1. Implementation

Copy and paste your implementation of each ranking algorithm, together with the corresponding final MAP/P@10/MRR/NDCG@10 performance you get from each ranking function. *Use the default parameter settings suggested here* (30pts)

Boolean Dot Product

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

Okapi BM25

```
protected float score(BasicStats stats, float termFreq, float docLength) {
    double k1 = 1.5; double k2 = 750; double b = 1;
    double N = stats.getNumberOfDocuments();
    double df = stats.getDocFreq();
    double n = stats.getAvgFieldLength();
    double t1 = (Math.log((N-df+0.5)/(df+0.5)));
    double t2 = ((k1+1)*termFreq)/(k1*(1-b+b*(docLength/n))+termFreq);
    double t3 = ((k2+1)*1)/(k2+1);
    return (float)(t1 * t2 * t3);
}
```

TF-IDF

```
protected float score(BasicStats stats, float termFreq, float docLength) {
    double N = stats.getNumberOfDocuments();
    return (float)((1 + Math.log(termFreq))*Math.log((N + 1)/stats.getDocFreq()));
}
```

Pivoted Length Normalization

```
protected float score(BasicStats stats, float termFreq, float docLength) {
    double s = 0.75;
    double N = stats.getNumberOfDocuments();
    double avgDocLength = stats.getAvgFieldLength();
    return (float)((1+(Math.log(1+(Math.log(termFreq)))))/
                (1s+(s*docLength/avgDocLength))*
                (Math.log((N+1)/stats.getDocFreq())));
}
```

Jelinek-Mercer

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

    double p_s = ((1-g)*termFreq/docLength+ (g*model.computeProbability(stats)));
    double a_d = g;
    return (float)(Math.log(p_s/(a_d*model.computeProbability(stats))));
}
```

The Portion of the Equation, $|q|log\alpha_d$ was not included, as the gamma value was same for every document, and adding it wouldn't affect the ranking

Dirichlet Prior

```
protected float score(BasicStats stats, float termFreq, float docLength) {
    double p_s =  ((termFreq + m * model.computeProbability(stats))/(docLength + m));
    double a_d = (m)/(m + docLength);
    return (float)(Math.log(p_s/(a_d*model.computeProbability(stats))));
}
```

Portion for Adding $|q|log\alpha_d$

```
[Inside the Searcher.java, runsearch()]

TopDocs docs = indexSearcher.search(luceneQuery, searchQuery.fromDoc() +
                                                    searchQuery.numResults());
ScoreDoc[] hits = docs.scoreDocs;
String field = searchQuery.fields().get(0);

