Information Retrieval and Data Mining

Porject 1 2020-2021

Team 2

Karypidis Stathis [Github]

Anagnostou Pantazis [Github]

Abstract: In this project we conduct an experiment on the basics of information retrieval. The project was based on the Lemur Project or Indri Search Engine. More specifically, we have 4 collections of texts (fbis, fr94, ft, latimes) and 150 information needs/topics which are tested on two different retrieval models. The first one is the default model that Indri has and its results will be our baseline results. On the other model we use query expansion with Relevance Feedback of terms of the collections or terms from a thesaurus. The version of the Lemur Porject used is 5.11 on Ubuntu 16.04 and the evaluation tool used in trec_eval 9.07.

The code of this project can be found on github: https://github.com/pantanag/IR-DM-Exercise-2020-21-Team-2

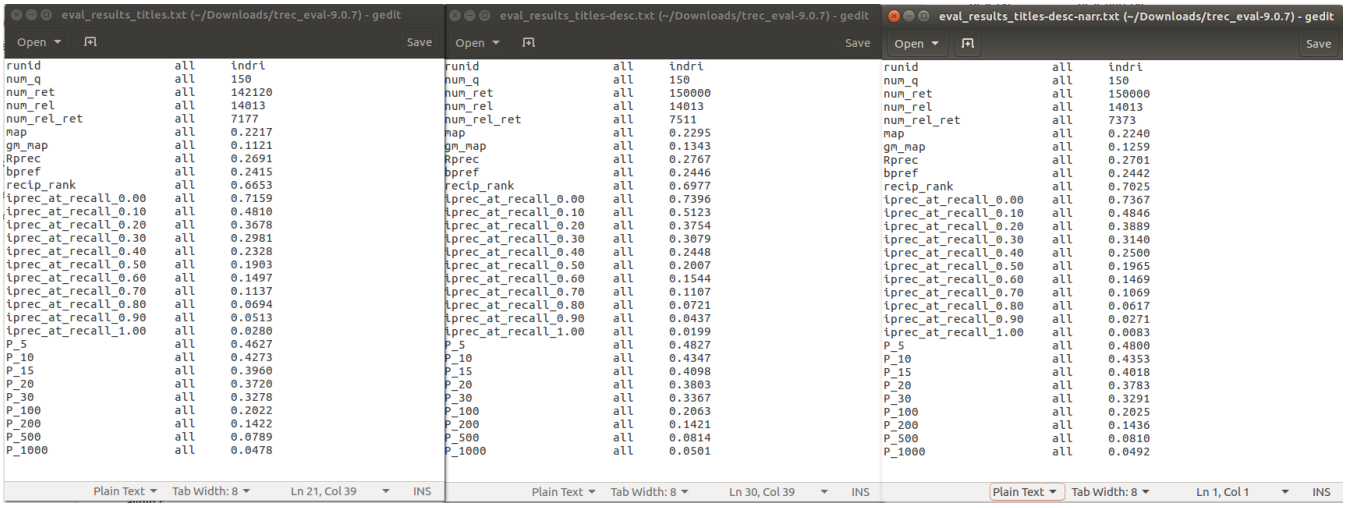
Part A

In the first part of this project we use the default retrieval model of Indri. After the setup of Indri is complete, we use the command to build our index of terms of our collections (default tokenizer and Krovetz stemmer) after we have modified the parameter file "IndriBuildIndex.parameter.file.Example" to our needs. More specifically, we add the paths to our collections and the path that our index is going to be stored.

```
<parameters>
 <corpus>
   <path>/home/stathis/Downloads/IR-2019-2020-Project-1/ft</path>
   <class>trectext</class>
 </corpus>
 <corpus>
   <path>/home/stathis/Downloads/IR-2019-2020-Project-1/fr94</path>
   <class>trectext</class>
 </corpus>
 <corpus>
   <path>/home/stathis/Downloads/IR-2019-2020-Project-1/fbis</path>
   <class>trectext</class>
 </corpus>
 <corpus>
   <path>/home/stathis/Downloads/IR-2019-2020-Project-1/latimes</path>
   <class>trectext</class>
 </corpus>
 <index>/home/stathis/Downloads/IR-2019-2020-Project-1/indices</index>
 <memory>4096M</memory>
 <storeDocs>true</storeDocs>
 <stemmer><name>Krovetz</name></stemmer>
</parameters>
```

