grams)] # total number of sentences per n-gram size
        self._total_n_grams = [0 for x in range(n_grams)] # total number of tokens per n-gram size

        # Create the _kwd_list / kwd_dict
        for sent in self._n_gram_sents:
            for indx, tokens in enumerate(sent):
                # Update the total number of sentences for the respective n-gram
                self._total_sents[indx] += 1
                for token in tokens:
                    added = self._add_kwd_to_list(token)
                    self._total_n_grams[added.length-1] += 1 # update the total number of the respective n-gram
                    added.count += 1 # increase count
                    # if kwd appears more than once in a sentence, sent_in will be updated once
                    added.sent_in.add(indx)

        # Finally update term frequencies and tfxidf score
        self._update_freq_score()

    def _preprocess(self, text, stopwords, n_gram_size, stemmer):
        """ The following method returns the sentences of the text
            as a list. Stemming also occurs in that method.
            Each sentence list has length up to n_gram_size
            Each sentence sublist corresponds to its different n-gram lists

            e.g. "My name is Bill. What is your name?" for n_gram_size=3

            sent1 = [['my', 'name', 'is', 'bill'], ['my name', 'name is', ..], ['my name is', 'name is Bill]]
            sent2 = [['what', 'is', 'your', 'name'], ['what is', 'is your' ..], ['what is your', 'is your name']]

            returns: [sent1, sent2]
        """

        tokens = Algorithm.tokenize_with_newlines(text, stopwords, lowercase=True) # tokenize

        sentences = Algorithm.make_sentences(tokens) # sentences as list of unigram list

        n_gram_sents = [] # the returned list of n-gram sentences

        for sent in sentences:
            n_grams = []
            if stemmer == False:
                n_grams = Algorithm.extract_all_n_grams(sent, n_gram_size, 1)
            else:
                stemmed = [Algorithm.porter_stem(t) for t in sent]
                n_grams = Algorithm.extract_all_n_grams(stemmed, n_gram_size, 1)
            n_gram_sents.append(n_grams)
```

```
return n_gram_sents

def _update_freq_score(self):
    """
    The method calculates the TF of each Keyword object and also
    updates the score of each Keyword by applying TF X ISF
    where ISF = log10 (N/n), N -> total number of sentences, n -> number of sentences the kwd is in
    """
    for kwd in self._kwd_list:
        # Update frequency
        if self._total_sents[kwd.length-1]!=0:
            kwd.tf = kwd.count/self._total_n_grams[kwd.length-1]
        else:
            print(f'total n_gram =0 when {kwd.txt} found!!!!')
            kwd.tf=0 # Just for not crashing

        # Update tfxidf score
        try:
            kwd.score= kwd.tf * math.log10(self._total_sents[kwd.length-1]/len(kwd.sent_in))
        except:
            kwd.score=0

        # If there is just one sentence => tf- isf = tf
        if kwd.score==0 and self._total_sents[kwd.length-1] == 1:
            kwd.score = kwd.tf

    def get_frequencies(self):
        """
        The method returns a list of tuples ('word', frequency')
        It will be used for TFXIDF
        """
        freq_list=[]
        for kwd in self._kwd_list:
            freq_list.append((kwd.txt, kwd.tf))
        return freq_list
```

10. Κώδικας της κλάσης Journal_App

```
''' *****'''

MAIN LANDING PAGE AND STATE LOADER

*****'''

import streamlit as st
import os
import sys
import session_manager as sm

class Journal_App:
    def __init__(self):
        # SETTINGS
        BASE_FOLDER = os.path.dirname(os.path.abspath(__file__))
        CONFIG_FILE = "config.json"
        DATA_BASE = os.path.join(BASE_FOLDER, 'database', 'app_db.db')
        KEYBERT_MODEL_NAME= 'all-MiniLM-L6-v2'
        KEYBERT_LOCAL_PATH = os.path.join(BASE_FOLDER, 'src', 'kwd_algorithms', 'keybert_models',
                                           KEYBERT_MODEL_NAME)

        APP_VIEWS=['base', 'login', 'suggest', 'update_journal', 'update_paper', 'update_user']
        APP_SUBVIEWS=['update_journal_del',
                     'update_paper_del', 'update_paper_add',
                     'update_user_del', 'update_user_add', 'update_user_current', 'update_user_admin',
                     ] # subviews are inside tabs

        # Create the state manager
        self.sm = sm.Session_Manager(db_path=DATA_BASE,
                                     config_file = CONFIG_FILE,
                                     app_views=APP_VIEWS+APP_SUBVIEWS,
                                     keybert_model_name=KEYBERT_MODEL_NAME,
                                     keybert_local_path=KEYBERT_LOCAL_PATH)

    # METHODS
    def _render_navbar(self):
        ''' Renders the navbar it gets a session_stage parameter'''
        with st.sidebar:
            # always visible
            if st.button("🏠 Login", type='tertiary'):
                self.sm.set_current_view(self.sm.VIEWS['login'])
            if st.button("🔍 Publish Suggestions", type='tertiary'):
                self.sm.set_current_view(self.sm.VIEWS['suggest'])