if(sim instanceof DirichletPrior){
    double m = ((DirichletPrior) sim).getM();
    for(int i = 0; i < hits.length; i++){
        Document d = indexSearcher.doc(hits[i].doc);
        List<String> tokens = new ArrayList<String>();
        tokens = tokenizeString(analyzer, d.toString());
        hits[i].score += (DirichletPrior)sim}.getQueryLength()*Math.log(m/(m +
                                                                    tokens.size()));
    }
    Arrays.sort(hits,(a,b)->(int)((b.score - a.score)/ Math.abs(b.score - a.score)));
}
```

	MAP	P@10	MRR	NDCG@10
Boolean Dot Product	0.21093625	0.28817204	0.59493185	0.34281428
Okapi BM25	0.21767134	0.30752688	0.59375793	0.36021567
TF-IDF	0.23722601	0.30967741	0.67674365	0.37735615
Pivoted Length Norm	0.15737778	0.23978494	0.43587655	0.26883078
Jelinek-Mercer	0.25854028	0.34193548	0.67845704	0.41099846
Dirichlet Prior	0.18487356	0.23655913	0.54444365	0.28728951

2. Please carefully tune the parameters in BM25 and Dirichlet prior smoothed Language Model. Report the best MAP you have achieved and corresponding parameter settings.

- Okapi BM25
  - Max MAP Value: 0.25936064
  - Paramaters:
    - K1 = 1.2
    - K2 = (Doesn't affect Ranking as last variable is 1 for all doc)
    - B = 0.75
- Dirichlet Prior
  - Max MAP Value: 0.2632312
  - Parameter:
    - $\lambda = 71$

3. Comparing Models.

Pivoted Length Normalization vs BM25

- Query: the synthesis of networks with given sampled data transfer functions
- Difference: 0.07583168630816622
- The terms used in the model seem relatively identical, normalizing the length of document with the average length of the document, or normalizing with the document Frequency and the number of document. The difference is found in the method the two models use in normalizing the count of term in the document (diminishing returns) is slightly different as Pivoted length uses logarithm, while BM25 simply adds the count to the denominator. Additionally, the normalization of the count of terms is a big factor in determining the result. And from the given two options, Pivoted length will go up higher, given the k1 value we have to BM25 as BM25 isn't that good at handling verbose queries.

Pivoted Length Normalization vs Dirichlet Prior

- Query: systems of data coding for information transfer
- Difference: 0.1753237909618664
- Language models have the term $|q|log\alpha_d$ added on to the end of the equation. This value is larger for longer queries and not for short queries. The query with the biggest difference is a relatively short query with pretty general terms. The short query could have an effect in resulting in a smaller value for the Dirichlet Prior. This would have been the main reason Dirichlet Prior had a lower value for this query, as that term would be the most significant term changing for different queries for the Language model.

BM25 vs Dirichlet Prior

- Query: systems of data coding for information transfer
- Difference: 0.17281056344863882
- Similar to the previous comparison, as the Language Models, Dirichlet Prior included, takes account for the length of the query, shorter queries impact the calculations to come out rather low.

4. Pick one of the previously implemented scoring functions out of

   - **Okapi BM25**

   to analyze under what circumstance the chosen scoring function will mistakenly favor some less relevant document (*i.e.*, ranks a less relevant document at a higher position than a more relevant one). Please correspond your analysis with what you have found in Problem 3.

- Okapi BM25 suffers from verbose queries as it sometimes, can result in the penalty of the verbose query being larger than the actual weight obtained from the equation, resulting in a much smaller value than it actually should be. This could also be partially due to the way Okapi normalizes the values as well. From what was found in problem 1, Okapi BM25 generally performed better compared to Dirichlet Prior Smoothing, but when we compare queries between Okapi BM25 and Dirichlet Prior Smoothing in the same way we compare it in Problem 3, queries found, such as "observations of rapid fluctuations in the earths magnetic field and their relation to the propagation of hydromagnetic waves in the exosphere" or "an abstract on the field distribution surrounding a charged thin circular disc resting on an infinite dielectric slab," were rather long and verbose ones. And thus, Okapi BM25 seems to rank a more relevant document in a lower position for verbose queries, and thus placing a less relevant document in a higher position.

Can you briefly summarize the major contribution of this paper?

- This paper mostly criticizes on the current information retrieval methods including the ones using vector space models like Okapi BM25 or Probabilistic Models or Language Models such as the Dirichlet Prior that these methods in that they are developed in such a way that the relevance of a query and a document is coarsely modeled at the level of document and query rather than the granular level of terms. This results in issues such as Okapi having cases where the penalty is larger than the weight, and therefore the paper suggests new straightforward constraints and a more inductive way of getting the relevance that would enable the functions to calculate the relevance in the level of terms. The paper also improvises the existing functions to fit the new criteria, using the functions derived for primitive weighting, dealing with query growth and document growth.

How do you think you can fix the problem you have identified in the ranking result analysis?

- The problem identified was that the original Okapi didn't perform as well for longer queries, and would give a lower score than expected for long and verbose queries. This issue could be fixed by modifying the weighing function of the Okapi to be a more general form that takes more account into the document generation as well as the constraints mentioned in the paper – a non-query term would penalize, while a query-term would add a diminishing increase. Therefore with verbose queries, a document with dense query terms would be ranked much higher.

Please relate your solution and corresponding implementation in the report. Also report the resulting ranking performance of your revised ranking algorithm.

- Implementing the approach given in the paper increased the MAP value as expected. The original Okapi, even with the tuned parameter only gave a MAP value up to 0.25936, while the new function written below gave a MAP value of 0.26715.

$$S(Q,D) = \sum_{w} C(w,Q) \cdot weight(w) \cdot \frac{C(w,D)}{\frac{b}{avdl} \cdot |D| + b + C(w,D)}$$

$$weight(w) = ln \frac{N - df(q) + 0.5}{df(q) + 0.5} \cdot \frac{1}{\frac{b}{avdl} + b + 1}$$

```
@Override
protected float score(BasicStats stats, float termFreq, float docLength) {
    b = 0.75;
    double N = stats.getNumberOfDocuments();
    double df = stats.getDocFreq();
    double avdl = stats.getAvgFieldLength();
    double lamda = (double)(1/(b+(b/avdl)+1));
    double weight = (Math.log((N-df+0.5)/(df+0.5))) * lamda;
    double term1 = termFreq/((b/avdl)*docLength + b + termFreq);
    return (float)(term1 * weight);
}
```

Additionally, the other outputs, P@10, NDCG and MRR were also found to have improved with the new implementation as well.

	MAP	P@10	MRR	NDCG
Old_Okapi	0.21767134	0.30752688	0.59375793	0.36021567
Tuned_Okapi	0.25936064	0.347311827	0.68091915	0.41992821
New_Okapi	0.26714930	0.35053763	0.69904782	0.42450660