Next we use the command mentioned before `IndriBuildIndex IndriBuildIndex.parameter.file.Example` to build our index. With the command `dumpindex /path/to/index v` we can see our index too. The next step is the creation of a joint file containing all the qrels files (relevant judgments) with the use of the command `cat qrels.301-350.trec6.adhoc qrels.351-400.trec7.adhoc qrels.401-450.trec8.adhoc > qrels.301-450.trec.adhoc` and the creation of a file containing all the information needs (topics) with the corresponding command `cat topics.301-350.trec6 topics.351-400.trec7 topics.401-450.trec8 > topics.trec`. If we are given the above files we can skip to next part which is the creation of the three (3) queries files. After having read the queries file with only the titles (was given as an example) we proceed to remove any punctuation and handle any specific characters so that our 3 files are in the best format possible (and that Indri can understand them).

Our results were the below:



eval_results_titles.txt (~/Downloads/trec_eval-9.0.7) - gedit

```
runid                   all     indri
num_q                   all     150
num_ret                 all     142120
num_rel                 all     14013
num_rel_ret             all     7177
map                     all     0.2217
gm_map                  all     0.1121
Rprec                   all     0.2691
bpref                   all     0.2415
recip_rank              all     0.6653
iprec_at_recall_0.00    all     0.7159
iprec_at_recall_0.10    all     0.4810
iprec_at_recall_0.20    all     0.3678
iprec_at_recall_0.30    all     0.2981
iprec_at_recall_0.40    all     0.2328
iprec_at_recall_0.50    all     0.1903
iprec_at_recall_0.60    all     0.1497
iprec_at_recall_0.70    all     0.1137
iprec_at_recall_0.80    all     0.0694
iprec_at_recall_0.90    all     0.0513
iprec_at_recall_1.00    all     0.0280
P_5                     all     0.4627
P_10                    all     0.4273
P_15                    all     0.3960
P_20                    all     0.3720
P_30                    all     0.3278
P_100                   all     0.2022
P_200                   all     0.1422
P_500                   all     0.0789
P_1000                  all     0.0478
```

eval_results_titles-desc.txt (~/Downloads/trec_eval-9.0.7) - gedit

```
runid                   all     indri
num_q                   all     150
num_ret                 all     150000
num_rel                 all     14013
num_rel_ret             all     7511
map                     all     0.2295
gm_map                  all     0.1343
Rprec                   all     0.2767
bpref                   all     0.2446
recip_rank              all     0.6977
iprec_at_recall_0.00    all     0.7396
iprec_at_recall_0.10    all     0.5123
iprec_at_recall_0.20    all     0.3754
iprec_at_recall_0.30    all     0.3079
iprec_at_recall_0.40    all     0.2448
iprec_at_recall_0.50    all     0.2007
iprec_at_recall_0.60    all     0.1544
iprec_at_recall_0.70    all     0.1107
iprec_at_recall_0.80    all     0.0721
iprec_at_recall_0.90    all     0.0437
iprec_at_recall_1.00    all     0.0199
P_5                     all     0.4827
P_10                    all     0.4347
P_15                    all     0.4098
P_20                    all     0.3803
P_30                    all     0.3367
P_100                   all     0.2063
P_200                   all     0.1421
P_500                   all     0.0814
P_1000                  all     0.0501
```