            # admin submenu
            if self.sm.USER_ROLE=='admin' or self.sm.USER_ROLE=='user':
                if self.sm.USER_ROLE=='admin':
                    with st.expander("⚙️ Admin tools"):
                        if st.button("👤 Update User", type='tertiary'):
                            self.sm.set_current_view(self.sm.VIEWS['update_user'])
```

```
if st.button("📄 Update Journal", type='tertiary'):
    self.sm.set_current_view(self.sm.VIEWS['update_journal'])
if st.button("📄 Update Paper", type='tertiary'):
    self.sm.set_current_view(self.sm.VIEWS['update_paper'])

if self.sm.USER_ROLE=='user':
    with st.expander("⚙️ User tools"):
        if st.button("👤 My Profile", type='tertiary'):
            self.sm.set_current_view(self.sm.VIEWS['update_user'])
        if st.button("📄 Update Journal", type='tertiary'):
            self.sm.set_current_view(self.sm.VIEWS['update_journal'])
        if st.button("📄 Update Paper", type='tertiary'):
            self.sm.set_current_view(self.sm.VIEWS['update_paper'])

# logout
st.divider()
if st.button("🚪 Logout", type='tertiary'):
    # Force reset all views:
    for view, view_obj in self.sm.VIEWS.items():
        view_obj.reset()

    # Set user as visitor
    self.sm.set_user_role('visitor')
    self.sm.set_username("")

    # redirect to the login page
    self.sm.set_current_view(self.sm.VIEWS['login'])
    st.rerun()

def run(self):
    if self.sm.check:
        # 1. Important!!! Refresh the views with the current sm
        self.sm.apply_to_views()

        # 2. Get the view from the navbar
        self._render_navbar()

        # 3. render that view
        self.sm.CURRENT_VIEW.render()

# Run the app
app = Journal_App()
app.run()
```

11. Κώδικας της κλάσης Session_Manager

```
import streamlit as st
import src.app_utils as utils
import src.db_management as db
import src.kwd_algorithms.algorithms as algo

class Session_Manager():
    """
    A class storing the really needed shared variables and the page objects
    the state manager will start as soon as the app loads its external sources
    -existing db path,
    -config_file was loaded successfully
    - pages or views: a list of the pages (views) of the app
    """
    def __init__(self, db_path, config_file, app_views, keybert_model_name=None,
                 keybert_local_path=None):

        #DEFINE THE SETTINGS OF THE APP (CAPITAL LETTERS)
        if 'loading_assets' not in st.session_state:
            st.session_state['loading_assets']=True

        # 1. ----- LOAD EXTERNAL COMPONENTS -----
        # a) Load the configurations file
        if 'CONFIG' not in st.session_state:
            config_dict = utils.load_json(config_file)
            if config_dict == None:
                st.error(f'Fatal error: Configuration file {config_file} could not be loaded')
            else:
                st.write(f'Configuration file {config_file} loaded successfully')
                self.CONFIG = st.session_state['CONFIG'] = config_dict # a dict
        else:
            self.CONFIG = st.session_state['CONFIG']

        # b) Test the connection to the database
        db_connection=True
        if 'DB_PATH' not in st.session_state:
            try:
                db.connect_to_sqlite3(db_path)
                st.write(f'Connection to database was successful')
            except:
                db_connection = False
                st.error(f'Fatal error: Database connection with {db_path} could not be initialized')

            self.DB = st.session_state['DB_PATH'] = db_path # a string
        else:
            self.DB = st.session_state['DB_PATH']

        # c) Load the keybert model
        if 'KB_MODEL' not in st.session_state:
            with st.spinner(f' Loading Keybert model {keybert_model_name} ...'):
```

```
try:
    kb_model = algo.load_keybert_model(model_name=keybert_model_name,
                                       local_path=keybert_local_path)
    st.write(f'Keybert model {keybert_model_name} was loaded successfully')

except:
    kb_model = None
    st.error(f'Fatal error: Keybert model {keybert_model_name} could not be initialized')

self.KB_MODEL=st.session_state['KB_MODEL'] = kb_model # the model
else:
    self.KB_MODEL = st.session_state['KB_MODEL']

# d) Final check
self.check= self.CONFIG and db_connection and self.KB_MODEL

# Now rerun to get rid of the loading assets messages
if st.session_state['loading_assets'] and self.check:
    st.session_state['loading_assets']=False
    st.rerun()

# 2. ----- LOAD PAGES -----

# Load the views
if 'VIEWS' not in st.session_state:
    self.VIEWS = st.session_state['VIEWS'] = {} # It will hold the views of the app (pages);
                                                # a view can be accessed by sm.VIEWS['page_name']

page=None
for view in app_views:
    if view == 'base':
        from views.base import BaseView
        page=BaseView(view, self)
    if view == 'login':
        from views.login import LoginView
        page = LoginView(view, self)
    if view == 'suggest':
        from views.suggest import SuggestView
        page = SuggestView(view, self)
    if view == 'update_db':
        from views.update_db import UpdateDbView
        page = UpdateDbView(view, self)
    if view == 'update_journal':
        from views.update_journal import UpdateJournalView
        page = UpdateJournalView(view, self)
    if view == 'update_paper':
        from views.update_paper import UpdatePaperView
        page = UpdatePaperView(view, self)
    if view == 'update_paper_add':
        from views.update_paper_add import AddPaperView
        page = AddPaperView(view, self)
```

