Εργασία Solr
Συστήματα Ανάκτησης Πληροφοριών

Οικονομικό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής

Κουτσοπούλου Αθανασία Μαρία

3140092

Εισαγωγή

Από τις συλλογές που δινόντουσαν, επιλέχτηκε η τέταρτη, δηλαδή η Cranfield, με βάση το αποτέλεσμα του ΑΜ MOD 6, 3140092 MOD 6 = 4. Στην συλλογή αυτή υπάρχουν τα εξής αρχεία:

- cran.all.1400 – περιέχει όλα τα κείμενα της συλλογής που είναι συνολικά 1400. Το αρχείο έχει την παρακάτω μορφή:
    - .I – συμβολίζει ότι η γραμμή αυτή αναφέρει το id του κειμένου που ακολουθεί.
    - .T – συμβολίζει ότι οι γραμμές που ακολουθούν, έως το επόμενο .Symbol που θα συναντήσουμε, αφορούν τον τίτλο του κειμένου.
    - .A – συμβολίζει ότι οι γραμμές που ακολουθούν, έως το επόμενο .Symbol που θα συναντήσουμε, αφορούν τον συγγραφέα του κειμένου.
    - .B – συμβολίζει ότι οι γραμμές που ακολουθούν, έως το επόμενο .Symbol που θα συναντήσουμε, αφορούν πληροφορίες σχετικά με την βιβλιογραφία του κειμένου.
    - .W – συμβολίζει ότι οι γραμμές που ακολουθούν, έως το επόμενο .Symbol που θα συναντήσουμε, αφορούν περιεχόμενο του ίδιου του κειμένου.

  Ωστόσο, να σημειωθεί ότι σε ορισμένα κείμενα υπήρχε λανθασμένη μορφοποίηση, όπως στο κείμενο με id 240, όπου μετά από όλα τα άνω .Symbol που αναφέρουμε, υπήρχε ξανά το .A και το .B, και η γραμμές που τα ακολουθούσαν δεν είναι δυνατόν να αφορούν πράγματι τον συγγραφέα και την βιβλιογραφία, ή όπως στο κείμενο με id 576 υπήρχε δύο φορές το .W. Σε αυτά αλλά και σε άλλες δύο περιπτώσεις που εμφανίζονταν τέτοιου είδους λάθη στην μορφοποίηση, τα επαναλαμβανόμενα .Symbols και οι γραμμές που τα ακολουθούσαν αγνοήθηκαν.
- cran.qry – περιέχει όλα τα queries που θέλουμε να εκτελέσουμε πάνω στην συλλογή. Το αρχείο έχει την παρακάτω μορφή:
    - .I – συμβολίζει ότι η γραμμή αυτή αναφέρει το id του κειμένου που ακολουθεί.
    - .W – συμβολίζει ότι οι γραμμές που ακολουθούν, έως το επόμενο .Symbol που θα συναντήσουμε, αποτελούν ίδιο το query.

  Ωστόσο, να σημειωθεί ότι οι απαρίθμηση των queries ήταν λανθασμένη. Ενώ τα queries είναι συνολικά 225, τα ids έπαιρναν τυχαίες τιμές από 001 – 365. Έτσι λοιπόν κατά την μετατροπή τους στην κατάλληλη μορφή για να εκτελεστεί το query μέσω του solr, χρησιμοποιήθηκε counter και δόθηκαν τα ids από 1 – 225 στην σειρά.
- cranqrel – περιέχει όλα τα σχετικά κείμενα ανά query. Το αρχείο έχει την παρακάτω μορφή:
    - QueryId RelativeDocId RelevanceNumber

  Να σημειωθεί ότι προκειμένου να γίνει χρήση του trec_eval έγινε επεξεργασία του αρχείου ώστε να έρθει στην επιθυμητή μορφή, δηλαδή:
    - QueryId Iter RelativeDocId RelevanceNumber

  Όπου το iter είναι ένα string που αγνοείται, στην περίπτωσή μας είναι το μηδέν.
- Readme – περιέχει πληροφορίες σχετικά με την μορφή του cranrel, κυρίως για τη σημασία των διαφορετικών τιμών του relevanceNumber.