eval_results_titles-desc-narr.txt (~/Downloads/trec_eval-9.0.7) - gedit

```
runid                   all     indri
num_q                   all     150
num_ret                 all     150000
num_rel                 all     14013
num_rel_ret             all     7373
map                     all     0.2240
gm_map                  all     0.1259
Rprec                   all     0.2701
bpref                   all     0.2442
recip_rank              all     0.7025
iprec_at_recall_0.00    all     0.7367
iprec_at_recall_0.10    all     0.4846
iprec_at_recall_0.20    all     0.3889
iprec_at_recall_0.30    all     0.3140
iprec_at_recall_0.40    all     0.2500
iprec_at_recall_0.50    all     0.1965
iprec_at_recall_0.60    all     0.1469
iprec_at_recall_0.70    all     0.1069
iprec_at_recall_0.80    all     0.0617
iprec_at_recall_0.90    all     0.0271
iprec_at_recall_1.00    all     0.0083
P_5                     all     0.4800
P_10                    all     0.4353
P_15                    all     0.4018
P_20                    all     0.3783
P_30                    all     0.3291
P_100                   all     0.2025
P_200                   all     0.1436
P_500                   all     0.0810
P_1000                  all     0.0492
```

The metrics we are interested in are the below:

Metric	title	title+desc	title+desc+narr
Mean Average Precicion	22.17%	22.95%	22.4%
R-Precision	26.91%	27.67%	27.01%
Precision@10	42.73%	43.67%	43.33%

While observing the results we can see that our metrics dont vary so much across the three fields. Obviously the queries containing both the title and the desc give the best results (based on the metrics above). While adding the desc clearly boosts our results, adding the narr field too doesnt improve our search. This indicates that adding the desc field provides us with extra information but adding the narr field too comes to surplus. Let's look at the arithmetic mean of the Average Precision or MAP (mean of AP at the different values of Recall or area below the curve Precision - Recall). As we can see its between 22-23% for the first 3 cases. Then we examine the R-precision or the percentage $\frac{r}{R}$ where **r** is the number of our relevant results of a query until the first **R** texts where **R** is the number of relevant texts for the query. This metric has also not so good results being between 26-27%. Finally, regarding the Precision@10 we notice that it greatly higher than our previous two metrics. The results of the MAP and R-precision indicate that our retrieval quality isn't so good while the results of the Precision@10 indicate that even for the first 10 texts, 4 or 5 max texts are relevant with our search. In the next part we examine the query expansion with Relevance Feedback.

Part B

As mentioned before,in this part of the project, we use query expansion. Moreover, we acquire the 20-most frequent terms on the 15 most relevant texts of our query (based on the results of the Relevance Feedback). Apart from that we use a thesaurus to produce synonyms of those terms and add them to our improved query too.To do this, we must first examine our results to get the 15 most relevant texts/titles(files: **"results-titles.trec"** and **"results-titles-desc.trec"**). The function responsible for that is the following:

```
def get_top15(filename):
    #Open file with results
    f = open(filename+".trec","r")
    lines = f.readlines()
    previous = ""
    num_queries = []
    final ={}
    counter = 0
    #Read every line and check for a new number query
    for i in range(0,len(lines)):
        line = lines[i]
        line_values = line.split()
        if(line_values[0] != previous):
            num_queries.append(line_values[0])
            previous = line_values[0]
            #New query found so start keeping best 15
            start = True
        if(start):
            temp = []
            for k in range(0,15):
                temp.append(lines[i].split()[2])
                i = i+1
            start =  False
            final[num_queries[counter]] = temp
            counter = counter +1
    return final
```

After we have found the 15 most relevant titles, we must find the corpus its title belongs to. To do this we built a lexicon (stored as a json file) so that we have we have a correspondence between the texts (titles) and the corpus they belong to. The file containing them is the **paths.json** and its necessary for the **partB.py** to run (must be in the same folder). In case we don't have the lexicon then the function searches all the collections to find the right one and then searches the right text (the use of the lexicon is highly recommended becasue it's a lot faster).