```
if view == 'update_paper_del':
    from views.update_paper_del import DeletePaperView
    page = DeletePaperView(view, self)
if view == 'update_user':
    from views.update_user import UpdateUserView
    page = UpdateUserView(view, self)
if view == 'update_user_add':
    from views.update_user_add import UpdateUserAdd
    page = UpdateUserAdd(view, self)
if view == 'update_user_del':
    from views.update_user_del import UpdateUserDel
    page = UpdateUserDel(view, self)
if view == 'update_user_current':
    from views.update_user_current import UpdateUserCurrent
    page = UpdateUserCurrent(view, self)
if view == 'update_user_admin':
    from views.update_user_admin import UpdateUserAdmin
    page = UpdateUserAdmin(view, self)

st.session_state['VIEWS'][view]=page
else:
    self.VIEWS = st.session_state['VIEWS']

# 3. ----- SESSION STATE SHARED VARS THAT NEED UPDATE -----
# current view
if 'CURRENT_VIEW' not in st.session_state: # save the current state to visit it after session ends
    self.CURRENT_VIEW = st.session_state['CURRENT_VIEW']=self.VIEWS['base'] # the landing page
else:
    self.CURRENT_VIEW = st.session_state['CURRENT_VIEW']
# User status [admin, user, visitor]
if 'USER_ROLE' not in st.session_state:
    self.USER_ROLE = st.session_state['USER_ROLE'] = 'visitor'
else:
    self.USER_ROLE = st.session_state['USER_ROLE']

# User's username
if 'USER_NAME' not in st.session_state:
    self.USER_NAME = st.session_state['USER_NAME'] = ""
else:
    self.USER_NAME = st.session_state['USER_NAME']

if 'ALL_USERS' not in st.session_state:
    self.ALL_USERS = st.session_state['ALL_USERS']=[]
else:
    self.ALL_USERS = st.session_state['ALL_USERS']

# journals lists
if 'ALL_JOURNALS' not in st.session_state:
    self.ALL_JOURNALS = st.session_state['ALL_JOURNALS'] = []
else:
    self.ALL_JOURNALS = st.session_state['ALL_JOURNALS']
```

```
# journal-fos -> dict {'journal_name': ['fos1', 'fos2', ...]}
if 'JOURNAL_FOS' not in st.session_state:
    self.JOURNAL_FOS = st.session_state['JOURNAL_FOS'] = {}
else:
    self.JOURNAL_FOS = st.session_state['JOURNAL_FOS']

# fos values
if 'FOS_VALUES' not in st.session_state:
    self.FOS_VALUES = st.session_state['FOS_VALUES'] = [] # a list of the fos values
else:
    self.FOS_VALUES = st.session_state['FOS_VALUES']

# SETTERS

def apply_to_views(self):
    """
    In each session the sm will be recreated;
    always refresh the pages by passing them the updated sm object
    Always apply at the very top of the run after the session_manager has been created in each session
    The method call the refresh_session_manager() of each view at the beginning of each session
    """
    for view, view_obj in self.VIEWS.items():
        if view_obj:
            view_obj.refresh_session_manager(self)

def set_current_view(self, val):
    self.CURRENT_VIEW = st.session_state['CURRENT_VIEW']=val

# 2. Login Page
def set_user_role(self, val):
    self.USER_ROLE = st.session_state['USER_ROLE'] = val

def set_username(self, val):
    self.USER_NAME = st.session_state['USER_NAME'] = val

def set_all_users(self, val):
    self.ALL_USERS = st.session_state['ALL_USERS']=val

# 3. Updating journals, papers pages
def set_all_journals(self, val):
    self.ALL_JOURNALS = st.session_state['ALL_JOURNALS'] = val

def set_journals_fos(self, fos_dict):
    self.JOURNAL_FOS = st.session_state['JOURNAL_FOS'] = fos_dict

def set_fos_values(self, fos_list):
    self.FOS_VALUES = st.session_state['FOS_VALUES'] = fos_list
```

12. Κώδικας της αφηρημένης κλάσης View

```
import streamlit as st
import src.db_management as db

class View:
    def __init__(self, view_name, session_manager):
        """
        ABSTRACT CLASS FOR VIEWS
        -:param view_name -> the name of the view
        -:param session_manager -> the session manager object
        """
        self.name = view_name
        self.sm = session_manager
        self._id_counter=0
        self._keys=[] # self container storing the keys that have been created

    #ABSTRACT METHODS
    def render(self) :
        # abstract method; it will hold the rendering logic and render the view components"
        pass

    def _constructor(self):
        # abstract method; it will hold the attributes of the View objects"
        pass

    #PRIVATE - PROTECTED METHODS
    def _make_wdg_key(self):
        """
        return a unique id for that view
        as viewname_id (e.g. myview_07, or myview_12)
        also the new key is added to the _keys
        """
        self._id_counter+=1
        key=f'{self.name}_{str(self._id_counter-1)}'
        if self._id_counter-1<10:
            key= f'{self.name}_0{str(self._id_counter-1)}'
        self._keys.append(key)
        return key

    def _clear_keys(self):
        """ Clears all keys from the session state and empties keys"""
        for key in self._keys:
            if key in st.session_state:
                del st.session_state[key]
        self._keys=[]

    def _remove_key(self, popped):
        """ It removes a specific key from the session state and keys list"""
        new_keys=[]
        for key in self._keys:
            if key==popped:
                if key in st.session_state:
```

