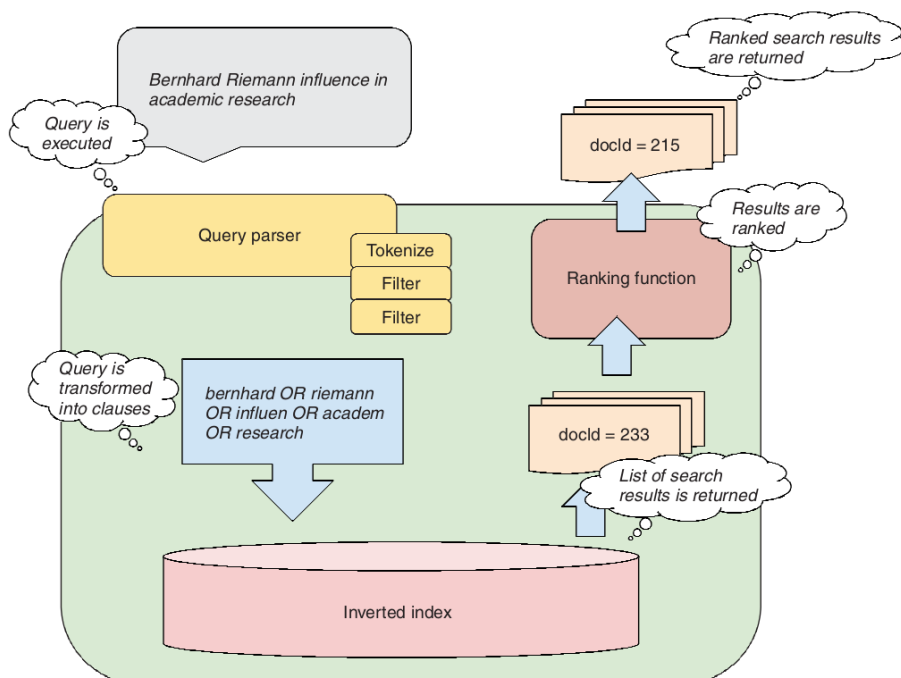


## Προγραμματιστική Εργασία: Σταθμίζοντας τα Αποτελέσματα της Ανάκτησης χρησιμοποιώντας Ενσωματώσεις Λέξεων (Word Embeddings)

Ο σκοπός της εργασίας είναι να εξασκηθείτε σε κλασικές μεθόδους και μοντέλα ανάκτησης πληροφορίας, αλλά και να εφαρμόσετε state-of-the-art τεχνικές για να βελτιώσετε τα αποτελέσματα ενός συστήματος ανάκτησης πάνω σε πραγματικά δεδομένα.

Ένα μοντέλο ανάκτησης είναι ένας μαθηματικός φορμαλισμός με βάση τον οποίο πραγματοποιείται η λειτουργία της ανάκτησης. Ένα σύστημα ανάκτησης, όπως για παράδειγμα η δημοφιλής μηχανή αναζήτησης Google, υιοθετεί ένα συγκεκριμένο μοντέλο ανάκτησης και το υλοποιεί με βάση την αντίστοιχη μαθηματική θεωρία. Ένα παράδειγμα ενός συστήματος ανάκτησης παρουσιάζεται στην Εικόνα 1. Ο χρήστης αρχικά υποβάλλει ένα ερώτημα στο σύστημα. Το ερώτημα του χρήστη αναλύεται σε ένα σύνολο όρων και εκτελείται στη δομή δεδομένων του συστήματος ανάκτησης (για κάθε όρο πραγματοποιείται μια αναζήτηση σε μία δομή ανεστραμμένου ευρετηρίου όπως θα δούμε στο μάθημα). Τα κείμενα της συλλογής που ταιριάζουν συλλέγονται και διαβιβάζονται στη συνάρτηση στάθμισης/ομοιότητας (*ranking function*), η οποία υπολογίζει ένα σκορ ομοιότητας κάθε κειμένου με το ερώτημα του χρήστη. Η συνάρτηση ομοιότητας αποτελεί την καρδιά ενός μοντέλου ανάκτησης και ουσιαστικά είναι η μαθηματική θεωρία του. Στη συνέχεια, τα κείμενα ταξινομούνται/κατατάσσονται με βάση το σκορ τους σε φθίνουσα σειρά, δηλαδή από αυτό με το μεγαλύτερο σκορ ομοιότητας, που θεωρείται το πιο σχετικό, προς αυτό με το μικρότερο σκορ. Και επιστρέφονται στο χρήστη για παράδειγμα τα πρώτα 10 κείμενα. Η διαδικασία της ταξινόμησης ονομάζεται *στάθμιση (ranking)* ή *βαθμολόγηση (scoring)*.



