CS 6501: Information Retrieval

MP2: Retrieval Evaluations

Yiwei Fang

yf5kq

October 31, 2018

1. **Copy and paste your implementation of each evaluation function into your report, together with the corresponding final MAP/P@10/MRR/NDCG@10 performance you get from each ranking function. (40pts + 20pts)**

- **MAP:**

```
double AvgPrec(String query, String docString) {
        ArrayList<ResultDoc> results = _searcher.search(query).getDocs();
        if (results.size() == 0)
                return 0; // no result returned

        HashSet<String> relDocs = new
HashSet<String>(Arrays.asList(docString.trim().split("\\s+")));
        int i = 1;
        double avgp = 0.0;
        double numRel = 0;
        double sumRel = 0;

        System.out.println("\nQuery: " + query);
        for (ResultDoc rdoc : results) {
                if (relDocs.contains(rdoc.title())) {
                        //how to accumulate average precision (avgp) when we
encounter a relevant document
                        numRel ++;
                        sumRel += numRel/i;
                        System.out.print("  ");
                } else {
                        //how to accumulate average precision (avgp) when we
encounter an irrelevant document
                        System.out.print("X ");
                }
                System.out.println(i + ". " + rdoc.title());

                ++i;
        }

        //compute average precision here
        // avgp = ?
        avgp = numRel != 0 ? sumRel/numRel : 0;
        System.out.println("Average Precision: " + avgp);
        return avgp;
}
```

- **P@10:**

```
double Prec(String query, String docString, int k) {
        double p_k = 0;
        //your code for computing precision at K here

        ArrayList<ResultDoc> results = _searcher.search(query).getDocs();
        if (results.size() == 0)
                return 0; // no result returned

        HashSet<String> relDocs = new
HashSet<String>(Arrays.asList(docString.trim().split("\\s+")));
```

```
		int i = 1;
		double numRel = 0;

		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				numRel ++;
			}
			++i;
			if (i == k+1)
				break;
		}

		//compute p_k
		p_k = numRel/k;
		System.out.println("p_k: " + p_k);
		return p_k;
	}
```

- **MRR:**

```
	double RR(String query, String docString) {
		double rr = 0;
		//your code for computing Reciprocal Rank here

		ArrayList<ResultDoc> results = _searcher.search(query).getDocs();
		if (results.size() == 0)
			return 0; // no result returned

		HashSet<String> relDocs = new
HashSet<String>(Arrays.asList(docString.trim().split("\\s+")));
		int i = 1;

		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				rr = (double)1/i;
				break;
			}
			++i;
		}


		System.out.println("rr: " + rr);
		return rr;
	}
```

- **NDCG:**

```
	double NDCG(String query, String docString, int k) {
		double ndcg = 0;
		double pcg = 0;
		double idcg = 0;
		int numRel = 0;

		//your code for computing Normalized Discounted Cumulative Gain here
```

```
        ArrayList<ResultDoc> results = _searcher.search(query).getDocs();
        if (results.size() == 0)
            return 0; // no result returned

        HashSet<String> relDocs = new
HashSet<String>(Arrays.asList(docString.trim().split("\\s+")));
        int i = 1;

        for (ResultDoc rdoc : results) {
            if (relDocs.contains(rdoc.title())) {
                double log = Math.log(1+i) / Math.log(2);
                pcg += (Math.pow(2, 4) - 1)/log;
            }
            if (i == k)
                break;
            ++i;
        }

        numRel = relDocs.size() > k ? k : relDocs.size();
        for (int j = 1; j <= numRel; j++) {
            double ilog = Math.log(1+j) / Math.log(2);
            idcg += (Math.pow(2, 4) - 1)/ilog;
        }

        ndcg = idcg != 0 ? pcg/idcg : 0;
        return ndcg;
    }
```

- **BM25:**

```
MAP: 0.4370747384183165

P@10: 0.34623655913978496

MRR: 0.6740525521696632

NDCG: 0.42603032215453884
```

- **TFIDF:**

```
MAP: 0.44135079497398244

P@10: 0.35053763440860214

MRR: 0.6899569381353063

NDCG: 0.43123592047089643
```

2. **In edu.virginia.cs.index.SpecialAnalyzer.java, we defined a special document analyzer to process the document/query for retrieval purpose. Basically, we built up a pipeline with filters of LowerCaseFilter, LengthFilter, StopFilter, and PorterStemFilter. Please disable some of the filters, e.g., without stopword removal or stemming, and test the new analyzer with the BM25 model. What is your conclusion about the effect of document analyzer on retrieval effectiveness? (20pts)** ***Note: this analyzer has to be used in both indexing time and query time!***

- LowerCaseFilter:

```
MAP: 0.4370747384183165

P@10: 0.34623655913978496

MRR: 0.6740525521696632

NDCG: 0.42603032215453884
```

- LengthFilter:

```
MAP: 0.4384336468284397

P@10: 0.3483870967741936

MRR: 0.6874979963935085

NDCG: 0.4300451674422877
```

- StopFilter:

```
MAP: 0.4357411644894412

P@10: 0.3344086021505377

MRR: 0.6735338428938381

NDCG: 0.4129613899798967
```

- PorterStemFilter:

```
MAP: 0.39226359602497685

P@10: 0.2720430107526882

MRR: 0.6179541773971634

NDCG: 0.3420401781424409
```

From the above statistic we can conclude that the PorterStemFilter will have the largest effect of document analyzer on retrieval effectiveness. Removal of other filters, like Stopwords filter, does not have too much effect because the BM25 have IDF functional part to penalize the common words in global scale. Without PorterStemFilter, for instance, “project” and “projects” are seemed as two independent words, so that it will largely effect the search documents results.

3. **In question 1, we compared the ranking model BM25 with TFIDF only by the mean value of the four evaluation metrics. As we already know that statistical test is necessary when we only have a small evaluation data set (93 queries in our case). Let's compute and report the p-value from <u>paired t-test</u> and <u>Wilcoxon signed-rank test</u> for the comparison over all four metrics. Based on your statistical test results, which is a better ranking algorithm? (20pts)** ***Note: you do not need to implement the calculation of those tests. You can find any Java/Python/Matlab implementation for this purpose, and just prepare the required input for it.***

```
PairTest_mAP: 0.5791233144269852

PairTest_P@K: 0.5889497122858898

PairTest_MRR: 0.31574187264274456

PairTest_NDCG: 0.4922986878125891

wilcoxonSignedRank_mAP: 0.7302120990522221

wilcoxonSignedRank_P@K: 0.033778768966735054

wilcoxonSignedRank_MRR: 2.1634161361057264E-5

wilcoxonSignedRank_NDCG: 0.9724909293126439
```

- We seek to reject the null hypothesis and the small p values are good. Paired T Test on both BM25 and TFIDF system do not have big different performance on four evaluation matrix. From Wilcoxon signed-rank test, on P@K and MRR evaluation matrix have a very small p values (typically ≤ 0.05) indicates strong evidence against the null hypothesis, so we can reject the null hypothesis. Pair test indicates that these two systems are not distinguishable among evaluation matrix based on 96 queries, while Wilcoxon signed-rank test have significantly emphasis on the difference between BM25 and TFIDF systems according to P@K and MRR. Maybe it because of the way that these two evaluation matrix computing the evaluation, effectiveness of different metrics are varying.