Συστήματα ανάκτησης πληροφοριών

4η Φάση Προγραμματιστικής Εργασίας

Βιβλιοθήκη LISA

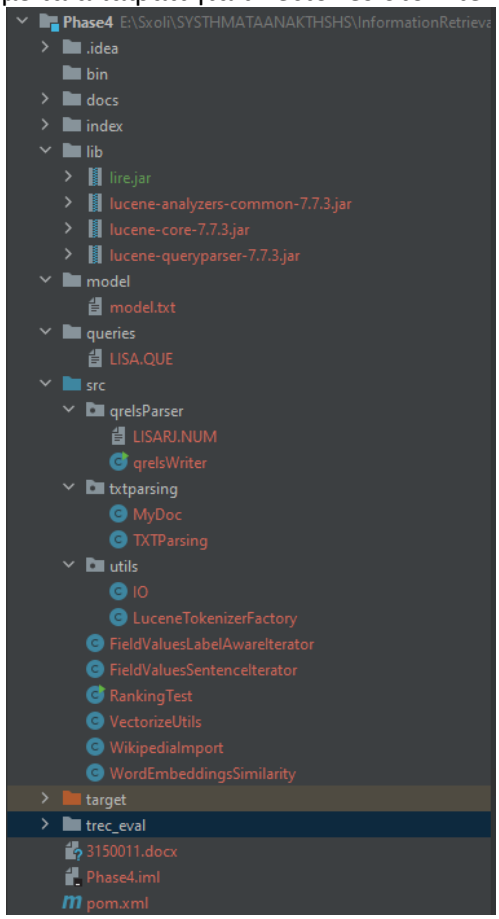
Ον/μο: ΒΙΤΑΛΗΣ ΙΩΑΝΝΗΣ
ΑΜ: 3150011

Εγκατάσταση

Για το setup του project χρησιμοποιήσα τον κώδικα που δώθηκε στην τελευταία διάλεξη αφού πραγματοποιήθηκε εγκατάσταση του Maven. Συγκεκριμένα:

1. Εγκατάσταση Maven
2. Download τον κώδικα που μας δώθηκε και δημιουργία νέου project στο intellij
3. mvn clean install στο φάκελο του project
4. Έλεγχος οτι το project των διαλέξεων λειτουργεί
5. Προσθήκη των .jar άρχειων, lire.jar και των τριών που χρειάζονται για την lucene
6. Προσθήκη κώδικα/αρχείων όπου κρίνω απαραιτητο για την υλοποιήση της εργασίας

Η τελική μορφή της εργασίας με τα απαιραίτητα directories στο Intellij είναι η εξής:



Δομή Κώδικα

Αρχικά χρησιμοποιώ κώδικα που υλοποίησα σε προηγούμενες φάσεις της εργασίας για την δημιουργία του Index με χρήση της Lucene με σημαντική αλλαγή στα πεδία του Fieldtype όπως μας δώθηκε και παρουσιαζεται εδώ:

```
private void indexDoc(IndexWriter indexWriter, MyDoc mydoc){

  try {
    FieldType type = new FieldType(TextField.TYPE_STORED);
    type.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS);
    type.setTokenized(true);
    type.setStored(true);
    type.setStoreTermVectors(true);
    type.setStoreTermVectorOffsets(true);
    type.setStoreTermVectorPositions(true);
```

Ο κώδικας της εργασίας με προεκπαιδευμένο μοντέλο και χωρίς είναι σχεδόν ο ίδιος για αυτό και δεν ήταν απαραίτητη η δημιουργία ενός δεύτερου project

- Για το πρώτο μέρος τις εργασίας στην κλάση RankingTest, το function testRanking() πρέπει να έχει την εξής μορφή:

```
/**Comment to run with the pretrained model*/
FieldValuesSentenceIterator iterator = new FieldValuesSentenceIterator(reader, fieldName);
Word2Vec vec = new Word2Vec.Builder()
        .layerSize(50)
        .windowSize(6)
        .tokenizerFactory(new LuceneTokenizerFactory(new EnglishAnalyzer()))
        .iterate(iterator)
        .seed(12345)
        .build();
vec.fit();


/**Comment the next line to run without the pretrained model*/
//Word2Vec vec = WordVectorSerializer.readWord2VecModel("model\\model.txt");
```

- Ενώ για το δεύτερο μέρος για να γίνει η φόρτωση του προεκπαιδευμένου μοντέλου πρέπει να έχει την εξής μορφή:

```
/**Comment to run with the pretrained model*/
/*FieldValuesSentenceIterator iterator = new FieldValuesSentenceIterator(reader, fieldName);
Word2Vec vec = new Word2Vec.Builder()
        .layerSize(50)
        .windowSize(6)
        .tokenizerFactory(new LuceneTokenizerFactory(new EnglishAnalyzer()))
        .iterate(iterator)
        .seed(12345)
        .build();
vec.fit();*/


/**Comment the next line to run without the pretrained model*/
Word2Vec vec = WordVectorSerializer.readWord2VecModel( path: "model\\model.txt");
```

Οι υπόλοιπες κλάσεις του κώδικα παραμένουν ίδιες με τις προηγούμενες φάσεις (TXTParsing, MyDoc, IO) ενώ οι υπόλοιπες έκρινα πως δεν χρειάζονται αλλαγές για την υλοποίηση της εργασίας και χρησιμοποιήθηκαν όπως μας δώθηκαν.