Εικόνα 1: Υποβολή ερωτήματος, ανάκτηση και στάθμιση συναφών κειμένων.

Στη διάρκεια του μαθήματος θα γνωρίσουμε κλασικά μοντέλα ανάκτησης, καθένα από τα οποία ορίζει ένα διαφορετικό τρόπο (μαθηματικό φορμαλισμό όπως είπαμε) με τον οποίο υπολογίζει την ομοιότητα των κειμένων με το ερώτημα του χρήστη, βασίζεται επομένως σε διαφορετική *συνάρτηση ομοιότητας*. Για παράδειγμα, θα δούμε στατιστικά μοντέλα, όπως το *μοντέλο διανυσματικού χώρου (Vector Space Model – VSM)*, τα οποία μετρούν την ομοιότητα ανάμεσα στο ερώτημα του χρήστη και στα κείμενα υπολογίζοντας στατιστικά για τους κοινούς όρους τους. Η απόφαση για τη στάθμιση ενός κειμένου βασίζεται στο πόσο συχνά ένας όρος εμφανίζεται σε ένα κείμενο ή/και σε όλη τη συλλογή. Επίσης, θα δούμε μοντέλα που υπολογίζουν την ομοιότητα με βάση τη θεωρία πιθανοτήτων, όπως το *γλωσσικό μοντέλο (Language Model – LM)* και το *πιθανοτικό μοντέλο (Okapi BM25)*. Στην περίπτωση αυτή, το σύστημα ανάκτησης σταθμίζει ένα κείμενο με βάση την πιθανότητα να είναι συναφές με το ερώτημα.

Τα κλασικά μοντέλα ανάκτησης «αντιλαμβάνονται» την ομοιότητα κειμένων και ερωτημάτων ως μια διαδικασία *λεξικής σύγκρισης* (term matching). Η απλοποιημένη ιδέα είναι ότι ένα κείμενο θεωρείται συναφές με ένα ερώτημα εφόσον περιέχει ακριβώς τους όρους του ερωτήματος. Η ιδέα αυτή είναι περιοριστική. Σε έναν ιδανικό κόσμο ένα σύστημα ανάκτησης θα προσπαθούσε να κατανοήσει την *πληροφοριακή ανάγκη* του χρήστη και να απαντήσει ανάλογα. Για παράδειγμα, θα μπορούσε να καταλάβει ότι όταν ο χρήστης υποβάλλει ένα ερώτημα “laptop” είναι σωστό και θεμιτό να του επιστρέψει ως συναφή, κείμενα που περιέχουν τη λέξη laptop, αλλά και κείμενα που περιέχουν τη λέξη notebook και όχι αναγκαστικά τη λέξη laptop. Κι αυτό γιατί οι λέξεις laptop και notebook είναι *σημασιολογικά* παρόμοιες, έχουν παρόμοια σημασία. Ένα τέτοιο σύστημα ανάκτησης προσπαθεί να καταλάβει τη σημασιολογία των λέξεων και να αποφασίσει αν ένα κείμενο είναι συναφές με ένα ερώτημα ή όχι.

Η δημιουργία ενός τέτοιου συστήματος ανάκτησης είναι δύσκολη. Όμως, όπως θα δείτε, η εφαρμογή τεχνικών βασισμένων σε νευρωνικά δίκτυα μπορεί να γεφυρώσει το χάσμα ανάμεσα στο ερώτημα του χρήστη και στην πραγματική πληροφοριακή του ανάγκη. Οι τεχνικές αυτές δημιουργούν νέες, σημασιολογικά πλούσιες αναπαραστάσεις των λέξεων, των κειμένων και των ερωτημάτων, ξεπερνώντας την απλή λεξική σύγκριση.