Το java project που δημιουργήθηκε είναι υπεύθυνο για όλα τα παρακάτω.

Μετατροπή Cran.all.1400 σε newDocs.xml

Αρχικά, έπρεπε να γίνει μετατροπή της συλλογής κειμένων μας, δηλαδή του cran.all.1400 αρχείου, σε αρχείο xml αρχείο με την κατάλληλη μορφή ώστε να είναι δυνατόν να φορτωθούν τα κείμενα στο Solr. Η μορφή του xml είναι η εξής:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<add>
    <doc>
        <field name="identifier"> </field>
        <field name="title"> </field>
        <field name="author"> </field>
        <field name="bAttr"> </field>
        <field name="text"> </field>
    </doc>
</add>
```

Η μετατροπή γίνεται μέσα από την *main* της κλάσης *ModifyCollection*. Αυτή καλεί την μέθοδο *parseFile*, όπου και τελικά υλοποιεί την μετατροπή. Λαμβάνει ως είσοδο το όνομα του αρχείου, δηλαδή το cran.all.1400. Για να χρησιμοποιηθεί το αρχείο στο Solr προκειμένου να φορτωθούν τα κείμενα, δημιουργούμε αρχικά ως root element το "**add**" και σε αυτό προσθέτουμε νέο παιδί element "**doc**" για κάθε νέο κείμενο που συναντάμε στο αρχείο εισόδου. Σε κάθε "doc", έχουμε πέντε παιδιά elements "**field**" με το name attribute να παίρνει τις τιμές που φαίνονται στην παραπάνω εικόνα. Και τέλος σε κάθε "field" element, έχουμε ως παιδί ένα **text node** με το αντίστοιχο περιεχόμενο που βρέθηκε στο αρχείο εισόδου.

Ας προσπαθήσουμε να εξηγήσουμε λίγο παραπάνω την λογική του κώδικα.
Δημιουργούμε το "add" root element και στην συνέχεια προσθέτουμε σε αυτό ως παιδί ένα νέο "doc" element όταν έχουμε ολοκληρώσει όλες τις προσθήκες στο "doc" αυτό. Συγκεκριμένα οι πληροφορίες για τα text nodes λαμβάνονται με την παρακάτω λογική.
Καταρχάς έχουμε 4 StringBuilders που αντιστοιχούν στο περιεχόμενο του "title", "author", "bAttr", και τέλος "text", δηλαδή των .T, .A, .B, .W αντίστοιχα.
Ακόμα έχουμε 4 Boolean μεταβλητές που χρησιμοποιούμε ως flags ώστε όταν διαβάσουμε γραμμή που δεν ξεκινά με κάποιο από τα .Symbol, να ξέρουμε σε πιο StringBuilder πρέπει να κάνουμε append το περιεχόμενο της γραμμής αυτής. Τα flags αυτά παίρνουν τις ανάλογες τιμές κάθε φορά που η γραμμή που διαβάστηκε ξεκινά με κάποιο .Symbol.
Έχουμε λοιπόν ένα Switch Case Statement, που κάθε φορά που διαβάζουμε νέα γραμμή, ελέγχουμε αν ξεκινά με κάποιο .S, και αν ναι εκτελούνται οι αντίστοιχες εντολές του εκάστοτε case που κυρίως δίνουν τιμές σε διάφορα flags που μας χρειάζονται στην συνέχεια προκυμμένου να αποφανθούμε για το τι πρέπει να συμβεί, διαφορετικά εκτελούνται οι εντολές του default, όπου εκεί έχουμε if-else if έτσι ώστε να ανανεώσουμε το σωστό StringBuilders.
Ιδιαίτερη περίπτωση είναι το case .I όπου, εκτός από όσα αναφέρθηκαν για όλα τα case, εδώ – εάν δεν είναι το πρώτο κείμενο – δημιουργούμε τα field elements και τα αντίστοιχα text nodes τους για το προηγούμενο κείμενο. Δηλαδή, αν διαβάσουμε νέο .I, πριν κάνουμε οτιδήποτε άλλο, θα πάμε να δημιουργήσουμε τα fields και όλες τις σχετικές πληροφορίες για το προηγούμενο κείμενο, συνεπώς για το τελευταίο μέχρι στιγμής doc element, χρησιμοποιώντας τα 4 StringBuilders που αναφέραμε

νωρίτερα.
Το ίδιο κάνουμε και όταν φτάσουμε στο τέλος του αρχείου εισόδου, καθώς το τελευταίο doc element, δεν έχει λάβει ακόμη τα fields του, αφού δεν ακολούθησε κάποιο .I στο αρχείο εισόδου.

Solr – Core Creation and Load Files

Αρχικά μέσα από το bin directory ξεκινάμε το Solr, με την εντολή:

solr start

Στην συνέχεια δημιουργούμε νέο core με την εντολή:

solr create -c "informationRet"

, μέσα στο οποίο θα φορτώσουμε τα documents της συλλογής και στην συνέχεια θα εκτελέσουμε τα queries.

Τέλος, πλέον από το main solr directory, φορτώνουμε τα documents της συλλογής από το *newDocs.xml* αρχείο που δημιουργήσαμε στο προηγούμενο βήμα (Μετατροπή cran.all.1400 σε .xml) στο core που μόλις δημιουργήσαμε, μέσω του post.jar τους solr, τρέχοντας την εντολή:

java -Dc="informationRet" -jar example/exampledocs/post.jar newDocs.xml