```
        del st.session_state[key]
    else:
        new_keys.append(key)
    self._keys=new_keys

def _sync_widget(self, widget_spec, var, sync_wdg_none=True):
    """
    Important!!! call this method at the begining of the render method
    the method links a widget's output with an attribute
    :param widget_spec a widget specifications dictionary which must contain at least a 'key' value
    :param sync_wdg_none; If True, the widget will be synchronized even its value was None;
    keep the default True especially when a view renders many subviews
    """
    if sync_wdg_none:
        if widget_spec['key'] not in st.session_state or st.session_state[widget_spec['key']]==None:
            st.session_state[widget_spec['key']] = var
    else:
        if widget_spec['key'] not in st.session_state:
            st.session_state[widget_spec['key']] = var

def _force_set_widget(self, widget_spec, var):
    """
    Important!!! call this method when force setting a value (e.g. in resets)
    the method set a widget's output to the current value
    :param widget_spec: a widget specifications dictionary which must contain
    at least a 'key' value as key !!!!!
    """
    if widget_spec['key'] not in st.session_state:
        st.session_state[widget_spec['key']] = var
    st.session_state[widget_spec['key']] = var

def _handle_db_exception(self, e):
    """ It shows warning messages to the user in case that there is an Operationl Error to the database """
    import sqlite3
    if isinstance(e, FileNotFoundError):
        st.error("Fatal Error Database file missing or corrupt")

    elif isinstance(e, sqlite3.OperationalError):
        if "database is locked" in str(e):
            st.warning("Database busy try again in few seconds")
        else:
            st.error("Internal Operation error. Try again")
    else:
        pass # if it is another error (not related to the db) it should be passed in the except section of the
        specific caller

def _update_journals(self):
    """ The method should be called each time a journal is added or deleted or sm.ALL_JOURNALS is empty
    - fetches all journals and all fos values from db,
    - sorts journals by alphabetical order
    - updates the session manager ALL_JOURNALS
```

```
- updates the session manager FOS_VALUES
- updates the session manager JOURNAL_FOS dict
'''
try:
    with db.connect_to_sqlite3(self.sm.DB) as con:
        # get all journals
        all_journals= db.get_all_journal_titles(con)
        sorted_journals = sorted(all_journals, key=lambda x: x.lower())
        self.sm.set_all_journals(sorted_journals)

        # get all fos values
        self.sm.set_fos_values(db.get_all_fos_values(con))

        # Update the journal fos dict
        j_fos_dict = {}
        for j in self.sm.ALL_JOURNALS:
            j_fos_dict[j] = db.get_journal_fos_tuple(con, j)[1]
        self.sm.set_journals_fos(j_fos_dict)

except Exception as e:
    self._handle_db_exception(e)

# PUBLIC METHODS
def refresh_session_manager(self, new_sm):
    '''
    Important this method is the key to the app.
    It should be called for each view at the beginning of the app when we have a new session
    So, in each session the view will point to the new session manager; otherwise the view will point to
    the old session manager
    Override this method for Views that contain subviews!!!,
    the sub_views must also point to the new session manager!!!
    '''
    self.sm = new_sm

def reset(self):
    # The method sets a View object to the default state when created
    self._clear_keys()
    self._constructor()
    print(f'Reset called from the abstract class for {self.name}')
```

13. Κώδικας της κλάσης Suggest_View (View)

```
import streamlit as st
import src.db_managment as db
import src.app_utils as util
from views._abstract_view import View

class SuggestView(View):
    def __init__(self, class_name, session_manager):
        super().__init__(class_name, session_manager)
        self._constructor()

    def _constructor(self):
        # ATTRIBUTES
        self._paper_text = None # the text of the paper
        self._paper_file = None # the uploaded file of the paper
        self._results_df=util.get_pandas_df([],[]) # the df to be rendered
        self._current_pagination=1 # The current page showed in pagination results

        #STATES
        self._show_file_uploader=True
        self._show_calculate_results=False # Results will be calculated once
        self._show_results=False

        # Widgets
        self._file_uploader = {"key": self._make_wdg_key(),
                                'label': "Upload the paper's file (supported types .pdf, .docx, .txt)",
                                'type': ["pdf", "docx", "txt"]}

        self._btn_reset = {"key": self._make_wdg_key(), 'label': 'Make a new search'}

    # app - methods
    def _get_similar_journals(self):
        """ It updates the self._results_df as df[journal_name, score, fos]"""

        ALGORITHM = self.sm.CONFIG['ALGORITHM']
        STEMMER = self.sm.CONFIG['STEMMER']
        N_GRAMS = self.sm.CONFIG['N_GRAMS']
        STOP_WORDS = self.sm.CONFIG['STOP_WORDS']
        KEYBERT_MODEL = None
        try:
            KEYBERT_MODEL = self.sm.KB_MODEL
        except:
            pass
        try:
            with db.connect_to_sqlite3(self.sm.DB) as con:
                try:
                    results= db.get_similar_papers_db(con,
                                                        query_text=self._paper_text,
                                                        algorithm_name= ALGORITHM,
```

