Chengjun Yuan Information Retrieval MP3

1

Implementation of Boolean Models.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
	if (termFreq > 0) return 1;
	else return 0;
}
```

Implementation of TF-IDF dot product.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
	float tf = (float) (1.0 + Math.log10(termFreq));
	float idf = (float) Math.log10((stats.getNumberOfDocuments() + 1.0) /
stats.getDocFreq());
	return tf * idf;
}
```

Implementation of Okapi BM25.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
	float k1 = 1.5f;
	float k2 = 750.0f;
	float b = 1.0f;
	float queryFreq = 1.0f;
	float p1 = (float) Math.log((stats.getNumberOfDocuments() - stats.getDocFreq()
+ 0.5) / (stats.getDocFreq() + 0.5));
	float p2 = (float) (k1 + 1) * termFreq / (k1 * (1 - b + b * docLength /
stats.getAvgFieldLength()) + termFreq);
	float p3 = (float) (k2 + 1) * queryFreq / (k2 + queryFreq);
	return p1 * p2 *p3;
}
```

Implementation of Pivoted Length Normalization.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
	float s = 0.75f;
	float queryFreq = 1.0f;
	float p1 = (float) (1 + Math.log(1 + Math.log(termFreq))) / (1 - s + s *
docLength / stats.getAvgFieldLength());
	float p2 = (float) Math.log((stats.getNumberOfDocuments() + 1) /
stats.getDocFreq()) * queryFreq;
	return p1 * p2;
}
```

Implementation of Jelinek-Mercer Smoothing.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
```

```
      float p_s = (float) ((1 - lamda) * termFreq / docLength + lamda *
model.computeProbability(stats));
      return (float) Math.log10(p_s / lamda / model.computeProbability(stats));
}

// add the rest part in searcher.java
QueryParser parser = new QueryParser(Version.LUCENE_46, field, analyzer);
            if(sim instanceof JelinekMercer) {
             double lamda = ((JelinekMercer) sim).returnLamda();
             for(ScoreDoc hit : hits){
                   hit.score += queryLength * (float) Math.log10(lamda);
                }
             Arrays.sort(hits, new Comparator<ScoreDoc>() {
                   public int compare(ScoreDoc a, ScoreDoc b) {
                         if (a.score < b.score) return 1;
                         else if (a.score == b.score) return 0;
                         else return -1;
                   }
             });
            }
```

Implementation of Dirichlet Prior Smoothing.

```
protected float score(BasicStats stats, float termFreq, float docLength) {
      float alpha_d = mu / (mu + docLength);
      float p_s = (float) (termFreq + mu * model.computeProbability(stats)) /
(docLength + mu);
       return (float) Math.log10(p_s / alpha_d / model.computeProbability(stats));
}

// add the rest part in searcher.java
else if (sim instanceof DirichletPrior) {
             double mu = ((DirichletPrior) sim).returnMu();
             for(ScoreDoc hit : hits){
                   Document doc = indexSearcher.doc(hit.doc);
                   int docLength = 0;
                   String contents = doc.getField(field).toString();
                   String parsed = parser.parse(contents).toString();
                   docLength = parsed.split(" ").length;
                   double alpha_d = mu / (mu + docLength);
                   hit.score += queryLength * (float) Math.log10(alpha_d);
                }

             Arrays.sort(hits, new Comparator<ScoreDoc>() {
                   public int compare(ScoreDoc a, ScoreDoc b) {
                         if (a.score < b.score) return 1;
                         else if (a.score == b.score) return 0;
                         else return -1;
                   }
             });
            }
```

Implementation of AvgPrec.

