Session 3: User Relevance Feedback

Yimin Pan
Niebo Zhang

November 2019

1 We will, we will Rocchio you

In this section we are going to make detailed explanation on the implementation done in `Roccio.py`, which we took `IndexFilePreprocess.py` as template. The first was to understand what `IndexFilePreprocess.py` does which was not a big deal since the code was pretty comprehensible for the reader.

The second part was to modify the function `toTFIDF` which we implemented in last session, the main issue here is that we want to change the code so instead of lists we are using dictionaries, making the later processes easier to code. Something that should be mentioned is that we avoided normalizing the content of the array because it doesn't affect the result so we decided not to do useless stuff, and worked directly with the weigth of each term.

The following part was to understand how the Rocchio's algorithm work so we can make the implementation. For a better implementation we used a dictionary named `query_dict` which contains all the words of the i-th iteration query and obviously its importance in the search, which is computed each time by the algorithm given:

$$\text{Query'} = \alpha \cdot \text{Query} + \beta \cdot \frac{d_1 + \cdots + d_k}{k}$$

Finally we choose the R more important terms and merge them again into the list `query` with the specific format $(term)$^$(weight)$.

For the correct functionality of the code we did several merges between lists, this is where using dictionaries are more efficient than working with lists.

The cost of merging two vectors of size n and m is $O(n+m)$ and we keep `nhits`(k) documents from the response. If the size of all the documents vector is n then the cost of merging the remaining documents should be $O(kn)$. On the other hand, maps are usually implemented with binary search trees, so the cost of search and insert are lower than a list ($O(log(n))$ vs $O(n)$). Merging two maps, then, could cost $m * log(n)$ which in theory is higher than $O(n+m)$, but from the previous sessions, we know that most of the terms are repeated ($m << n$), so in the practice merging maps is faster than lists. Furthermore, when using maps we can suppose not sorted lists, so we can modify the `document_term_vector` function such that it return not ordered lists, this way we are avoiding an extra work of sorting two lists.

We didn't find any difficulty in terms of implementation, given that we could take advantatge of most part of the code from the previous section, then basically was to apply the formula.

2 Experimentation

In our experiment we won't modify the initial queries (*football basketball*) in order to play a little bit with the parameters, and find out wheter it has any effect on the result or not. `20_newsgroup` text corpus from the previous session will be indexed to elastic search for the testing. The reason is that documents from `20_newsgroup` usually refers to very specific topics, by constrast novels are extensive texts, and consequently have a high probability of matching with the word of the query. At each experiment we will modify only a single parameter.

1. $nround = 5$, $\alpha = 1$, $\beta = 0.2$, $k = 5$, $R = 5$

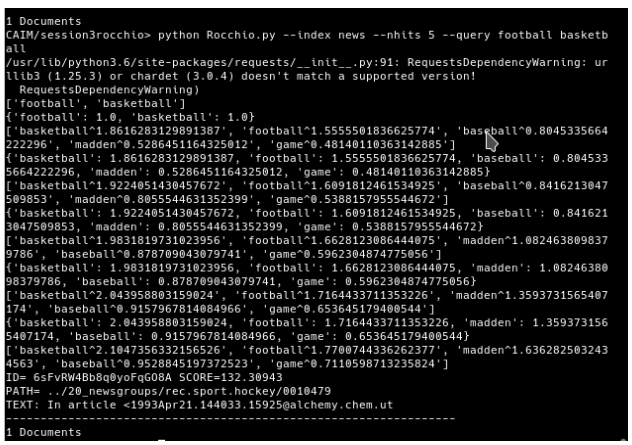
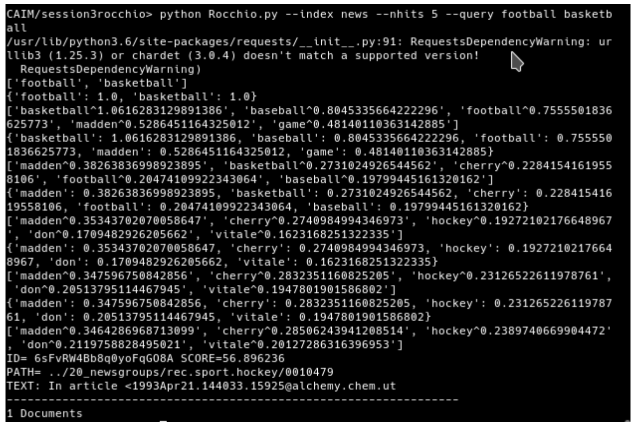
   

   Figure 1: Experiment 1

   When α is high seems that the resulting query is highly related to the initial query, we still have basketball and football, although it has added other words (baseball, games, ...) that are commonly related to the ones we queried.