Επομένως, ένα καλό μοντέλο ανάκτησης πρέπει να λαμβάνει υπόψη του τη σημασιολογία. Όπως ίσως θα φαντάζεστε, αυτό επηρεάζει και τον τρόπο με τον οποίο σταθμίζονται τα κείμενα. Για παράδειγμα:

- Όταν θέλουμε να υπολογίσουμε το σκορ ομοιότητας ενός κειμένου που θεωρήθηκε συναφές γιατί περιείχε έναν σημασιολογικά παρόμοιο όρο με το ερώτημα, πρέπει το κείμενο αυτό να βαθμολογηθεί διαφορετικά απ’ ότι τα κείμενα που περιέχουν ακριβώς τον όρο του ερωτήματος;
- Αν δημιουργήσουμε αναπαραστάσεις από νευρωνικά δίκτυα (πχ. word2vec αναπαραστάσεις των λέξεων) που θα καταλαβαίνουν την πληροφοριακή ανάγκη του χρήστη, πώς θα τις χρησιμοποιήσουμε για να ανακτήσουμε κείμενα και να σταθμίσουμε τα αποτελέσματα της ανάκτησης;


Στη συνέχεια, θα χρησιμοποιήσουμε νευρωνικά δίκτυα για να δούμε πώς αυτά μπορούν να βελτιώσουν τα κλασικά μοντέλα ανάκτησης ή να δημιουργήσουν νέα (και καλύτερα) μοντέλα ανάκτησης.

Στην εργασία αυτή, καλείστε να δημιουργήσετε ένα σύστημα ανάκτησης που θα εφαρμόζει κλασικά μοντέλα και μεθόδους ανάκτησης καθώς και μοντέλα ανάκτησης που βασίζονται σε νευρωνικά δίκτυα και συνδυασμούς αυτών, για να λύσετε ένα συγκεκριμένο πρόβλημα ανάκτησης πάνω σε μία συλλογή δεδομένων. Για το σκοπό αυτό θα χρησιμοποιήσετε τη Lucene και τη DeepLearningForJava (DL4J)

στη συλλογή κειμένων IR2024, καθώς και το εργαλείο αξιολόγησης `trec_eval`. Η Lucene είναι μία open source βιβλιοθήκη που παρέχει εργαλεία για τη δημιουργία μίας μηχανής αναζήτησης. Η DeepLearningForJava είναι μια βιβλιοθήκη που παρέχει υλοποιημένα μοντέλα νευρωνικών δικτύων, όπως το `word2vec`, για τη java. ~~Οι διαθέσιμες συλλογές κειμένων βρίσκονται στο τέλος του εγγράφου και θα επιλέξετε μία σύμφωνα με τον αριθμό μητρώου σας. Κάθε συλλογή περιλαμβάνει ένα σύνολο από κείμενα και ερωτήματα μαζί με τις σωστές συναφείς απαντήσεις.~~

## Φάση 1 – Baseline – Μοντέλο Ανάκτησης Διανυσματικού Χώρου

Στην πρώτη φάση της εργασίας θα εφαρμόσετε το μοντέλο διανυσματικού χώρου στη συλλογή κειμένων IR2024. Η IR2024 είναι μία συλλογή από 18.316 κείμενα. Περιλαμβάνει ένα σύνολο από ερωτήματα μαζί με τις σωστές συναφείς απαντήσεις. Το σύστημά σας απλά θα επιστρέφει τα πιο σχετικά κείμενα σε κάθε ερώτημα.