```
n_grams=N_GRAMS,
stop_words=STOP_WORDS,
stemmer=STEMMER,
keybert_model=KEYBERT_MODEL,
top_n= None) #None means all query

# remove the paper entries from the results
results = [(rec[0], rec[2]) for rec in results]

# filter the results to get unique journals
journal_list = util.get_single_records (tuple_list=results, filter_id=0,
                                       returned_size=self.sm.CONFIG["MAX_FETCHED_JOURNALS"])
# Append the journal Fos to each journal in the list
journal_list = [(row[0], row[1], ', '.join(self.sm.JOURNAL_FOS[row[0]])) for row in journal_list ]
# Update the self.results_df
self._results_df = util.get_pandas_df(journal_list, columns=['Journal', 'Similarity', 'FOS'])

except Exception as e:
    self._handle_db_exception(e)

except Exception as e:
    self._handle_db_exception(e)

def _render_read_paper_text(self):
    st.markdown('### Upload your manuscript (upload only .pdf, .docx, .txt)')
    self._paper_file = st.file_uploader(**self._file_uploader)
    if self._paper_file:
        try:
            self._paper_text = util.read_file(file_path=self._paper_file,
                                              read_lowercase=False,
                                              as_para_list=False, encoding='utf-8',
                                              read_as_bytes=True)
            self._show_calculate_results=True

        except:
            st.error('Unsupported file format; upload only .pdf, .docx, .txt')
            self._paper_text = None

            if self._paper_text != None and self._paper_text == "":
                st.error('You provided an empty file')
                self._paper_text = None
                self._show_calculate_results = False
            else:
                self._paper_file = None

def _render_calculation(self):
    with st.spinner("Finding similar journals..."):
        self._get_similar_journals()
        if not self._results_df.empty:
            self._show_file_uploader = False
            self._show_calculate_results = False
```

```
self._show_results = True
st.rerun()
else:
    st.markdown(f'### Your manuscript did not return any matching journal!')

def _render_results(self):
    # Some custom css
    st.markdown(f'''<style>
        .stMainBlockContainer {{ max-width:90vw;}}
    </style>''',
        unsafe_allow_html=True)
    st.markdown('#### Find below the suggested journals to publish your manuscript:')
    st.write()
    df_slice = self._results_df.copy() # That value holds the rendered df

    # Paginator functionality (if required)
    page_size = self.sm.CONFIG['JOURNALS_PER_PAGE']
    if len(self._results_df) > page_size:
        total_pages = (len(self._results_df) - 1) // page_size + 1
        page = self._current_pagination # the current page of the paginator
        col1, col2, col3 = st.columns([1, 6, 1])
        with col1:
            if st.button("← Prev", key="__show_results__page__prev") and page > 1:
                self._current_pagination -= 1
                st.rerun()

        with col2:
            page = self._current_pagination
            start = (page - 1) * page_size
            end = start + page_size
            st.markdown(f'<div style="text-align:center;">Page {page} / {total_pages}</div>',
                unsafe_allow_html=True)

        with col3:
            if st.button("Next →", key="__show_results__page__next") and page < total_pages:
                self._current_pagination += 1
                st.rerun()

    # Make a copy of the DataFrame slice
    df_slice = self._results_df.iloc[start:end].copy()

    # Rendering the dataframe (paginated or not)
    st.dataframe(
        df_slice,
        column_config={
            "last_column": st.column_config.TextColumn(
                "Last Column",
                width="large"
            )
        }
    )
)
```

```
# Rendering reset button
st.divider()
self._btn_reset['on_click'] = self.reset
st.button(**self._btn_reset)

def render(self):
    if not self.sm.JOURNAL_FOS: # First Load
        # Call the parent method to update sm.ALL_JOURNALS, sm.ALL_JOURNAL_FOS, sm.FOS_VALUES
        self._update_journals()

    if self.sm.JOURNAL_FOS: # Subsequent loads
        if self._show_file_uploader: # Upload the file and read its text; advance to calculations
            self._render_read_paper_text()
        if self._show_calculate_results: # Make calculations and fetch results; advance to show results
            self._render_calculation()
        if self._show_results: # Show results to the user
            self._render_results()
    else:
        st.error('The data base is empty. Login as user or admin to add papers to the database')
```

14. Βοηθητικές μέθοδοι από src.db_managment που χρησιμοποιήθηκαν στην Suggest_View

```
def connect_to_sqlite3(db_path, timeout=30):  
    """ Connect to database and return the connection object.  
    :param timeout secs to wait for a busy db to execute a statement  
    It raises exception if path doesn't exist or db is busy for more than timeout seconds  
    """  
    import os.path  
    import sqlite3  
    if not os.path.exists(db_path):  
        raise FileNotFoundError(f'Connection to database "{db_path}" can not be established')  
  
    conn = sqlite3.connect(db_path, timeout=timeout) # wait for timeout secs if too busy  
    conn.execute("PRAGMA foreign_keys = 1") # Enable foreign key checks and cascade functionality  
    conn.execute("PRAGMA journal_mode = WAL;") # Enable simultaneous reading and writing to db  
    return conn  
  
def __calculate_normalized_tf_idf(tokens_freq_list:list, tokens_df:dict, docs_number):  
    """  
    The method returns a list of tuples of the tokens with their normalized tf-idf scores  
    -params: tokens_freq_list: The document we want to calculate tf-idf as a list of tuples (token, freq)  
            tokens_df: A dictionary of the form {token:corpus_frequency}  
            N: total number of documents  
    """  
    # Create a dict of the document:  
    document= {t[0]:t[1] for t in tokens_freq_list}  
  