2. $nround = 5$, $\alpha = 0.2$, $\beta = 1$, $k = 5$, $R = 5$

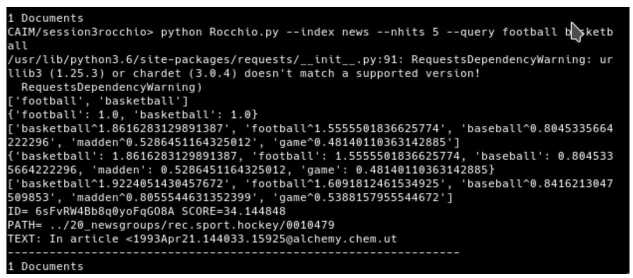


```
CAIM/session3rocchio> python Rocchio.py --index news --nhits 5 --query football basketb
all
/usr/lib/python3.6/site-packages/requests/__init__.py:91: RequestsDependencyWarning: ur
llib3 (1.25.3) or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
['football', 'basketball']
{'football': 1.0, 'basketball': 1.0}
['basketball^1.0616283129891386', 'baseball^0.8045335664222296', 'football^0.7555501836
625773', 'madden^0.5286451164325012', 'game^0.48140110363142885']
{'basketball': 1.0616283129891386, 'baseball': 0.8045335664222296, 'football': 0.755550
1836625773, 'madden': 0.5286451164325012, 'game': 0.48140110363142885}
['madden^0.38263836998923895', 'basketball^0.2731024926544562', 'cherry^0.2284154161955
8106', 'football^0.20474109922343064', 'baseball^0.19799445161320162']
{'madden': 0.38263836998923895, 'basketball': 0.2731024926544562, 'cherry': 0.228415416
19558106, 'football': 0.20474109922343064, 'baseball': 0.19799445161320162}
['madden^0.35343702070058647', 'cherry^0.2740984994346973', 'hockey^0.19272102176648967
', 'don^0.1709482926205662', 'vitale^0.1623168251322335']
{'madden': 0.35343702070058647, 'cherry': 0.2740984994346973, 'hockey': 0.1927210217664
8967, 'don': 0.1709482926205662, 'vitale': 0.1623168251322335}
['madden^0.347596750842856', 'cherry^0.2832351160825205', 'hockey^0.23126522611978761',
 'don^0.20513795114467945', 'vitale^0.1947801901586802']
{'madden': 0.347596750842856, 'cherry': 0.2832351160825205, 'hockey': 0.23126522611978761, 'don': 0.20513795114467945, 'vitale': 0.1947801901586802}
['madden^0.3464286968713099', 'cherry^0.28506243941208514', 'hockey^0.2389740669904472'
, 'don^0.2119758828495021', 'vitale^0.20127286316396953']
ID= 6sFvRW4Bb8q0yoFqGO8A SCORE=56.896236
PATH= ../20_newsgroups/rec.sport.hockey/0010479
TEXT: In article <1993Apr21.144033.15925@alchemy.chem.ut
-----------------------------------------------------------------
1 Documents
```

Figure 2: Experiment 2

Now, as we decreased the α, the resulting query show terms that have nothing to do with the initial query. Like, *cherry* or *don*. Observe that we even lost the initial query words!