1. Προεπεξεργαστείτε τη συλλογή κειμένων IR2024 (αρχείο `documents.txt`) προκειμένου να είναι σε κατάλληλη μορφή για να χρησιμοποιηθεί από τη μηχανή αναζήτησης Lucene.
2. Δημιουργήστε ένα ευρετήριο από τη συλλογή χρησιμοποιώντας τη μηχανή αναζήτησης Lucene. Επιλέξτε κατάλληλο Analyzer και συνάρτηση ομοιότητας των `ClassicSimilarity`. Κάθε κείμενο θα πρέπει να αποθηκευτεί σε ένα `field` της Lucene. 
3. Εκτελέστε τα ερωτήματα (αρχείο `queries.txt`) πάνω στο ευρετήριο και συλλέξτε τις απαντήσεις της μηχανής, τα  $k$  πρώτα ανακτηθέντα κείμενα, για  $k = 50$ .
4. Αξιολογήστε τις απαντήσεις σας συγκρίνοντάς τις με τις σωστές απαντήσεις (αρχείο `qrels.txt`) χρησιμοποιώντας το εργαλείο αξιολόγησης `trec_eval` και τα μέτρα αξιολόγησης MAP (mean average precision) και `Precision@k` (ακρίβεια στα  $k$  πρώτα ανακτηθέντα κείμενα) για  $k = 5, 10, 15, 20$ .
5. Καταγράψτε τα πειράματά σας σε μια αναφορά. Περιγράψτε πώς υλοποιήσατε τα 4 παραπάνω βήματα, συμπεριλάβετε screenshots όπου θεωρείτε χρήσιμο (εγκατάσταση εργαλείων, εκτέλεση κώδικα, εκτέλεση `trec_eval`), και φτιάξτε έναν πίνακα με τα αποτελέσματα του `trec_eval` για τις διάφορες τιμές του  $k$ . Συζητήστε τα αποτελέσματά σας. Δημιουργήστε ένα αρχείο pdf με την αναφορά σας. Θα υποβάλετε την αναφορά σας, τον κώδικά σας και τα αποτελέσματα του `trec_eval` σε ένα αρχείο zip με ονομασία `αριθμός_μητρώου1_αριθμός_μητρώου2.zip` (π.χ. `3200100_3200200.zip`). Μη συμπεριλάβετε τη συλλογή κειμένων. Καταγράψτε τις πηγές σας.

## Φάση 2 – Πιθανοτικό και Γλωσσικό Μοντέλο Ανάκτησης

Επαναλάβετε τη Φάση 1 αλλάζοντας κατάλληλα τη συνάρτηση ομοιότητας έτσι ώστε να εφαρμόσετε το πιθανοτικό και το γλωσσικό μοντέλο ανάκτησης για να λύσετε το πρόβλημα της ανάκτησης. Οι συναρτήσεις ομοιότητας που θα χρησιμοποιήσετε είναι οι εξής: `BM25Similarity` και `LMJelinekMercerSimilarity`. Θα τις εξετάσουμε όταν θα δούμε τα αντίστοιχα μοντέλα ανάκτησης που θα παρουσιαστούν στο μάθημα.

### Φάση 3 - Ανάκτηση χρησιμοποιώντας Ενσωματώσεις Λέξεων (Word Embeddings)

Στη φάση αυτή θα χτίσετε ένα σύστημα ανάκτησης, στο οποίο οι αναπαραστάσεις των κειμένων και των ερωτημάτων θα έχουν δημιουργηθεί με το word2vec. Το word2vec<sup>1,2</sup> είναι ένας αλγόριθμος που χρησιμοποιεί νευρωνικά δίκτυα πρόσθιας τροφοδότησης για τη μάθηση διανυσματικών αναπαραστάσεων των λέξεων (γνωστές ως ενσωματώσεις λέξεων), που μπορούν να χρησιμοποιηθούν για την εύρεση λέξεων με παρόμοια σημασία ή λέξεων που εμφανίζονται σε παρόμοια περιβάλλοντα (contexts). Το μοντέλο εκτιμά την πιθανότητα να επιλεγεί μία λέξη (output) με βάση το περιβάλλον της (input). Εξάγει τους κοντινότερους γείτονες μιας λέξης εξετάζοντας τα συμφραζόμενα, το περιβάλλον της λέξης, και καθορίζει πότε δύο λέξεις είναι σημασιολογικά συναφείς (όταν εμφανίζονται σε ίδιο ή παρόμοιο περιβάλλον-context). Υπό το πρίσμα αυτό μπορούμε να χρησιμοποιήσουμε το μοντέλο για να ανακαλύψουμε λέξεις που παρουσιάζουν παρόμοια σημασιολογία. Τα κείμενα που αποτελούνται από λέξεις παρόμοιας σημασιολογίας με τις λέξεις του ερωτήματος του χρήστη μπορούν να θεωρηθούν συναφή με το ερώτημα και να ανακτηθούν. Για παράδειγμα, αν ο χρήστης υποβάλει το ερώτημα "hiking", και υπάρχουν κείμενα που περιέχουν τη λέξη trekking, τα κείμενα αυτά θα επιστραφούν στο χρήστη, παρόλο που δεν περιέχουν τον όρο hiking, επειδή το word2vec θα έχει θεωρήσει τους δύο όρους ως σημασιολογικά παρόμοιους εφόσον και οι δύο εμφανίζονται σε παρόμοιο περιβάλλον.