```
D:\Desktop\Solr\solr-7.2.0\bin>solr start
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!

D:\Desktop\Solr\solr-7.2.0\bin>solr create -c "informationRet"
WARNING: Using _default configset. Data driven schema functionality is enabled by default, which is
         NOT RECOMMENDED for production use.
         To turn it off:
            curl http://localhost:8983/solr/informationRet/config -d '{"set-user-property": {"update.autoCreateFields":"
false"}}'

Created new core 'informationRet'

D:\Desktop\Solr\solr-7.2.0\bin>cd..

D:\Desktop\Solr\solr-7.2.0>java -Dc="informationRet" -jar example/exampledocs/post.jar newDocs.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/informationRet/update using content-type application/xml...
POSTing file newDocs.xml to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/informationRet/update...
Time spent: 0:00:02.547
```

Solr – Modify Managed Schema

Προκειμένου να έχουμε ένα αποδοτικότερο indexing, και να χρησιμοποιήσουμε διάφορες τεχνικές τύπου stemming, stopwords κτλ, τροποποιήσαμε το Managed Schema που δημιούργησε το Solr, το οποίο βρίσκεται στο directory: **solr-7.2.0\server\solr\ informationRet \conf**. Αυτό είχε τα εξής πεδία:

```
<field name="author" type="text_general"/>
<field name="bAttr" type="text_general"/>
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="identifier" type="plongs"/>
<field name="text" type="text_general"/>
<field name="title" type="text_general"/>
```

Όμως το text_general, δεν υλοποιεί καμία από της ζητούμενες τεχνικές. Έτσι λοιπόν, πρώτη μας σκέψη ήταν να χρησιμοποιήσουμε κάποιο από τα υπάρχοντα fieldType πεδία ως type στα πεδία των κειμένων μας, εκτός του identifier, δηλαδή τα title, author, bAttr και text, έτσι ώστε να πετύχουμε το ζητούμενο. Συγκεκριμένα, σκεφτήκαμε να χρησιμοποιήσουμε το text_en όμως προκειμένου να έχουμε μεγαλύτερο

έλεγχο στο ποιος tokenizer αλλά και ποια filtra θα χρησιμοποιηθούν αποφασίσαμε τελικά να δημιουργήσουμε ένα εξ ολοκλήρου νέο fieldType.

Το νέο fieldType ονομάστηκε **text_infoRet** και το θέσαμε ως type στο title, author, bAttr και text. Το οποίο φαίνεται εδώ:

```
<fieldType name="text_infoRet" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
      <tokenizer class="solr.ClassicTokenizerFactory"/>
      <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
      <filter class="solr.EnglishMinimalStemFilterFactory"/>
      <filter class="solr.ClassicFilterFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
      <filter class="solr.KStemFilterFactory"/>
  </analyzer>
  <analyzer type="query">
      <tokenizer class="solr.ClassicTokenizerFactory"/>
      <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
      <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
      <filter class="solr.EnglishMinimalStemFilterFactory"/>
      <filter class="solr.ClassicFilterFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
      <filter class="solr.KStemFilterFactory"/>
  </analyzer>
</fieldType>
```

Σε αυτό χρησιμοποιήσαμε τα εξής:

- **ClassicTokenizerFactory –** ως tokenizer επιλέχτηκε ο συγκεκριμένος όπου κάνει tokenize όπως ακριβώς ο StandardTokenizer με delimeters το space και τα σημεία στίξης, όμως με εξαιρέσεις τα παρακάτω:
    - Τελείες όπου δεν ακολουθούνται από κενό κρατούνται ως μέρος του token, πχ example.keep
    - Παύλα που συνδέει αριθμούς ή λέξεις που περιέχουν αριθμούς κρατείται χωρίς να σπάει, πχ goto-page7
    - Αναγνωρίζει internet domains και emails
- **StopFilterFactory** με words="lang/stopwords_en.txt" ignoreCase="true" – όπου εφαρμόζει την τεχτνική των stopwords, δηλαδή αγνοεί αυτές τις λέξεις που βρίσκονται στο stopwords.txt. Να σημειωθεί εδώ, πως έχουμε ανανεώσει το αρχείο αυτό χειροκίνητα.
- **EnglishMinimalStemFilterFactory –** filter όπου μετατρέπει όποια λέξη βρίσκεται στον πληθυντικό στην αντίστοιχη λέξη στον ενικό.
- **ClassicFilterFactory –** όπου αφαιρεί τις τελείες από τα ακρόνυμα και το κτητικό s πχ Ο.Π.Α. -> ΟΠΑ και cat’s -> cat
- **LowerCaseFilterFactory –** όπου μετατρέπει όλα τα κεφαλαία γράμματα σε μικρα
- **KeywordMarkerFilterFactory** protected="protwords.txt" **–** όπου θέτει ως «προστατευμένες» τις λέξεις που βρίσκονται στο protwords.txt και έτσι δεν υπόκεινται σε stemming
- **KStemFilterFactory –** όπου εφαρμόζει stemming στα tokens, με λιγότερο «επιθετικό» τρόπο από αυτόν του PorterStemFilter

- **SynonymGraphFilterFactory –** το οποίο χρησιμοποιείται μόνο στα queries, και χρησιμοποιεί τις λέξεις που υπάρχουν στο synonyms.txt ώστε να αντικαταστεί όσες υπάρχουν στο query με κάθε συνώνυμο της.

Έτσι λοιπόν έχουμε πλέον το modified managed schema.

Solr Reindex Docs

Προκειμένου να γίνει reindex στα documents, πρέπει αρχικά να κάνουμε restart τον server. Επομένως μέσα στο bin directory τρέχουμε τις παρακάτω εντολές:

solr stop -all
solr start

Τέλος, πρέπει να σβήσουμε τα documents από το core και να τα ξαναφορτώσουμε, ώστε να γίνουν reindex με βάση το modified managed schema. Προκειμένου να διαγράψουμε τα documents χρειαζόμασταν ένα νέο αρχείο όπου και δημιουργήσαμε, το *deleteDocs.xml* και μέσα περιέχει το εξής: *<delete><query>*:*</query></delete>* . Έτσι μέσα στον main solr directory και χρησιμοποιώντας ξανά το post.jar του solr, τρέχουμε τις παρακάτω εντολές:

java -Dc=”informationRet” -jar example/exampledocs/post.jar deleteDocs.xml
java -Dc=”informationRet” -jar example/exampledocs/post.jar newDocs.xml

```
D:\Desktop\Solr\solr-7.2.0\bin>solr stop -all
Stopping Solr process 13256 running on port 8983

Waiting for 4 seconds, press a key to continue ...