    # Update document frequencies as freq *= log (N/token_df); exceptions give freq=0  
    for token in document:  
        try:  
            document[token] *= math.log10(docs_number/tokens_df[token])  
        except:  
            document[token] =0  
  
    #Calculate the squared sum of the paper  
    ssum = 0  
    for token in document:  
        ssum+= document[token] * document [token]  
    ssum = math.sqrt(ssum)  
  
    # Normalize the scores (watch for totally irrelevant documents -> all token scores are 0)  
    if ssum!=0:  
        for token in document:  
            document[token]/=ssum  
  
    return document  
  
def _get_similar_papers_tf_idf_db(con, query_text,n_grams, stop_words, stemmer, top_n=None):  
    """
```

```
Use this method ONLY for TF-IDF
It returns a list of tuples (paper_id, cosine_score) of the most relevant papers (not sorted)
'''
cursor = con.cursor()

# 1. Get the fgrequencies list of the query -> normalized query score list gives the frequencies for tf-idf
query_score_list = _get_algo_normalized_list(query_text, 'tf-idf', n_grams, stop_words, stemmer,
                                             keybert_model=None, top_n=top_n)

if len(query_score_list) > 0:
    # 1. Find the id of the algorithm
    algorithm = cursor.execute('SELECT id FROM ALGORITHM WHERE name=?', ['tf-idf']).fetchall()
    algorithm_id = algorithm[0][0]

    # 2. Calculate the N (total Number of papers = corpus) for tf-idf
    cursor.execute("""
        SELECT COUNT(DISTINCT paper_id)
        FROM PAPER_SCORES
        WHERE algo_id = ?
    """, (algorithm_id,))
    N = cursor.fetchone()[0]

    # 3. For each token calculate the document frequency (how many times exists in the corpus)
    rows= cursor.execute("""
        -- Return how many papers contain the token
        SELECT t.text, COUNT(DISTINCT ps.paper_id) -- token text instead of id to map with the query
        FROM PAPER_SCORES ps
        JOIN TOKEN t ON t.id = ps.token_id -- join the token fk to get the text
        WHERE ps.algo_id=?
        GROUP BY ps.token_id
    """, (algorithm_id,))
    all_token_rows= rows.fetchall() # a list of tuples (text, df)
    all_token_df_dict=dict(all_token_rows)

    # 4. Calculate for each paper each token's score and the sum of squares => make scores normalized

    # Get the ids of the papers and chunk them;
    paper_ids = cursor.execute ("SELECT id from PAPER").fetchall()
    paper_chunks = create_chunks(paper_ids, 800)

    #Create the papers dict so to feed the tf-idf calculation method
    papers_dict={} # Initially it will store the paper_scores {paper_id: [(token, frequency)]
                  #next it will store as {paper_id: {token, normalized_score}}
    for chunk in paper_chunks:
        placeholders=', '.join(['?']*len(chunk))
        score_rows = cursor.execute(f"""
            SELECT ps.paper_id, t.text, ps.score
            FROM PAPER_SCORES ps
            JOIN TOKEN t ON t.id = ps.token_id -- join the token fk to get the text
            WHERE ps.algo_id = {algorithm_id}
            AND ps.paper_id IN ({placeholders})
        """, [x[0] for x in chunk]).fetchall()
```

```
for r in score_rows: # r[0]-> paper_id, r[1] -> token, r[2] -> algorithm's score
    indx = papers_dict.get(r[0])
    if indx is None:
        papers_dict[r[0]]=[(r[1], r[2])] # => paper_dicts {paper_id=[(token, score),...]}
    else:
        papers_dict[r[0]].append((r[1], r[2]))

# For each paper make papers_dict in the form {paper_id:{token: normalized_tf_ids}}
for p_id in papers_dict:
    papers_dict[p_id]=__calculate_normalized_tf_idf(papers_dict[p_id], all_token_df_dict, N)

# 5. Calculate the sum of squares => make scores normalized
normalized_query = __calculate_normalized_tf_idf(query_score_list, all_token_df_dict, N)

# 6. Calculate cosine similarity for each paper in the db and store in a list of tuples (paper_id, cosine)
papers_cosine_list=[]
for p_id in papers_dict:
    cosine=0
    for token in papers_dict[p_id]:
        indx=normalized_query.get(token)
        if indx is not None: # token of the paper exists also to the query
            cosine+=papers_dict[p_id][token] * normalized_query[token]
    if cosine>0:
        papers_cosine_list.append((p_id, cosine))

# 7. return as sorted list of tuples
return papers_cosine_list.sort(key=lambda x: x[1], reverse=True)

def _get_similar_papers_no_tf_idf_db(con, query_text, algorithm_name, n_grams, stop_words,
                                    stemmer, keybert_model, top_n=None):
    """
    Use this method for all algorithms EXCEPT for TF-IDF
    It returns a list of tuples (paper_id, cosine_score) of the most relevant papers (not sorted)
    """
    cursor = con.cursor()