Ο στόχος μας είναι να σταθμίσουμε τα κείμενα για ένα συγκεκριμένο ερώτημα, όμως το word2vec αναπαριστά τους όρους των κειμένων ως διανύσματα και όχι ακολουθίες όρων, τα κείμενα δηλαδή, ως διανύσματα. Επομένως, το πρώτο πράγμα που θα πρέπει να κάνετε είναι να βρείτε έναν τρόπο να χρησιμοποιήσετε τα διανύσματα των όρων για να αναπαραστήσετε κείμενα και ερωτήματα. Ένας απλός τρόπος να δημιουργηθούν διανύσματα κειμένων από διανύσματα λέξεων είναι με τον υπολογισμό του μέσου όρου των διανυσμάτων των λέξεων κάθε κειμένου. Πρόκειται για μια απλή μαθηματική πράξη: κάθε στοιχείο στη θέση j κάθε διανύσματος όρου προστίθεται και το άθροισμα διαιρείται με το πλήθος των διανυσμάτων των όρων που λάβαμε υπόψη μας, δηλαδή το πλήθος των όρων του κειμένου (είναι παρόμοιο με τον υπολογισμό του αριθμητικού μέσου) (Κώδικας 1.0).

---

```
public static INDArray toDenseAverageVector(Word2Vec word2Vec, String...
terms) {

    return word2Vec.getWordVectorsMean(Arrays.asList(terms));

}
```

---

#### Κώδικας 1.0: DL4J παράδειγμα υπολογισμού του mean διανύσματος

Η τεχνική αυτή μπορεί να εφαρμοστεί τόσο στα κείμενα όσο και στα ερωτήματα αφού και αυτά αποτελούν ακολουθίες όρων. Τα διανύσματα που θα προκύψουν θα είναι πυκνά, με περισσότερη σημασιολογική πληροφορία από τα διανύσματα των άλλων μοντέλων που είδαμε, αλλά ταυτόχρονα, θα απαιτούν λιγότερη μνήμη (ή χώρο στο δίσκο) για να αποθηκευτούν. Διαισθητικά, οι όροι που το word2vec θα έχει θεωρήσει ότι έχουν παρόμοια σημασιολογία θα έχουν παρόμοια διανύσματα, επομένως τα διανύσματα των κειμένων και του ερωτήματος που θα προκύψουν από την παραπάνω διαδικασία και θα αποτελούνται από σημασιολογικά παρόμοιους όρους θα θεωρούνται παρόμοια.

Αφού δημιουργηθούν τα διανύσματα των κειμένων και των ερωτημάτων, στη συνέχεια, υπολογίζεται η ομοιότητά τους με βάση το πόσο κοντά βρίσκονται στο χώρο. Τα βήματα που θα πρέπει να ακολουθήσετε είναι τα παρακάτω:

1. Εκπαιδεύστε ένα μοντέλο `word2vec` (Κώδικας 2.0) χρησιμοποιώντας τη συλλογή κειμένων ως είσοδο και τη βιβλιοθήκη `DeepLearningForJava` (DL4J), η οποία παρέχει υλοποιημένα μοντέλα νευρωνικών δικτύων, όπως το `word2vec`, για τη `java`. Η διαδικασία της παραγωγής του μοντέλου μπορεί να παραμετροποιηθεί ως προς την αρχιτεκτονική που θα χρησιμοποιηθεί, το μέγεθος παραθύρου ελέγχου και τον αριθμό των διαστάσεων. Οι διαθέσιμες αρχιτεκτονικές είναι οι `Skip-gram` και `CBOW` που θα αναλυθούν στο μάθημα. Το παράθυρο ελέγχου ορίζει τον αριθμό των λέξεων που θα λαμβάνονται υπόψη, πριν και μετά από την κάθε λέξη. Όσον αφορά τον αριθμό των διαστάσεων, αυτός ορίζει την πολυπλοκότητα και το μέγεθος του μοντέλου.