D:\Desktop\Solr\solr-7.2.0\bin>solr start
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!

D:\Desktop\Solr\solr-7.2.0\bin>cd ..

D:\Desktop\Solr\solr-7.2.0>java -Dc="informationRet" -jar example/exampledocs/post.jar deleteDocs.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/informationRet/update using content-type application/xml...
POSTing file deleteDocs.xml to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/informationRet/update...
Time spent: 0:00:00.576

D:\Desktop\Solr\solr-7.2.0>java -Dc="informationRet" -jar example/exampledocs/post.jar newDocs.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/informationRet/update using content-type application/xml...
POSTing file newDocs.xml to [base]
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/informationRet/update...
Time spent: 0:00:02.838
```

Μετατροπή cranqrel

Προκειμένου να τρέξουμε το trec_eval και να πάρουμε τα στατιστικά αποτελέσματα για τα documents που γίνονται retrieve μέσω του Solr, πρέπει να μετατρέψουμε το cranqrel όπως αναφέραμε στην εισαγωγή. Επομένως, έχουμε δημιουργήσει την μέθοδο *modifyQrelFIle* στην κλάση *ModifyCollection*, όπου στην ουσία διαβάζουμε όλο το αρχείο, και στην συνέχεια ξαναγράφουμε από την αρχή στο αρχείο από την αρχή σβήνοντας όλα τα προηγούμενα. Όταν το ξαναγράφουμε, παίρνουμε γραμμή γραμμή ότι διαβάσαμε, κάνουμε trim, και μετά αντικαθιστούμε το πρώτο “space” που συναντάμε με “space 0 space”. Η *modifyQrelFIle* καλείται από την *main* της *ExecuteQueries.*

Queries Execution

Το query execution γίνεται με την *main* της *ExecuteQueries*.
Να σημειωθεί πως έχουμε δημιουργήσει μια βοηθητική κλάση Query, με πεδία το id και το query string. Έτσι αρχικά μέσω της μεθόδου *readAndTransformQueries*, στην κλάση *ExecuteQueries*, διαβάζουμε όλα τα queries από το cran.qry και δημιουργούμε νέο Query object και το προσθέτουμε στο *queries Arraylist* της κλάσης. Το αρχείο όπως αναφέραμε στην εισαγωγή έχει συγκεκριμένη μορφή, επομένως, κάθε γραμμή που διαβάζουμε ελέγχουμε αν ξεκινά με .I, τότε αν δεν είναι το πρώτο query, τότε προσθέτουμε το query string που διαβάσαμε μέχρι τώρα στο προηγούμενο Query object, και δημιουργούμε και το νέο Query object μόνο με id μέχρι στιγμής. Όποτε δεν ξεκινά με .I, κάνουμε append στο StringBuilder που χρησιμοποιούμε για να κατασκευάσουμε το query string. (Όμοια λογική με το διάβασμα του cran.all.1400.)
Αφού λοιπόν έχουμε πλέον το *queries Arraylist* με όλα τα queries, τότε μέσω της *runQueries* της κλάσης *ExecuteQueries,* τρέχουμε ένα ένα τα queries. (Η *runQueries* παίρνει μία Boolean παράμετρο η οποία είναι true αν θέλουμε να τρέξουμε τα queries με rerank, διαφορερικά είναι false και τρέχουμε τα queries χωρίς rerank. Στην συγκεκριμένη φάση λοιπόν παιρνάμε false.) Συγκεκριμένα, αρχικά δημιουργούμε ένα νέο solr client και για κάθε ένα query μέσα στο Arraylist, πρώτα κάνουμε tokenize το query string με το κενό, και δημιουργούμε ένα stringbuilder ως εξής: “text:token OR text:token2 …”. Έπειτα δημιουργούμε ένα HashMap με τα query parameters, όπως το q, rows και fl, το οποίο το περνάμε στο MapSolrParams. Στη συνέχεια, παίρνουμε το response από τον client ως ένα SolrDocumentList, και για κάθε ένα document από αυτά γράφουμε στο νέο μας txt αρχείο με τα αποτελέσματα, μια εγγραφή της μορφής: qid 0 docId 1 score STANDARD. Το όνομα του αρχείου μας είναι το queryResults.

Trec Eval First Evaluation – Χωρίς Rerank

Για την αξιολόγηση των αποτελεσμάτων μας, τρέχουμε την εντολή:

trec_eval -q -a cranqrel queryResults