    # 1. Get the normalized query score list
    query_score_list = _get_algo_normalized_list(query_text, algorithm_name, n_grams, stop_words,
                                                stemmer, keybert_model, top_n)

    if len(query_score_list)>0:
        #1. Find the id of the algorithm
        algorithm = cursor.execute('SELECT id FROM ALGORITHM WHERE name=?',
                                   [algorithm_name.strip()]).fetchall()
        algorithm_id = algorithm[0][0]

        #2. Calculate the cosine similarity
        # a. create the collections (dictionaries)
        query_scores_dict= dict(query_score_list) # Create a dict for the query tokens {token:score}
```

```

db_scores_dict={} # Create the dict for the database papers {'paper_id': 'cosine_score'}

# b. ask the db to fetch the algorithm x paper x token combinations for each token of the query
query_score_list_chunks = create_chunks(query_score_list, chunk_size=900) # chunk the query
for chunk in query_score_list_chunks:
    placeholders=', '.join(['?']*len(chunk))
    sql_prep=f"""
    SELECT t.text, ps.paper_id, ps.score
    FROM PAPER_SCORES ps
    JOIN TOKEN t ON t.id = ps.token_id
    WHERE ps.algo_id = {algorithm_id}
    AND t.text IN ({placeholders});
    """
    items = cursor.execute(sql_prep, [t[0] for t in chunk]).fetchall()

# c. Calculate cosine similarity for all papers
for r in items: # r[0] -> token_txt, r[1] -> paper_id, r[2] -> algorithm's score
    indx = db_scores_dict.get(r[1])
    if indx is None: # paper do not exist
        # add it as paper_token_score * query_token_score
        db_scores_dict[r[1]] = r[2] * query_scores_dict[r[0]]
    else:
        # increase by paper_token_score * query_token_score
        db_scores_dict[r[1]] += r[2] * query_scores_dict[r[0]]

#3. Return the papers as list of tuples
# Create a list of tuples [(p_id, cosine_sim)] and sort descending by cosine_sim
papers_list = [(paper, db_scores_dict[paper]) for paper in db_scores_dict]
return papers_list.sort(key=lambda x: x[1], reverse=True)
return []

def get_similar_papers_db(con, query_text, algorithm_name, n_grams, stop_words, stemmer,
keybert_model, top_n=None):
    """
    It returns a list of tuples (journal_name, paper_name, cosine_score) of the most relevant papers
    sorted by cosine_score
    :param top_n it applies to the query (returns the top_n tokens); default=None => all query terms
    """
    paper_id_tuples=None # The list of fetched paper ids of the form (paper_id, score)
    if algorithm_name.strip().lower() == 'tf-idf':
        paper_id_tuples = _get_similar_papers_tf_idf_db(con, query_text, n_grams, stop_words,
                                                         stemmer, top_n)
    else:
        paper_id_tuples = _get_similar_papers_no_tf_idf_db(con, query_text, algorithm_name,
                                                            n_grams, stop_words, stemmer,
                                                            keybert_model, top_n)

    if len(paper_id_tuples)>0:
        similar_papers = [] # the final list
        paper_scores = {t[0]:{'score':t[1]} for t in paper_id_tuples} # a dict {paper_id:score}
        paper_id_chunks = create_chunks(paper_id_tuples, chunk_size=900) # chunk the query
        for chunk in paper_id_chunks:

```

```
placeholders = ','.join(['?'] * len(chunk))
sql_prep = f"""
    SELECT p.id, p.title, j.name
    FROM PAPER p
    JOIN JOURNAL j ON j.id = p.journal_id
    WHERE p.id IN ({placeholders});
    """

# get a list of tuples (id, paper_name, journal_name)
items = con.cursor().execute(sql_prep, [t[0] for t in chunk]).fetchall()

for item in items:
    try:
        score = paper_scores[item[0]]['score']
        similar_papers.append((item[2], item[1], score))
    except:
        print('Exception for creating the list for (journal, paper, score)')
similar_papers.sort(key=lambda x: x[2], reverse=True) # sort by score
return similar_papers

return[]
```

15. Βοηθητικές μέθοδοι από src.app_utils που χρησιμοποιήθηκαν στην Suggest_View

```
import pandas as pd
from docx import * # pip package name python-docx
import fitz # pip package PyMuPDF - reading pdf files

def load_docx(file_path, read_lowercase, as_para_list):
    """ It reads a .docx file and it returns the text
        :params: read_lowercase
        :params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
        works the same if the file_path is a local file or bytes (via html)
    """
    text = Document(file_path)
    sentence = "" # Paragraph sentence
    doc=[]
    for p in text.paragraphs:
        if read_lowercase:
            sentence += p.text.lower()
        else:
            sentence += p.text
        doc.append(sentence)
        sentence=""

    if as_para_list:
        return doc
    return "\n".join(doc)

def load_txt(file_path, read_lowercase, as_para_list, encoding='utf-8'):
    """ It reads a .txt file from disk and it returns the text
        :params: read_lowercase
        :params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
    """
    doc=[] # the final doc will be a list of paragraphs; each paragraph will be a single string