---



```
String filePath = new ClassPathResource(
    "documents.txt").getFile()
    .getAbsolutePath();
SentenceIterator iter = new BasicLineIterator(filePath);


Word2Vec vec = new Word2Vec.Builder()
    .layerSize(100)
    .windowSize(5)
    .iterate(iter)
    .elementsLearningAlgorithm(new CBOW<>())
    .build();
vec.fit();

String[] words = new String[]{"here", "go", "the", "terms", "of", "the",
    "query"};
for (String w : words) {
    Collection<String> lst = vec.wordsNearest(w, 2);
    System.out.println("2 Words closest to '" + w + "': " + lst);
}
```

---

#### Κώδικας 2.0: DL4J word2vec παράδειγμα

2. Από τα διανύσματα λέξεων που δημιουργήθηκαν από το `word2vec` μοντέλο, δημιουργήστε τα διανύσματα κειμένων και ερωτημάτων υπολογίζοντας το μέσο όρο των διανυσμάτων των όρων τους (Κώδικας 1.0).
3. Υπολογίστε τη συννημιτονοειδή ομοιότητα των ερωτημάτων με τα κείμενα στο νέο χώρο. Ταξινομήστε τα κείμενα σε φθίνουσα σειρά ομοιότητας και συλλέξτε τα  $k$  κείμενα με το μεγαλύτερο σκορ ομοιότητας, για  $k = 5, 30, 50$ .
4. Αξιολογήστε τα αποτελέσματά σας συγκρίνοντάς τα με τις σωστές απαντήσεις (αρχείο `qrels.txt`) χρησιμοποιώντας το εργαλείο αξιολόγησης `trec_eval` και τα μέτρα αξιολόγησης MAP (mean average precision) και `Precision@k` (μέση ακρίβεια στα  $k$  πρώτα ανακτηθέντα κείμενα) για  $k = 5, 10, 20$ . 
5. Καταγράψτε τα πειράματά σας σε μια αναφορά. Περιγράψτε πώς υλοποιήσατε τα παραπάνω βήματα, συμπεριλάβετε screenshots όπου θεωρείτε χρήσιμο (εγκατάσταση εργαλείων, εκτέλεση κώδικα, εκτέλεση `trec_eval`), και φτιάξτε έναν πίνακα με τα αποτελέσματα του `trec_eval`  τις διάφορες τιμές του  $k$ . Στην αναφορά σας συγκρίνετε τα αποτελέσματα της Φάσης 4 με τα αποτελέσματα των προηγούμενων φάσεων. Υπάρχει κάποια βελτίωση; Προσπαθήστε να αιτιολογήσετε τα αποτελέσματά σας είτε αυτά είναι θετικά είτε αρνητικά.
6. Ένας τρόπος για να βελτιωθούν τα αποτελέσματά σας είναι να εκπαιδεύσετε το `word2vec` αλγόριθμο σε περισσότερα δεδομένα εκπαίδευσης. Επειδή αυτό χρειάζεται πολλούς υπολογιστικούς πόρους για να πραγματοποιηθεί μπορούμε να χρησιμοποιήσουμε έτοιμα προ-εκπαιδευμένα `word2vec` μοντέλα. Υπάρχουν στο διαδίκτυο διαθέσιμα μοντέλα που έχουν

εκπαιδευτεί σε μεγάλο όγκο δεδομένων για παράδειγμα σε ένα dump από τη Wikipedia. Χρησιμοποιήστε το προεκπαιδευμένο μοντέλο που θα βρείτε [εδώ](#) (μέγεθος 580Mbytes). Πρόκειται για ένα μοντέλο το οποίο έχει εκπαιδευτεί σε ένα dump της Wikipedia του Φεβρουαρίου του 2017. Το μοντέλο γνωρίζει 273.930 διαφορετικές αγγλικές λέξεις. 

7. Επαναλάβετε τα βήματα 2 έως 5 παραπάνω χρησιμοποιώντας το συγκεκριμένο προεκπαιδευμένο μοντέλο.
8. Δημιουργήστε ένα αρχείο pdf με την αναφορά σας. Υποβάλετε την αναφορά σας, τον κώδικά σας και τα αποτελέσματα του `trec_eval` σε ένα αρχείο zip με ονομασία `αριθμός_μητρώου1_αριθμός_μητρώου2.zip` (πχ. `3200100_3200200.zip`). Μη συμπεριλάβετε τη συλλογή κειμένων. Καταγράψτε τις πηγές σας.