```
The following measures are interpolated versions of measures above.
For the following, interpolated_prec(X) == MAX (prec(Y)) for all Y >= X
All these measures are experimental

Average interpolated precision over all rel docs
                  0.4196
R-based-interpolated-Precision:
    Exact:        0.4183
    At 0.20  R:   0.7253
    At 0.40  R:   0.5947
    At 0.60  R:   0.5162
    At 0.80  R:   0.4512
    At 1.00  R:   0.4183
    At 1.20  R:   0.3599
    At 1.40  R:   0.3243
    At 1.60  R:   0.2991
    At 1.80  R:   0.2728
    At 2.00  R:   0.2606
Average interpolated precision for first R docs retrieved:
                  0.4980
```

Έτσι βλέπουμε ότι το average precision μας όταν επιστρέφουμε 500 rows φτάνει το 0.4196.
Στο rar της εργασία έχουμε προσθέσει και το firstTrecEval.txt όπου περιέχει τα αποτελέσματα τις εντολής συνολικά.

Reranking

Προκειμένου να κάνουμε rerank στα queries μας, χρησιμοποιήθηκε η τεχνική learning to rank που παρέχεται από το Solr.

Στο rerank αποφασίσαμε να δώσουμε **έμφαση στην μεγαλύτερη λέξη του query**, καθώς λόγω του είδους των κειμένων τις συλλογής θεωρούμε ότι αυτή έχει την μεγαλύτερη σημασία και θα εμφανίζεται πιο σπάνια στα κείμενα. Επίσης έτσι δύσκολα θα επιλεγεί μια stopword που θα ήταν άσκοπο. Επιπλέον, σκεφτήκαμε να χρησιμοποιήσουμε και την λέξη που υπάρχει περισσότερες φορές μέσα στο query, αλλά παρατηρήσαμε ότι κάτι τέτοιο θα ήταν ανούσιο, καθώς τα περισσότερα query είναι αρκετά σύντομα και δεν έχουν επαναλαμβανόμενες λέξεις.
Ακόμα, αποφασίσαμε να δώσουμε δύο διαφορετικά βάρη, ανάλογα με το που εντοπίζουμε τη συγκεκριμένη λέξη σε ένα document της συλλογής. Έτσι δίνουμε **μεγαλύτερο βάρος αν η λέξη εντοπιστεί στον τίτλο**, και λιγότερο αν εντοπιστεί στο περιεχόμενο συνολικά.

Να σημειωθεί ότι, στη *runQueries*, που καλείται από την main της *ExecuteQueries*, αυτή την φορά περνάμε ως παράμετρο true. Επομένως, θα αναλάβει να τρέξει τα queries με rerank. Έτσι, εσωτερικά κατά την κατασκευή του query param, καλεί την *maxLengthWordInQuery* που είναι υπεύθυνη για την εύρεση της λέξης με το μεγαλύτερο μήκος.

Για να εφαρμόσουμε λοιπόν το leraning to rank ακολουθήσαμε τα παρακάτω βήματα:

- Προσθήκη στο solrconfig.xml, που βρίσκεται στο φάκελο conf του core μας, των παρακάτω γραμμών, έτσι ώστε να ενεργοποιήσουμε το learning to rank.