After we have found the right collection the article belongs to all that remains is for us to open it and extract the text. Once we have extracted the text , we tokenize it and we use the library **Counter** to keep track of the frequency of its term in the text.

```
#Function to retrieve text from the file obtained before
def retrieve_text(filename,title):
    text = ""
    #Pattern for html and etc tags
    cleanTags = re.compile('<.*?>')
    #Open and read file
    with codecs.open(filename,"r",encoding="latin-1") as data:
        lines = data.readlines()
        for i in range(0,len(lines)):
```

```
            line = lines[i]
            #Find the right section of our file and gather only the <TEXT> part
            if title in line:
                i = i +1
                while("<TEXT>" not in lines[i]):
                    i=i+1
                i = i+1
                while("</TEXT>" not in lines[i]):
                    #Remove html tags and dont add empty or newlines
                    tempLine = re.sub(cleanTags,"",lines[i])
                    if tempLine not in ["","\n"]:
                        text = text + tempLine
                    i=i+1
    #Remove punctuation
    out = text.translate(text.maketrans('','',string.punctuation))
    return out
#Create dictionary of tokenized words and their frequency
def make_dict(text):
    ks = krovetz.PyKrovetzStemmer()
    #Use nltk tokenizer to get every word in the text
    lista = nltk.word_tokenize(text)
    lista = [ks.stem(word) for word in lista]
    counter = Counter(lista)
    #Create stopwords array to dismiss them later
    stopwords = nltk.corpus.stopwords.words('english')
    stopwords.extend(list(string.ascii_lowercase))
    stopwords.extend([word.capitalize() for word in stopwords])
    pops = set(stopwords).intersection(counter.keys())
    for i in pops:
        counter.pop(i)
    most_common = sorted(counter.items(),key = lambda pair: (-pair[1],pair[0]))
    #Return only first 50 words
    return most_common[0:50]
```

As we can see in the above code our function returns the 50 (not 20) most frequent terms in the 15 most relevant texts that were retrieved. The reason behind this is that we want to avoid cases of double-terms when we proceed to query expansion. Finally, all that's left to do is the query expansion while avoiding the double effect mentioned before and cases of vehicle-vehicles (by using **Krobetz Stemmer**). Our improved queries are then built and ready to be tested. The function responsible for all those things mentioned is the below:

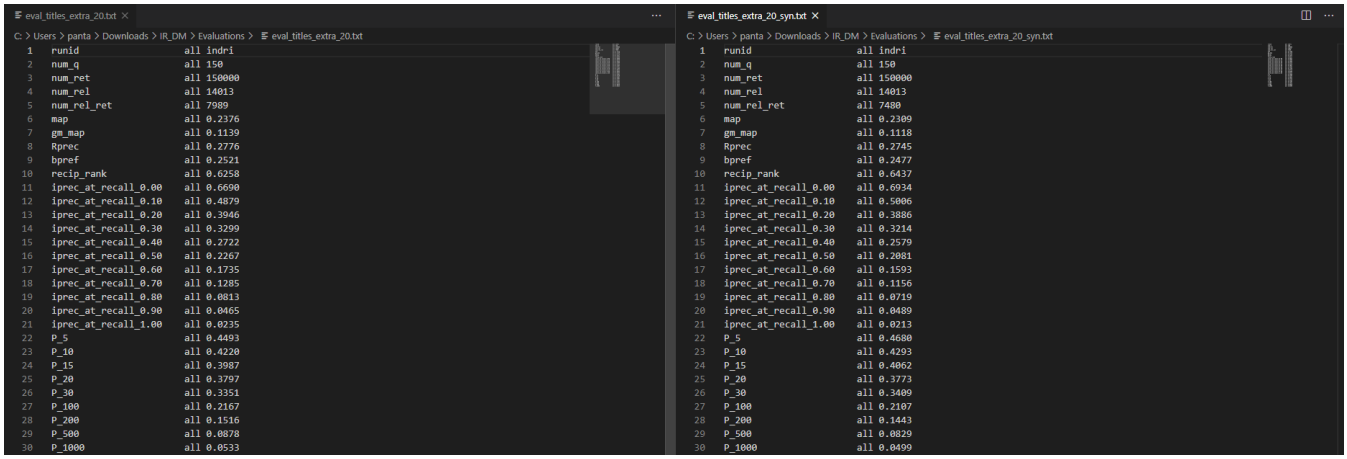
```
def create_improved_queries(json_file,originalQueries,output,weight):
    #Data extraction from json file
    f = open(json_file+".json",'r')
    data = json.load(f)
    f.close()
    keys = sorted(data.keys())
    currPath  = os.getcwd()
    if not os.path.exists("Improved Queries"):
        os.mkdir("Improved Queries")
    os.chdir(currPath + "/Improved Queries")
    #Create dictionary for current (old) queries
    count = 301
    queries = {}
    originalQueries = remove_pun(originalQueries)
    for q in originalQueries:
        queries[str(count)] = q
        count+=1
```

```
    #Create improved queries
    improvedQueries = []
    for key in keys:
        temp = ""
        items = data[key]
        #Count to keep track of number of iterations (we want top 20 only)
        tempCount = 0
        for item in items:
            #Check for duplicates
            if item[0].lower() not in queries[key].lower() and tempCount < 20:
                temp = temp + " " + item[0]
                tempCount +=1
        #Build new improved query
        improvedQueries.append(temp)
    write_queries(originalQueries,output+"extra-20-queries",improvedQueries,weight)
    queries_extra = combine_texts(originalQueries,improvedQueries)
    tempCount = 0
    #Create synonyms queries
    synQueries =[]
    ks = krovetz.PyKrovetzStemmer()
    for query in queries_extra:
        temp = ""
        tokenized = nltk.word_tokenize(query)
        words = [ks.stem(token).lower() for token in tokenized]
        for word in query.split():
            synsets = wn.synsets(word)
            count = 0
            for syn in synsets:
                for l in syn.lemmas():
                    if l.name().lower() not in temp.lower() and count !=2 and "_" not
in l.name() and l.name().lower() not in words:
                        temp = temp + " " + l.name()
                        count += 1
        synQueries.append(temp)
    synQueries = remove_pun(synQueries)
    synQueries = combine_texts(improvedQueries,synQueries)
    write_queries(originalQueries,output+"extra-20-syn-queries",synQueries,weight)
    os.chdir(currPath)
```