Για τη φάση αυτή, δείτε την έτοιμη κλάση `WordEmbeddingsSimilarity.java`<sup>4</sup> που δίνεται στο `eclass`.

#### Φάση 4 - Ανάκτηση χρησιμοποιώντας πολλαπλές συναρτήσεις ομοιότητας – ΠΡΟΑΙΡΕΤΙΚΗ!

Μία ίσως καλύτερη λύση που υποστηρίζεται και από πρόσφατη έρευνα<sup>3</sup>, προτείνει το συνδυασμό κλασικών μοντέλων ανάκτησης και μοντέλων που χρησιμοποιούν νευρωνικά δίκτυα, χρησιμοποιώντας πολλαπλές συναρτήσεις ομοιότητας την ίδια στιγμή. Αυτό μπορεί να πραγματοποιηθεί στη Lucene με την κλάση `MultiSimilarity`. Εξερευνήστε διαφορετικούς συνδυασμούς των μοντέλων των προηγούμενων φάσεων και επαναλάβετε τη Φάση 1 τροποποιώντας κατάλληλα τη συνάρτηση ομοιότητας της Lucene. Δοκιμάστε συνδυασμούς όπως: `Classic+BM25`, `Classic+LM`, `BM25+LM`, `WV+BM25`, `WV+Classic`, `WV+LM`.

#### Υλοποίηση

Η υλοποίηση της μηχανής αναζήτησης *προτείνεται* να πραγματοποιηθεί με χρήση Java, Lucene και DL4J. Μπορείτε να δοκιμάσετε μια άλλη μηχανή αναζήτησης (πχ. `ElasticSearch`, `Solr`) και άλλη γλώσσα προγραμματισμού (πχ. `python`), αλλά πιθανόν να έχετε περιορισμένη υποστήριξη από τη διδάσκουσα.

#### Βιβλιογραφία - Πηγές:

<sup>1</sup> Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 1–12.

<sup>2</sup> Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9.

<sup>3</sup> Dwaipayan Roy et al., (2016). Representing Documents and Queries as Sets of Word Embedded Vectors for Information Retrieval, Neu-IR '16 SIGIR Workshop on Neural Information Retrieval (Pisa, Italy).

<sup>4</sup> Ο κώδικας είναι από το βιβλίο *Deep Learning for Search*, Tommaso Teofili, 2019, Manning Publications.

**Εργαλεία:** τα εργαλεία που θα χρειαστείτε μπορείτε να τα βρείτε παρακάτω

Lucene	<a href="https://lucene.apache.org/">https://lucene.apache.org/</a>  <a href="https://lucene.apache.org/core/downloads.html">https://lucene.apache.org/core/downloads.html</a> - κατεβάστε από εδώ την τελευταία έκδοση της σειράς 7.x όχι της 8.x.
DL4J	<a href="https://deeplearning4j.org/">https://deeplearning4j.org/</a>  Παράδειγμα υλοποίησης word2vec με χρήση DL4J:  <a href="https://jrmerwin.github.io/deeplearning4j-docs/programmingguide/07_nlp">https://jrmerwin.github.io/deeplearning4j-docs/programmingguide/07_nlp</a>
Java 8++	
trec_eval	<a href="https://trec.nist.gov/trec_eval/">https://trec.nist.gov/trec_eval/</a> (διαθέσιμο στο eclass)

**Ημερομηνίες υποβολής φάσεων εργασίας (ενδέχεται να τροποποιηθούν)**

Φάση 1: 13 Μαΐου 2024 (15 βαθμοί)

Φάση 2: 31 Μαΐου 2024 (5 βαθμοί)

Φάση 3: τέλος εξαμήνου (20 βαθμοί)

Φάση 4: τέλος εξαμήνου (5 βαθμοί)

Η προγραμματιστική εργασία είναι υποχρεωτική και θα προσμετρηθεί στον τελικό βαθμό σας αν ο βαθμός γραπτής εξέτασης  $\geq 4$ . Υποβάλλεται σε ομάδες των 2 φοιτητών. Κάθε φάση θα βαθμολογηθεί ξεχωριστά. Στο τέλος του εξαμήνου θα πραγματοποιηθεί **ατομική** προφορική εξέταση, από την οποία θα εξαρτηθεί ο τελικός βαθμός κάθε φοιτητή στην εργασία.