    # read, write etc.(represented by r, w, r+, w+, rb, wb etc.)
    with open(file_path, "r", encoding=encoding) as file:
        txt_lines =file.readlines() # A list of lines that were read
        txt_lines[len(txt_lines)-1]+='\n'

    for line in txt_lines:
        if line !='\n':
            if read_lowercase:
                doc.append(line.strip().lower())
            else:
                doc.append(line.strip())
    #add a final '.' at the end
    if not doc[len(doc)-1].endswith('.'):
        doc+='.'
    if as_para_list:
```

```
    return doc
return "\n".join(doc)

def load_txt_bytes(file_path, read_lowercase, as_para_list, encoding='utf-8'):
    """ It reads a .txt file as stream of bytes and it returns the text
        :params: read_lowercase
        :params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
        Use it for reading from html, streamlit etc
    """
    file= file_path.read().decode(encoding, errors="ignore")
    doc=[] # the final doc will be a list of paragraphs; each paragraph will be a single string

    txt_lines =file.splitlines() # A list of lines that were read
    txt_lines[len(txt_lines)-1]+='\n'

    for line in txt_lines:
        if line !='\n':
            if read_lowercase:
                doc.append(line.strip().lower())
            else:
                doc.append(line.strip())
    #add a final '.' at the end
    if not doc[len(doc)-1].endswith('.'):
        doc+='.'

    if as_para_list:
        return doc
    return "\n".join(doc)

def load_pdf(file_path, read_lowercase, as_para_list):
    """ It reads a .pdf file from disk and it returns the text
        :params: read_lowercase
        :params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
    """
    text = ""
    with fitz.open(file_path) as doc:
        for page in doc:
            text += ' ' + page.get_text()
    # Split by double new lines because it is the most common paragraph spacing in pdf files
    # .strip() removes left and right spaces
    if read_lowercase:
        para_list = [para.strip().lower() for para in text.split('\n \n') if para.strip()]
    else:
        para_list = [para.strip() for para in text.split('\n \n') if para.strip()]

    if as_para_list:
        return para_list
    return "\n".join(para_list)

def load_pdf_bytes(file_path, read_lowercase, as_para_list):
```

```
""" It reads a .pdf file as stream of bytes and it returns the text
:params: read_lowercase
:params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
Use it for reading from html, streamlit etc
"""

doc = fitz.open(stream=file_path.read(), filetype="pdf")
text = ""

for page in doc:
    text += ' ' + page.get_text()
# Split by double new lines because it is the most common paragraph spacing in pdf files
# .strip() removes left and right spaces
if read_lowercase:
    para_list = [para.strip().lower() for para in text.split('\n\n') if para.strip()]
else:
    para_list = [para.strip() for para in text.split('\n\n') if para.strip()]

if as_para_list:
    return para_list
return "\n".join(para_list)

# Default reading is from local file
def read_file(file_path, read_lowercase, as_para_list, encoding='utf-8', read_as_bytes=False):
    """ It reads a .pdf, .docx .txt file and it returns the text
    :params: read_lowercase
    :params: as_para_list = True It returns a list (paragraphs) with text of each paragraph
    :params: read_as_bytes set it to True to read as bytes stream (html, streamlit etc)
    It throws exception for unsupported file format
    """

    if read_as_bytes: # it is a file (not path)
        filename = file_path.name.lower()

        if filename.endswith(".txt"):
            return load_txt_bytes(file_path, read_lowercase, as_para_list, encoding)

        elif filename.endswith(".pdf"):
            return load_pdf_bytes(file_path, read_lowercase, as_para_list)

        elif filename.endswith(".docx"):
            return load_docx(file_path, read_lowercase, as_para_list)

        else:
            raise Exception('Unsupported file extension; method reads only .docx, pdf and .txt')

    else: # Read from file path; read_as_bytes = False
        if str(file_path).endswith('.txt'):
            return load_txt(file_path, read_lowercase, as_para_list, encoding)

        elif str(file_path).endswith('.pdf'):
            return load_pdf(file_path, read_lowercase, as_para_list)

        elif str(file_path).endswith('.docx'): # docx is the same no matter it is bytes or local file
```

```
    return load_docx(file_path, read_lowercase, as_para_list)

    else:
        raise Exception('Unsupported file extension; method reads only .docx, pdf and .txt')

def get_pandas_df(data, columns, drop_pd_idx=True):
    """ When a df is created pandas creates an internal index;
        if we do not want that index we set the parameter drop_pd_idx=True"""
    if drop_pd_idx:
        return pd.DataFrame(data, columns=columns).reset_index(drop=drop_pd_idx)
    return pd.DataFrame(data, columns=columns)

def get_single_records(tuple_list, filter_id, returned_size=None):
    """
        The method receives a list of tuples and returns a filtered version
        where filter_id items will be presented only once
        The method also returns a specific size of the filtered list
        :param tuple_list the list of tuples
        :param filter_id, the filtering position of unique items
        :param returned_size, the length of the returned list
    """
    end=returned_size
    if returned_size==None:
        end=len(tuple_list)
    added_ids=[] # It will hold the added unique values
    return_list=[]
    for rec in tuple_list:
        if rec[filter_id] not in added_ids:
            return_list.append(rec)
            added_ids.append(rec[filter_id])
            if len(added_ids)>=end:
                break
    return return_list
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και πληρούν τους κανόνες της επιστημονικής παράθεσης.