Αφού δημιουργηθούν τα queries χρησιμοποιούμε την μηχανή του indri lemur ώστε να ξανακάνουμε την αναζήτηση των information need μας και συγκρίνουμε τα αποτελέσματα που θα εξάγουμε με το **trec_eval**. Στις παρακάτω εικόνες φαίνεται και το evaluation των καινούριων queries (4 σε αριθμό). Η $1^{η}$ εικόνα δείχνει το evaluation στα βελτιωμένα queries (αριστερά: έξτρα όροι, δεξιά: έξτρα όροι + συνώνυμα) έχοντας ως βάση τα αποτελέσματα των queries που είχαν μόνο τον τίτλο ενώ η $2^{η}$ είναι με βάση τα αποτελέσματα των τίτλων και των περιγραφών (έχοντας ως βάση εννοούμε ότι χρησιμοποιήθηκαν εκείνα τα αποτελέσματα για την εφαρμογή της ανάδρασης σχετικότητας).

After we have created the improved queries we use the indri search engine to rerun our information need and then compare our results with **trec_eval**. The below images represent the evaluation for each method in the corresponding field. The first image shows the evaluation of the improved queries (**left**: extra terms, **right**: extra terms and synonyms) based only on the "titles-only" queries while on the second image the improved queries are based on the "titles-and-desc" queries.

***extra:** most frequent



In the following matrix we see the metrics we are interested in:

Metric	titles + extra20	titles + extra20 + syn	titles + desc + extra20	titles + desc + extra20 + syn
Mean Average Precicion	23.76 %	23.09 %	20.04 %	21.51 %
R-Precision	27.76 %	27.45 %	23.71 %	26.68 %
Precision@10	42.2 %	42.93 %	39.8 %	43.47 %

Based on the results above we can come to some conclusions. The extra terms (most frequent ones) and the synonyms applied to the "titles-only" original queries improved by a small percentage the MAP while the "titles-and-desc" decreased it by a lot. This can be explained by the length of our query which in the latter case is extremely large. In the R-Precision we observe the same things as the MAP. Moreover, on the "titles-only" with teh extra terms we see our best score. Finally, for the Precision@10 (which was explained in the part A of this project), we once again see simillar percentages (between 40-43.5). As we can understand our improved queries, which used for the Relevance Feedback only the titles (and not the desc too), improved our MAP and R-Precision percentages and kept Precision@10 to the same levels (realistically we had a slight drop). On the other hand by using both the titles and the desc for the Relevance Feedback we dropped our metrics.