```
<lib dir="${solr.install.dir:../../../..}/contrib/ltr/lib/" regex=".*\.jar" />
<lib dir="${solr.install.dir:../../../..}/dist/" regex="solr-ltr-\d.*\.jar" />

<queryParser name="ltr" class="org.apache.solr.ltr.search.LTRQParserPlugin"/>

<cache name="QUERY_DOC_FV"
       class="solr.search.LRUCache"
       size="4096"
       initialSize="2048"
       autowarmCount="4096"
       regenerator="solr.search.NoOpRegenerator" />

<transformer name="features" class="org.apache.solr.ltr.response.transform.LTRFeatureLoggerTransformerFactory">
  <str name="fvCacheName">QUERY_DOC_FV</str>
</transformer>
```

- Στην συνέχεια ξεκινούμε τον solr server με την εντολή:

  solr start -Dsolr.ltr.enabled=true

```
C:\Users\Nasia LiAD>solr start -Dsolr.ltr.enabled=true
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!
```

- **Έπειτα δημιουργούμε τα features που θα εισάγουμε στο core μας. Με βάση την ιδέα μας για το rerank που αναφέραμε άνω (στην αρχή του rerank section), κατασκευάζουμε το**

myEfiFeatures.json και τρέχουμε την παρακάτω εντολή χρησιμοποιώντας το curl, ώστε να τα ανεβάσουμε στο core:

```
C:\Users\Nasia LiAD>curl -XPUT "http://localhost:8983/solr/sr1/schema/feature-store" --data-binary "@D:\Desktop\Solr\Inf
ormationRetrievalSolr\config\myEfiFeatures.json" -H "Content-type:application/json"
Jan 31, 2018 3:59:28 PM org.restlet.Application start
INFO: Starting org.apache.solr.rest.SolrSchemaRestApi application
Jan 31, 2018 3:59:28 PM org.restlet.engine.log.LogFilter afterHandle
INFO: 2018-01-31        15:59:28        0:0:0:0:0:0:0:1 -       0:0:0:0:0:0:0:1 8983    PUT     /solr/schema/feature-sto
re      -       200     -       539     44      http://localhost:8983   curl/7.58.0     -
{
  "responseHeader":{
    "status":0,
    "QTime":309}}
```

- Επιπλέον, πρέπει να δημιουργήσουμε και να ανεβάσουμε στο core και το μοντέλο του learning to rank. Κατασκευάζουμε λοιπόν, με βάση πάλι την ιδέα μας για το rerank που αναφέραμε ανω, το myEfiModel.json. Να τονίσουμε ωστόσο ότι λόγω ότι δεν υπήρχε κάποιο training-set στην διάθεση μας προκειμένου να χρησιμοποιήσουμε κάποιον αλγόριθμο μηχανικής μάθησης ώστε να υπολογίζει τα βάρη, τα ορίζουμε χειρωνακτικά. Θέλουμε να δώσουμε περισσότερη έμφαση αν η λέξη που επιλέχτηκε – δηλαδή η μεγαλύτερη λέξη του query – υπάρχει στον τίτλο, ακομά έμφαση, αν και λιγότερη αν η λέξη υπάρχει στο κείμενο γενικά, και υποβαθμίζουμε λίγο το score που είχε το doc λάβει χωρίς το rerank. Έτσι θέτουμε τα εξής βάρη:
  - wordOnTitle: 1.5
  - wordOnText: 1.0
  - originalScore: 0.8

  Και απλά τρέχουμε την παρακάτω εντολή για να το ανεβάσουμε στο core.

