ΣΥΣΤΗΜΑΤΑ ΑΝΑΚΤΗΣΗ ΠΛΗΡΟΦΟΡΙΩΝ

3ο Παραδοτεο

ΜΠΙΛΙΑΣ ΓΕΩΡΓΙΟΣ 3200278
ΙΩΑΝΝΗΣ ΠΑΤΟΥΧΑΣ 3200149

Η κλάση Phase3 έχει σχεδιαστεί για να εκτελεί διάφορες εργασίες που σχετίζονται με την επεξεργασία φυσικής γλώσσας και την ανάκτηση πληροφοριών. Οι κύριοι στόχοι αυτού του κώδικα είναι να εκπαιδεύσει ένα μοντέλο Word2Vec σε ένα σύνολο εγγράφων, να υπολογίσει τα μέσα διανύσματα λέξεων τόσο για τα ερωτήματα όσο και για τα έγγραφα και, στη συνέχεια, θα αξιολογήσει την ομοιότητα μεταξύ αυτών των διανυσμάτων για την ταξινόμηση των εγγράφων ως απόκριση σε ερωτήματα.

1. **IndexReader και Word2Vec Model initialization**

   Ξεκινάμε αρχικοποιώντας ένα IndexReader για την ανάγνωση των εγγράφων με ευρετήριο και στη συνέχεια το χρησιμοποιούμε για να δημιουργήσουμε έναν FieldValuesSentenceIterator για επανάληψη μέσω του περιεχομένου του εγγράφου. Το μοντέλο Word2Vec έχει ρυθμιστεί με καθορισμένες παραμέτρους όπως το μέγεθος του επιπέδου και το μέγεθος του παραθύρου.

```
IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths.get("Phase
1\\src\\main\\index")));
        FieldValuesSentenceIterator iter = new
FieldValuesSentenceIterator(reader, "content");
        Word2Vec vec = new Word2Vec.Builder()
                .layerSize(100)
                .windowSize(5)
                .tokenizerFactory(new DefaultTokenizerFactory())
                .iterate(iter)
                .build();
        vec.fit();
```

2. Ανάγνωση ερωτημάτων από αρχείο

Ορίζουμε τη μέθοδο readQueriesFromFile για ανάγνωση και ανάλυση ερωτημάτων από ένα αρχείο. Τα ερωτήματα αποθηκεύονται σε έναν χάρτη όπου κάθε αναγνωριστικό ερωτήματος αντιστοιχεί σε μια λίστα όρων.

```
      private static void readQueriesFromFile(Map<String, List<String>>
queriesMap) {

    try (BufferedReader reader = new BufferedReader(new FileReader("Phase
1\\src\\main\\queries.txt"))) {
        String line;
        String currentQueryId = null;
        List<String> currentQueryTerms = null;

        while ((line = reader.readLine()) != null) {
            String[] parts = line.split("///");
            if (parts.length > 0) {
                String queryText = parts[0].trim();

                if (queryText.startsWith("Q") && queryText.length() <= 3) {
                    if (currentQueryId != null && currentQueryTerms != null) {
                        queriesMap.put(currentQueryId, currentQueryTerms);
                    }
                    currentQueryId = queryText;
                    currentQueryTerms = new ArrayList<>();
                } else if (!queryText.isEmpty() && currentQueryId != null) {
                    currentQueryTerms.addAll(Arrays.asList(queryText.split("\\s+"
)));
                }
            }
        }
        if (currentQueryId != null && currentQueryTerms != null) {
            queriesMap.put(currentQueryId, currentQueryTerms);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3. Υπολογισμός μέσου όρου διανυσμάτων

Για κάθε ερώτημα και έγγραφο, υπολογίζουμε τα μέσα διανύσματα λέξεων χρησιμοποιώντας τη μέθοδο toDenseAverageVector. Αυτή η μέθοδος παίρνει μια λίστα όρων και επιστρέφει τη μέση διανυσματική τους αναπαράσταση.

```
public static INDArray toDenseAverageVector(Word2Vec word2Vec, String... terms) {
    return word2Vec.getWordVectorsMean(Arrays.asList(terms));
}
```

4. Ανάλυση εγγράφων

We parse the document file and store the content of each document. The document vectors are computed similarly to the query vectors.

```
List<MyDoc> docs = new ArrayList<>();
try {
    docs = TXTParsing.parse("Phase 1\\src\\main\\documents.txt");
} catch (Exception e) {
    throw new RuntimeException(e);
}

// Initialize the map to store document vectors
Map<String, INDArray> docVectorsMap = new HashMap<>();

for (MyDoc doc : docs) {
    String docId = doc.getID();
    String docContent = doc.getContent();
    INDArray averageVector = toDenseAverageVector(vec, docContent.split("\\s+"));
    docVectorsMap.put(docId, averageVector);
}
```

5. Υπολογιζοντας cosine Similarity

Υπολογίζουμε την ομοιότητα συνημιτόνου μεταξύ των διανυσμάτων ερωτήματος και των διανυσμάτων εγγράφων χρησιμοποιώντας τη μέθοδο cosineSimilarity.