```
private static double AvgPrec(String query, String docString) {
		ArrayList<ResultDoc> results = _searcher.search(query).getDocs();
		if (results.size() == 0)
			return 0;
		HashSet<String> relDocs = new
HashSet<String>(Arrays.asList(docString.split(" ")));
		int i = 1;
		double avgp = 0.0;
		double numRel = 0;
		System.out.println("\nQuery: " + query);
		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				numRel ++;
				avgp += numRel / i;
				System.out.print("  ");
			} else {
				System.out.print("X ");
			}
			System.out.println(i + ". " + rdoc.title());
			++i;
		}
		if (numRel > 0) avgp = avgp / relDocs.size();
		else avgp = 0.0;
		System.out.println("Average Precision: " + avgp);
		return avgp;
	}
```

Implementation of P@K.

```
private static double Prec(String query, String docString, int k) {
		double p_k = 0;
		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.split(" ")));
		int i = 0;
		double numRel = 0;
		System.out.println("\nQuery: " + query);
		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				numRel ++;
				System.out.print("  ");
			} else {
				System.out.print("X ");
			}
			System.out.println(i + ". " + rdoc.title());
			++i;
			if (i == k) break;
		}
		p_k = numRel / k;
		return p_k;
	}
```

Implementation of Reciprocal Rank.

```
private static double RR(String query, String docString) {
		double rr = 0;
		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.split(" ")));
		int i = 1;
		System.out.println("\nQuery: " + query);
		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				break;
			} else {
				System.out.print("X ");
			}
			System.out.println(i + ". " + rdoc.title());
			++i;
		}
		rr = 1.0 / i;
		return rr;
	}
```

Implementation of Normalized Discounted Cumulative Gain.

```
private static double NDCG(String query, String docString, int k) {
		double ndcg = 0;
		double idcg = 0.0;
		// compute idcg.
		int i = k;
		while (i > 0) {
			idcg += Math.log(2.0) / Math.log(1 + i);
			i --;
		}
		// compute ndcg.
		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.split(" ")));
		i = 1;
		System.out.println("\nQuery: " + query);
		for (ResultDoc rdoc : results) {
			if (relDocs.contains(rdoc.title())) {
				ndcg += Math.log(2.0) / Math.log(1 + i);
				System.out.print("  ");
			} else {
				System.out.print("X ");
			}
			System.out.println(i + ". " + rdoc.title());
			++i;
			if (i > k) break;
		}
		ndcg = ndcg / idcg;
		return ndcg;
```