3. $nround = 2$, $\alpha = 1$, $\beta = 1$, $k = 5$, $R = 5$

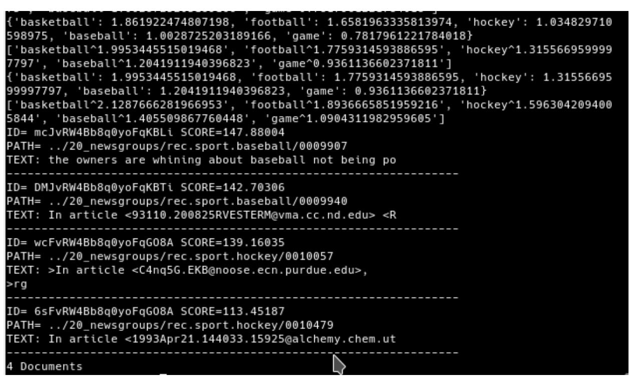


```
1 Documents
CAIM/session3rocchio> python Rocchio.py --index news --nhits 5 --query football basketb
all
/usr/lib/python3.6/site-packages/requests/__init__.py:91: RequestsDependencyWarning: ur
llib3 (1.25.3) or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
['football', 'basketball']
{'football': 1.0, 'basketball': 1.0}
['basketball^1.8616283129891387', 'football^1.5555501836625774', 'baseball^0.8045335664
222296', 'madden^0.5286451164325012', 'game^0.48140110363142885']
{'basketball': 1.8616283129891387, 'football': 1.5555501836625774, 'baseball': 0.804533
5664222296, 'madden': 0.5286451164325012, 'game': 0.48140110363142885}
['basketball^1.9224051430457672', 'football^1.6091812461534925', 'baseball^0.8416213047
509853', 'madden^0.8055544631352399', 'game^0.5388157955544672']
ID= 6sFvRW4Bb8q0yoFqGO8A SCORE=34.144848
PATH= ../20_newsgroups/rec.sport.hockey/0010479
TEXT: In article <1993Apr21.144033.15925@alchemy.chem.ut
-----------------------------------------------------------------
1 Documents
```

Figure 3: Experiment 3

We have less iteration, so the result is more dependant to the firsts loops.

4. $nround = 5$, $\alpha = 1$, $\beta = 1$, $k = 10$, $R = 5$



```
{'basketball': 1.861922474807198, 'football': 1.6581963335813974, 'hockey': 1.034829710
598975, 'baseball': 1.0028725203189166, 'game': 0.7817961221784018}
['basketball^1.9953445515019468', 'football^1.7759314593886595', 'hockey^1.315566959999
7797', 'baseball^1.2041911940396823', 'game^0.9361136602371811']
{'basketball': 1.9953445515019468, 'football': 1.7759314593886595, 'hockey': 1.31556695
99997797, 'baseball': 1.2041911940396823, 'game': 0.9361136602371811}
['basketball^2.1287666281966953', 'football^1.8936665851959216', 'hockey^1.596304209400
5844', 'baseball^1.405509867760448', 'game^1.0904311982959605']
ID= mcJvRW4Bb8q0yoFqKBLi SCORE=147.88004
PATH= ../20_newsgroups/rec.sport.baseball/0009907
TEXT: the owners are whining about baseball not being po
-----------------------------------------------------------------
ID= DMJvRW4Bb8q0yoFqKBTi SCORE=142.70306
PATH= ../20_newsgroups/rec.sport.baseball/0009940
TEXT: In article <93110.200825RVESTERM@vma.cc.nd.edu> <R
-----------------------------------------------------------------
ID= wcFvRW4Bb8q0yoFqGO8A SCORE=139.16035
PATH= ../20_newsgroups/rec.sport.hockey/0010057
TEXT: >In article <C4nq5G.EKB@noose.ecn.purdue.edu>,
>rg
-----------------------------------------------------------------
ID= 6sFvRW4Bb8q0yoFqGO8A SCORE=113.45187
PATH= ../20_newsgroups/rec.sport.hockey/0010479
TEXT: In article <1993Apr21.144033.15925@alchemy.chem.ut
-----------------------------------------------------------------
4 Documents
```

Figure 4: Experiment 4

We can observe that the number of documents found increased from 1 to 4.

5. $nround = 5$, $\alpha = 1$, $\beta = 1$, $k = 5$, $R = 10$

Figure 5: Experiment 5

When R is high enough, elastic search tend to find less documents.

3 Conclusion

To sum up we are going to talk about the effects of touching each parameter and what it can mean.

1. *nRound* determines the number of iteration of applied to the Rocchio's rule. So, its logical to think that its strongly related to the influence of α and β to the new query. Higher iteration, more influence.

2. When we look at the formula, it's not hard to find out that α is applied to the actual query. At the first iteration, actual query = query from the user, so α determines the weight of the input query. It is obvious that with α small enough we will lose the input query.

3. β, the other way round, decides how the query created from `TFIDF` of the documents affects the new query.

4. A corollary from the previous two items is that the weight of α and β are not independent, but strongly correlated. For example to increase the weight of the input query we can increase α or decrease β.

5. The variable k trims the number of documents to calculate `TFIDF`, when this number is high, we retrieve more documents at the end, this means we have more false positives. Consequently more recall.

6. With the variable R we limit the maximum size of the new query at each iteration, when the value of R is low, we have less false positives, or in another words, more true positives. As consequence we have higher precision.

At the end, it's a trade off between recall and precision. Which is hard to say wich one is better, it will all depend on the context and which kind of retrieval we are searching of.