
Information Retrieval and Text Mining: Assignment 2

Problem 1.*Recall*

postings: The postings list contains each token in each document, just as the TagTokenizer decoded it. Each posting is a triple: (word, document, position). You can see this data by using `dump-index`¹.

stemmedPostings: This is the same data as the postings file, except all words have also been stemmed with the Porter stemmer.

1. Query, “mountain climbing” editing the last part of the query as follows:
 - (a) `feature:dirichlet(#extents:climb:part=stemmedPostings()))`
 - (b) `feature:dirichlet(#extents:climb:part=postings()))`
 - (c) `feature:dirichlet(#extents:climbing:part=postings()))`
 - (d) `feature:dirichlet(#extents:climbing:part=stemmedPostings()))`

Compare and explain different results for each of the above four different queries.

2. As you learned in class, stemming words like “operating system” with Porter stemmer will always be a problem for retrieval ². This is because “operating”, “operative”, “operational” etc all are stemmed to “oper” by Porter stemmer. Confirm this in Galago.

You are given only stemmedPostings and postings. Can you overcome this problem using Galago Query language? If yes, How? If not, why not?

For another example of use of Porter stemmer decreasing the effectiveness of the query, check what happens with “tooth paste”.

3. Think of 3 more examples where using Porter stemmer would decrease effectiveness of query?
4. Think of 3 examples where using Porter stemmer might increase effectiveness of query?

Confirm your answers for above with Galago.

5. True or False?
 - (a) In a Boolean retrieval system, stemming never lowers precision³.
 - (b) In a Boolean retrieval system, stemming never lowers recall⁴.

Demonstrate your answer with a new query in your Boolean retrieval system. Note the query and the documents retrieved.

Problem 2.

¹<http://www.galagosearch.org/quick-start.html>

²<http://www.ims.uni-stuttgart.de/lehre/teaching/2011-WS/ir/pdfnew/02voc.pdf>

³precision is the number of relevant documents returned by your system divided by the total number of documents returned by your system

⁴recall is the number of relevant documents returned by your systems, divided by the number of relevant documents in the corpus

In this exercise we are going to extend our Boolean information retrieval system to support phrase queries of arbitrary length. Our query language already supports phrase queries in its syntax. So you can try something like:

```
Query: "new york"
Traceback (most recent call last):
  File "./query.py", line 162, in <module>
    main(sys.argv[1])
  File "./query.py", line 152, in main
    print parseQueryString(ir,query)
  File "./query.py", line 104, in parseQueryString
    return parseQuery(ir,reversed(list(tokenize(string))))
  File "./query.py", line 124, in parseQuery
    stack.append(ir.Qphrase(token))
AttributeError: IRSystem instance has no attribute 'Qphrase'
```

It fails because your implementation of `IRSystem` does not have a method `Qphrase`. `Qphrase` should expect a list of terms and return a list of all documents where the terms occur consecutively. Have a look at the slides⁵ and implement it!

Problem 3. (optional)

Implement Wildcard queries! You might use the B-tree or permuterm approach (i.e. you might just support queries of the form `term*` and `*term` or general Wildcard queries like `ter*m`).

Problem 4.

Until now our system returns sorted lists usually, search engines rank their results by relevance. In class you have seen one very famous approach to rank results: tfidf-ranking. Implement a class `IRSystem` that reads a corpus file and has a method which returns the n best scored documents for a query (now just a list of terms not a Boolean expression). To make your system efficient you should use a max heap and reduce the calculation to the documents that contain all the query terms. Hint: you might check if there exists a max heap implementation for Python. Could your Boolean system be useful at some point? Evaluate your system by analysing the rankings of - at least - two queries. Why is it important to normalize document vectors, but not to normalize queries? Could you give an example where document normalization is important?

```
class IRSystem:
    @param f : a file object
    def __init__(self,f):
        pass

    @param query : list of terms
    @param n : natural number
    def nbest(self,query,n):
        pass
```

Exercise date: Tuesday, November 22, 2011, 14:00, room 3.8!

⁵<http://www.ims.uni-stuttgart.de/lehre/teaching/2011-WS/ir/pdfnew/02voc.pdf>