```
        }
```

Performance results:

	MAP	P@10	MRR	NDCG@10
Boolean	0.2109	0.2882	0.5949	0.3267
TF-IDF	0.2549	0.3559	0.6795	0.3929
Okapi BM25	0.2177	0.3075	0.5938	0.3391
Pivoted Length Normalization	0.1574	0.2398	0.4359	0.2527
Jelinek-Mercer	0.2585	0.3419	0.6785	0.3885
Dirichlet Prior	0.1849	0.2366	0.5455	0.2708

2 Tune parameters in BM25 :

MAP		K1				
		1.2	1.3	1.4	1.5	1.6
b	0.75	**0.2594**	0.2574	0.2550	0.2538	0.2510
	0.8	0.2550	0.2530	0.2492	0.2473	0.2448
	0.85	0.2516	0.2464	0.2440	0.2417	0.2389
	0.9	0.2461	0.2420	0.2377	0.2352	0.2321
	1.0	0.2307	0.2254	0.2205	0.2177	0.2136
	1.1	0.2166	0.2095	0.2036	0.1981	0.1933

According to the MAP in the upper table, the optimal performance of BM25 is located at **b=0.75 and K1=1.2**. The variance of parameter k2 in BM25 has less influence on the MAP.

Tune parameters in Dirichlet Prior :

	μ						
	2000	2100	2200	2300	2400	2500	2600
MAP	**0.1910**	0.1891	0.1877	0.1865	0.1855	0.1849	0.1835

From this table, we can see that the performance of Dirichlet Prior smoothing increases as the parameter μ decrease. The optimal value of μ is **2000** in the range of (2000 – 3000).

3 reduce some filters and evaluate the influence on the performance of BM25.

	Non Disable	Disable LowerCaseFilter	Disable LengthFilter	Disable StopFilter	Disable StemmerFilter
MAP	0.2177	0.2177	0.2181	0.1402	0.1646
P@10	0.3075	0.3075	0.3065	0.1989	0.2333

MRR	0.5938	0.5938	0.5935	0.4557	0.5350
NDCG@10	0.3391	0.3391	0.3386	0.2210	0.2633

From the upper table, we can find that disabling or removing lowerCaseFilter or LengthFilter has little influence on the performance of retrieval. However, the stopFilter and stemmerFilter has very apparent effectiveness on retrieval of BM25. Their absence can highly decrease the precision of BM25 retrieval. This is because stopFilter can remove the most common but meaningless words which will be noise for the retrieval model. The stemmerFilter will derive the different words with the similar word-root to the same word, which increase the opportunities of finding the same words between queries and documents and increase the rank precision.

4 For analysis of TF-IDF dot-product model vs. Boolean dot-product model.

One query sentence is picked where TF-IDF performs much better than Boolean does.

Query	TF-IDF AP	Boolean AP
i would like information on the range of **static relays** suitable for use at **high switching** rates.	0.2719	0.0697

The first two return of TF-IDF are:

1	1455 design of **static** relays for signalling and control the advantages of **static** **relays** and the replacement of mechanical **relays** by semiconductor devices are discussed circuit diagrams are given
2	8652 **static** switching a transistor nor unit output only when all inputs absent is shown to be **suitable** as a basic element for building **switching** systems

The first two return of Boolean are:

X 1	776 computer **switching** with high power transistors a method for selecting the most **suitable** type of power transistor for particular **switching** applications is given
X 2	812 the **use** of silicon diodes in d c modulators and their applications to drift correctors for computing amplifiers advantages over the conventional relay modulator include good **high** frequency response low noise level low **switching** power requirements and an operational life limited only by the thermionic valves zero stability is comparable wite that of the conventional relay

By comparison with their first two outputs, we can see that TF-IDF considers the term frequency and use IDF to reduce the influence of common words, such as “use”. So the TF-IDF can retrieve the sentence with more frequency of important words as shown in upper table. But the Boolean method can only retrieve the sentences with the same words and does not discriminate their influence.

For analysis of TF-IDF vs. BM25 (with tuned parameters).

The 28th query is picked where the TF-IDF performs much better than BM25 does.

Query	TF-IDF AP	BM25
the **effect** of **small** distortions in the **surface** of a **cavity** **resonator**	0.4049	0.1801

The first two ranked query results of TF-IDF. The first one is relevant but the second one is irrelevant.

1	**effect** of **surface** roughness on losses in microwave **resonant** **cavities** measurements of value were made on a cylindrical mode **cavity** specially designed so that the internal **surface** could be treated in various ways eg by plating or polishing the magnitude of the discrepancies between the measured values and those calculated for a perfectly smooth **surface** depends on the dimensions of the **surface** irregularities and is very **small** when the **surface** unevenness is no greater than the skin depth
X 2	retardation **effects** caused by ferrite sample size on the frequency shift of a resonant **cavity** expressions are derived using perturbation theory for the frequency shift of a circularly polarized resonant microwave **cavity** due to insertion of a **small** ferrite sample

The first two ranked query results of BM25. Both of them are irrelevant.

X 1	the concept of heterogeneous surface impedance and its application to cylindrical **cavity** resonators formulae are developed relating the q factor and the resonant frequency of a **cavity** to its dimensions and the fourier components of the **surface** impedance function the cases of circumferential and axial heterogeneity are analysed in detail in general a unique value of **surface** impedance cannot apply to an unbounded periodic sheet unless the period is small compared with
X 2	retardation effects caused by ferrite sample size on the frequency shift of a resonant **cavity** expressions are derived using perturbation theory for the frequency shift of a circularly polarized resonant microwave cavity due to insertion of a small ferrite sample

From the upper comparison, since the BM25 penalize the length of long documents, so the long-length relevant document is not preferred by the BM25 method. So if the query has long-length relevant document, the TF-IDF outperforms BM25.

For analysis of BM25 vs. Dirichlet Prior (with tuned parameters).

The 19th query is picked where the BM25 performs much better than Dirichlet does

Query	TF-IDF AP	BM25
spherical **harmonic** **analysis** of the earths **magnetic** **field**	0.4749	0.1720

The first two ranked query results of BM25. Both of them are relevant.

1	rocket measurements of the **magnetic** **field** above new mexico absolute total intensity was measured in a rocket at heights up to and compared with the intensity predicted by spherical **harmonic** coefficients
2	a method for analysing values of the scalar **magnetic** intensity the method uses a series for the square of the scalar intensity the terms of which are obtained from the **spherical** harmonics generally applied to each component of the intensity those **magnetic** characteristics normally given by analysis of the vector intensity are obtained

The first two ranked query results of Dirichlet Prior. Both of them are irrelevant.

X 1	the possibility of electron total energy distribution **analysis** in a quasi **spherical** capacitor the energy distribution of electrons emitted at various angles from a plane disk shaped target at the centre of a **spherical** collector is determined from an **analysis** of electron trajectories the discrepancy between these and those for a **spherical** capacitor does not exceed one per cent the provision of an aperture in the collector for an electron gun results in a five per cent error
X 2	on the diffraction and reflection of waves and pulses by wedges and corners various problems arising in the theory of the excitation of a perfectly reflecting wedge or corner by a plane cylindrical or **spherical** wave are dealt with the incident wave is represented by a line source acoustic or electro**magnetic** parallel to the edge the **spherical** wave is emitted by an acoustic point source or by a hertz dipole with its axis parallel to the edge the case of an incident plane wave **field** is obtained as the limiting case for large distances of the source from the edge of the cylindrical or **spherical** wave excitation

According to the upper analysis, the BM25 prefers short length document since it penalize much on document length. Compared with BM25, the Dirichlet Prior prefer the document with the higher frequency of query terms. That is why the results from Dirichlet Prior have more query term frequency than those of BM25 as shown upper. Even though many words are highly repeated.

5 Here we adopt the document growth function (3.3.1 Formula 1 in Fang & Zhai 2005) to get the new retrieval function for Pivoted Normalization (PN) as shown below:

$$s(q,d) = \sum_{t \in D \cap Q} TF(C_t^D) \cdot C_t^Q \cdot weight(t) \frac{avdl + s}{avdl + |D| \cdot s}$$

Where $TF(C_t^D) = 1 + \ln(1 + \ln(C_t^D))$, $weight(t) = log \frac{N+1}{df(t)}$, and $avdl$ is the average length of document. The implementation codes are shown below:

```
protected float score(BasicStats stats, float termFreq, float docLength) {
	float s = 0.2f;
	float queryFreq = 1.0f;
	double k = 0.5;
	float p1 = (float) (1 + Math.log(1 + Math.log(termFreq)));
	float weight = (float) Math.log10((stats.getNumberOfDocuments() + 1) /
stats.getDocFreq());
	float p2 = queryFreq * weight;
```

```
        float p3 = (float) ((stats.getAvgFieldLength() + s) /
(stats.getAvgFieldLength() + docLength * s));
        return p1 * p2 * p3;
}
```
©

The results of two retrieval functions are listed below:

	New PN retrieval function 0 < s < 1		
	s = 0.2	s = 0.5	s = 0.8
MAP	0.2687	0.2604	0.2392

	Standard PN retrieval function 0 < s < 1		
	s = 0.2	s = 0.5	s = 0.8
MAP	0.2667	0.2290	0.1375

In this paper, three constraints are presented to improve the precision of retrieval. The first constraint is that adding one query term to a document must increase the score. The second one is that adding a non-query term to a document must decrease the score. And the last one is that the score due to adding a query term to a document must decrease as we add more and more documents. The comparison of upper two tables can show that the new PN retrieval function are more robust and less sensitive to the parameter s. What's more, the new function performs better with higher mean average precision than the standard one. This is because the new retrieval function is derived according to the three constraints.