Η qrelsWriter στον φάκελο qrelsParser είναι ίδια με προηγούμενες φάσεις και χρησιμοποιείται για την δημιοργία του αρχείου LISAQRELS.test στον φάκελο trec_eval που θα χρησιμοποιήσουμε με το trec_eval για την αξιολόγηση.

Λεπτομέρειες στον κώδικα

Οι τιμές για την δημιουργία του Word2Vec (layerSize, windowSize) έγιναν μετά την δοκιμή διαφόρων τιμών και κράτησα τις καλύτερες, η τιμή του layer εξαρτάται από το μέγεθος των δεδομένων οπότε τιμές κάτω του 50 επέστρεφαν χειρότερα αποτελέσματα.

Για την δημιουργία του Index χρησιμοποίησα τον EnglishAnalyzer, οπότε για την δημιουργία του Word2Vec αντικειμένου ο tokenizerFactory πρέπει να χρησιμοποιεί EnglishAnalyzer().

Για το Similarity του IndexSearcher δοκίμασα MEAN και TF_IDF smoothing με το πρώτο να επιστρέφει ελαφρώς καλύτερα αποτελέσματα

Χωρις προεκπαιδευμένο μοντέλο αποτελέσματα στον φάκελο trec_eval αρχεία k20.txt, k30.txt, k50.txt

Μέτρο Αξιολόγησης / **Κ πρώτα Ανακτηθέντα κείμενα**	MAP (Mean Average Precision)	avgPre@k (Average Precision @k)
K = 20	0.1206	P_05 = 0.1657 P_10 = 0.1200 P_15 = 0.0990 P_20 = 0.0914
K = 30	0.1262	P_05 = 0.1657 P_10 = 0.1200 P_15 = 0.0990 P_20 = 0.0914
K = 50	0.1319	P_05 = 0.1657 P_10 = 0.1200 P_15 = 0.0990 P_20 = 0.0914

Παρατηρούμε χειρότερα αποτελέσματα συγκριτικά με την 1η Φάση της προγραμματιστικής εργασίας.

Παρατηρούμε καλύτερα αποτελέσματα συγκριτικά με την 2η Φάση της προγραμματιστικής εργασίας.

Παρατηρούμε χειρότερα αποτελέσματα συγκριτικά με την 3η Φάση της προγραμματιστικής εργασίας και με την χρηση BM25 αλλά και με LMJ.

Ίσως να έχουμε χειρότερα αποτελέσματα από τις φάσεις 1 και 3 επειδή η συλλογή αποτελείται από πολλά κείμενα άρα τα βάρη δεν μπορούν να ρυθμιστούν πολύ αποτελεσματικά.

Με προεκπαιδευμένο μοντέλο αποτελέσματα στον φάκελο trec_eval αρχεία pretrained_k20.txt, pretrained_k30.txt, pretrained_k50.txt

Μέτρο Αξιολόγησης / **Κ πρώτα Ανακτηθέντα κείμενα**	MAP (Mean Average Precision)	avgPre@k (Average Precision @k)
K = 20	0.0084	P_05 = 0.0171 P_10 = 0.0143 P_15 = 0.0152 P_20 = 0.0114
K = 30	0.0087	P_05 = 0.0171 P_10 = 0.0143 P_15 = 0.0152 P_20 = 0.0114
K = 50	0.0087	P_05 = 0.0171 P_10 = 0.0143 P_15 = 0.0152 P_20 = 0.0114

Παρατηρούμε χειρότερα αποτελέσματα συγκριτικά με όλες τις Φάσεις της προγραμματιστικής εργασίας.

Αυτό μπορεί να ωφείλεται σε ένα προεκπαιδευμενο μοντέλο που δεν είναι κατάλληλο για την συλλογή LISA αλλά και σε δικό μου προγραμματιστικό λάθος καθώς δεν πρόλαβα να ελέγξω τυχον

παραμετροποιήσεις που μπορεί να χρειάζονταν για την ενσωμάτωση ενός προεκπαιδευμένου μοντέλου ώστε να παραχθούν ακριβής αποτελέσματα