Lucene experience: Describe your Indexer and Index Searcher built using the Lucene API. Include search enhancements you made along with justification for its effectiveness.

Indexer： I use Standard Anaylzer of Lucene Version_47, as you know, standard anaylzer usually is the best choice. It combined several complicated techniques to tokenize text. The difference here from the asg1 is that I parse each document into different fields, i.e., title, author, affilication, and content. After adding them into "Class Document", then I add each Document into IndexWriter to generate index file. The index files consist of several files, and I believe they are compressed and already not able to be understood by eye. It took 2744 milliseconds to generate index, total 1.59MB, comparing to my asg1, 4059ms and 1.25MB.

Searcher： Since the sample 225 queries are only about title or content, so I only make two queries:

```
Query titleQuery = new QueryParser(Version.LUCENE_47, "title", analyzer).parse(s);
Query contentQuery = new QueryParser(Version.LUCENE_47, "content", analyzer).parse(s);
```

Then I use BooleanQuery to combine them,

```
BooleanQuery q = new BooleanQuery();
q.add(titleQuery, Occur.SHOULD); // or Occur.SHOULD if this clause is optional
q.add(contentQuery, Occur.SHOULD); // or Occur.MUST if this clause is required
```

After several experiments, I found that it is best to set weight 0.2 for Field title and weight 0.8 to Field content. The way of experiment is to suppose the sum of two weights is 1. And for each weight pair (x, 1-x), I run the evaluation across 225 sample queries under top 20 results, to check the best precision and recall.

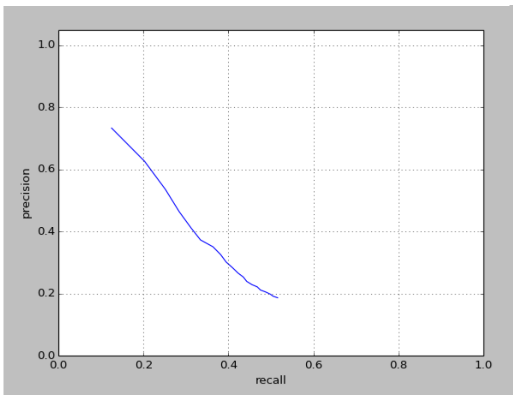
```
titleQuery.setBoost((float) 0.2);
contentQuery.setBoost((float) 0.8);
```

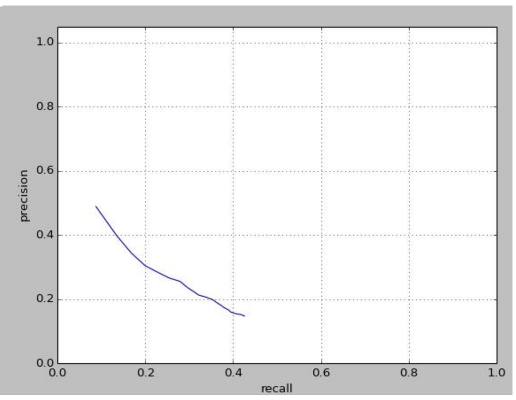
Benchmark output: Crisp and clear presentation of the results of two queries in the benchmark preferably in the form of a table or any other form of visualization.

The same as my asg1, evaluate from top 1 result to top 20 results across 225 sample queries, to compute the precisions and recalls, then we can make a precision-recall graph. After I get the precisions and recalls, I put the data into python to matplot it, such as we have the similar graph.

```
precisions = [0.733, 0.626, 0.536, 0.464, 0.411, 0.373, 0.351, 0.326, 0.302, 0.283, 0.266, 0.253, 0.239, 0.229,
0.222, 0.212, 0.206, 0.199, 0.192, 0.187]
recalls = [0.125, 0.202, 0.251, 0.284, 0.312, 0.3346, 0.363, 0.381, 0.394, 0.409, 0.422, 0.435, 0.443, 0.455,
0.467, 0.475, 0.487, 0.497, 0.505, 0.515]
```

As we can see, for top 1 result, precision and recall are 0.733 and 0.125; for top 20 results, precision and recall are 0.187 and 0.515. It does show a slightly better result than my asg1. Left is by lucene, right is of my own.


precision
recall


precision
recall