```
private static double cosineSimilarity(INDArray vec1, INDArray vec2) {
        return VectorizeUtils.cosineSimilarity(vec1.toDoubleVector(),
vec2.toDoubleVector());
    }
```

6. Ranking Documents

Για κάθε ερώτημα, ταξινομούμε τα έγγραφα με βάση τις βαθμολογίες ομοιότητάς τους. Τα κορυφαία k έγγραφα εκτυπώνονται και εγγράφονται σε ένα αρχείο αποτελεσμάτων.

```
    FileWriter writer = new FileWriter("Phase
1\\src\\main\\trec_eval\\results2.txt");

    for (Map.Entry<String, INDArray> queryEntry : vectorsMap.entrySet()) {
    String queryId = queryEntry.getKey();
    INDArray queryVector = queryEntry.getValue();
    docSimilarities = new HashMap<>();

    for (Map.Entry<String, INDArray> docEntry : docVectorsMap.entrySet()) {
        String docId = docEntry.getKey();
        INDArray docVector = docEntry.getValue();
        double similarity = cosineSimilarity(queryVector, docVector);
        docSimilarities.put(docId, similarity);
    }
```

```
    sortedDocs = new ArrayList<>(docSimilarities.entrySet());
    sortedDocs.sort(Map.Entry.<String, Double>comparingByValue().reversed());

    int k = 50;
    int rank = 1;
    for (int i = 0; i < Math.min(k, sortedDocs.size()); i++) {
        Map.Entry<String, Double> docEntry = sortedDocs.get(i);
        String docId = docEntry.getKey();
        double score = docEntry.getValue();
        writer.write(String.format("%s\tQ0\t%s\t%d\t%f\tMySystem\n", queryId,
docId, rank++, score));
    }
```

```
runid                   all MySystem
num_q                   all 10
num_ret                 all 500
num_rel                 all 152
num_rel_ret             all 38
map                     all 0.1262
gm_map                  all 0.0046
Rprec                   all 0.1603
bpref                   all 0.2566
recip_rank              all 0.2496
iprec_at_recall_0.00    all 0.3123
iprec_at_recall_0.10    all 0.3010
iprec_at_recall_0.20    all 0.2515
iprec_at_recall_0.30    all 0.2300
iprec_at_recall_0.40    all 0.1778
iprec_at_recall_0.50    all 0.1471
iprec_at_recall_0.60    all 0.1000
iprec_at_recall_0.70    all 0.0467
iprec_at_recall_0.80    all 0.0000
iprec_at_recall_0.90    all 0.0000
iprec_at_recall_1.00    all 0.0000
P_5                     all 0.2600
P_10                    all 0.2000
P_15                    all 0.1533
P_20                    all 0.1350
```

Παρατηρούμε οτι σε σχέση με τις αξιολογήσεις των προηγούμενων φάσεων ο συγκεκριμένος τρόπος αποδίδει

πολύ χειρότερα αποτελέσματα λόγω λάθος υπολογισμού του cosine similarity.

Όσον αφορά τη χρήση του μοντέλου της Wikipedia , αντί να εκπαίδευσουμε μοντέλο , χρησιμοποιήσαμε την εντολή :

```
Word2Vec vec = WordVectorSerializer.readWord2VecModel("Phase
1/src/main/model.bin");
```

Όπου κάνουμε import το model.bin που δόθηκε στην εκφώνηση

Τα αποτελέσματα ήταν πιο χαμηλά απο πριν

```
runid                   all MySystem
num_q                   all 10
num_ret                 all 500
num_rel                 all 152
num_rel_ret             all 33
map                     all 0.0983
gm_map                  all 0.0034
Rprec                   all 0.1249
bpref                   all 0.2374
recip_rank              all 0.2878
iprec_at_recall_0.00    all 0.2959
iprec_at_recall_0.10    all 0.2420
iprec_at_recall_0.20    all 0.1722
iprec_at_recall_0.30    all 0.1389
iprec_at_recall_0.40    all 0.1183
iprec_at_recall_0.50    all 0.1074
iprec_at_recall_0.60    all 0.0708
iprec_at_recall_0.70    all 0.0462
iprec_at_recall_0.80    all 0.0341
iprec_at_recall_0.90    all 0.0000
iprec_at_recall_1.00    all 0.0000
P_5                     all 0.1400
P_10                    all 0.1500
P_15                    all 0.1200
P_20                    all 0.1050
```

Ο λόγος μπορεί να είναι ότι τα διανύσματα μας με αυτά της Wikipedia να απέχουν πολύ.

Επίσης χρειάστηκε και το παρακάτω if-statement λόγω error

```
if (averageVector != null) {
            vectorsMap.put(queryId, averageVector);
        }
```