```
C:\Users\Nasia LiAD>curl -XPUT "http://localhost:8983/solr/sr1/schema/model-store" --data-binary "@D:\Desktop\Solr\Infor
mationRetrievalSolr\config\myEfiModel.json" -H "Content-type:application/json"
Jan 31, 2018 4:00:21 PM org.restlet.engine.log.LogFilter afterHandle
INFO: 2018-01-31        16:00:21        0:0:0:0:0:0:0:1 -       0:0:0:0:0:0:0:1 8983    PUT     /solr/schema/model-store
        -       200     -       372     13      http://localhost:8983   curl/7.58.0     -
{
  "responseHeader":{
    "status":0,
    "QTime":12}}
```

- Τέλος, τρέχουμε την main της *ExecuteQueries,* έχοντας αλλάξει την τιμή της μεταβλητής *useRerank* σε true, έτσι ώστε να καλεστεί η *runQueries* με παράμετρο true, για να εκτελέσει τα queries με το learnk to rank.

Trec Eval Second Evaluation – Με Rerank

Για την αξιολόγηση των νέων αποτελεσμάτων μας, τρέχουμε ξανά την εντολή:

trec_eval -q -a cranqrel queryResults

Έτσι βλέπουμε ότι το average precision μας όταν επιστρέφουμε 500 rows φτάνει πλέον το 0.3814 σε αντίθεση με το 0.4196 που είχαμε πριν το rerank.

Η πτώση αυτή θεωρούμε ότι είναι αναμενόμενη, καθώς το dataset μας ήταν μικρό για να δούμε κάποια ουσιαστική βελτίωση λόγω του rerank, και επιπλέον δεν είχαμε κάποιο train set προκειμένου να χρησιμοποιήσουμε κάποιον αλγόριθμο μηχανικής μάθησης, που πιθανώς να είχε καλύτερα αποτελέσματα.
Στο rar της εργασία έχουμε προσθέσει και το secondTrecEval.txt όπου περιέχει τα αποτελέσματα τις εντολής συνολικά.

```
The following measures are interpolated versions of measures above.
For the following, interpolated_prec(X) == MAX (prec(Y)) for all Y >= X
All these measures are experimental

Average interpolated precision over all rel docs
                0.3813
R-based-interpolated-Precision:
    Exact:      0.3965
    At 0.20  R:   0.6579
    At 0.40  R:   0.5403
    At 0.60  R:   0.4812
    At 0.80  R:   0.4229
    At 1.00  R:   0.3965
    At 1.20  R:   0.3433
    At 1.40  R:   0.3084
    At 1.60  R:   0.2849
    At 1.80  R:   0.2616
    At 2.00  R:   0.2513
Average interpolated precision for first R docs retrieved:
                0.4584
```

Σημείωση, properties φάκελος

Μέσα σε αυτόν τον φάκελο παραθέτουμε όλα τα αναγκαία αρχεία που χρησιμοποιήθηκαν και τροποποιήθηκαν κατά την άνω διαδικασία, όλης της εργασίας, καθώς και τα διάφορα αρχεία αποτελεσμάτων. Συγκεκριμένα:

- deletedDocs.txt – το χρησιμοποιούμε για να διαγράφουμε όλα τα docs του core, προκειμένου να κάνουμε reindex.
- firstTrecEval.txt – τα αποτελέσματα του trec eval χωρίς το rerank.
- managed-schema – το επεξεργασμένο μας managed schema με το νέο field type μας το text_irTest, που χρησιμοποιούμε για το indexing και το query.
- myEfiFeatures.json: τα features για το Learning-to-Rank.
- myEfiModel.json: το μοντέλο για το Learning-to-Rank.
- secondTrecEval.txt – τα αποτελέσματα του trec eval με το rerank.
- solrconfig: το επεξεργασμένο αρχείο που προσθέσαμε τις απαιτούμενες γραμμές ώστε να ενεργοποιήσουμε το ltr όπως αναφέραμε άνω.
- stopwords_en.txt: το επεξεργασμένο αρχείο που προσθέσαμε τις πιο συχνές αγγλικές λέξεις ώστε αγνοούνται.