The libraries used were all standard java libraries and are all data structures or in regards to file io. The file IO was to read the file into maps, which then were evaluated and the results written to a file.

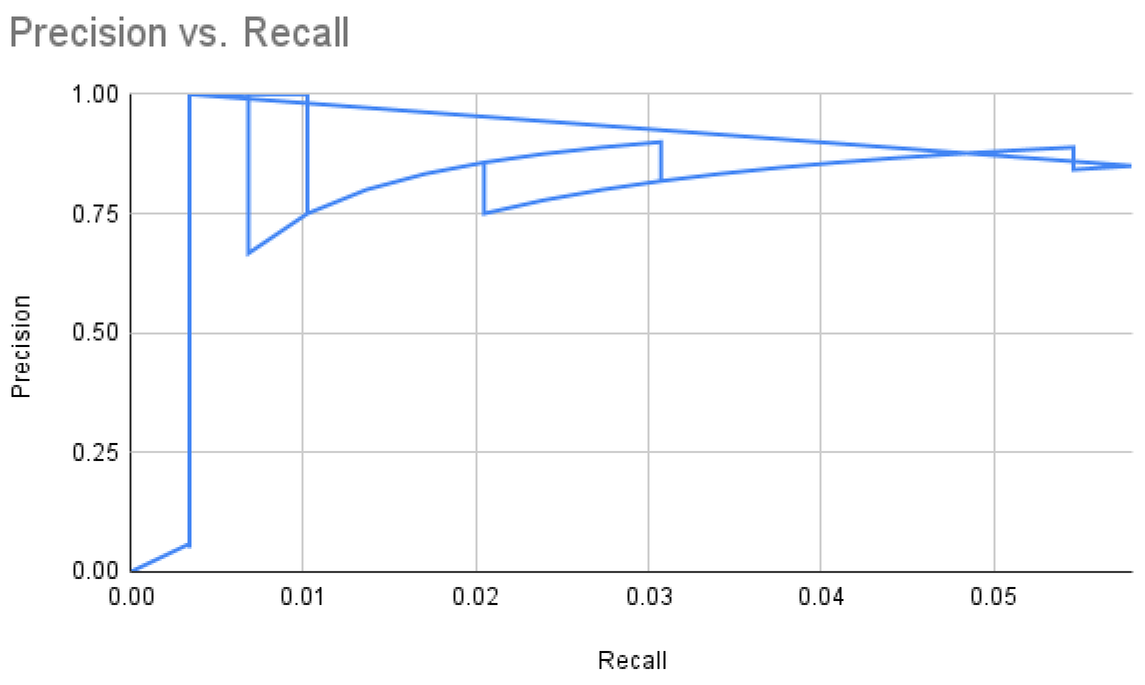
My implementation is relatively the same for all the different evaluation methods. All of the evaluation methods have a respective class that takes in a map of maps for the relevancy and scores. Where both the outer map takes in an integer for the queryId, and the inner map is the docids pointing to the ranking or relevancy respectively. Then a score is calculated for each individual query and the average is returned. Originally the score was calculated through a private function, as I only cared about the average, however the F1 function requires precision and recall for individual queries so I made them public to allow me to call them in the F1. The biggest flaw with my implementation is that by using a class that takes a set K and then returns a score I have to make 20 different objects for each of the k’s (1 through 20) in order to get the precision versus recall graph.

stress.trecrun	MRR	0.5672635174
stress.trecrun	P@5	0.3040322581
stress.trecrun	P@20	0.1669354839
stress.trecrun	NDCG@10	0.255978614
stress.trecrun	Recall@20	0.07200928107
stress.trecrun	F1@20	0.08234535546
stress.trecrun	MAP	0.04945825828
bm25.trecrun	MRR	0.2238094502
bm25.trecrun	P@5	0.1542168675
bm25.trecrun	P@20	0.1208835341
bm25.trecrun	NDCG@10	0.1987466126
bm25.trecrun	Recall@20	0.06741586374
bm25.trecrun	F1@20	0.07178878948
bm25.trecrun	MAP	0.07662411729
sdm.trecrun	MRR	0.6729665241
sdm.trecrun	P@5	0.4899598394
sdm.trecrun	P@20	0.3666666667
sdm.trecrun	NDCG@10	0.3056329327
sdm.trecrun	Recall@20	0.2124411893
sdm.trecrun	F1@20	0.219795269
sdm.trecrun	MAP	0.2586283614
ql.trecrun	MRR	0.7004394318
ql.trecrun	P@5	0.475502008
ql.trecrun	P@20	0.3602409639

ql.trecrun	NDCG@10	0.3061404925
ql.trecrun	Recall@20	0.2086375994
ql.trecrun	F1@20	0.2169545676
ql.trecrun	MAP	0.2517239344

The results I'm seeing suggest as you increase the number of documents being evaluated in precision, the value decreases, and recall gives a value less than precision at the same k. A large part of the fact precision goes down the larger the K is due to the fact we hope to use a retrieval model that will put the relevant documents higher ranked. If the precision was lower with a larger k value, then that would show an issue in our retrieval model as relevant documents would be ranked lower. MRR tends to give a larger number just based on the fact it only cares what rank the first relevant document is.

For mean average precision of a query with no documents retrieved depends on the fact if there are no relevant documents. If there are no relevant documents and no documents returned, then MAP should be 1. However if there are relevant documents but no documents returned then it should be 0. If there was a relevant document and no documents returned then the retrieval model failed. However if there's no relevant documents and no documents returned then it succeeded. That is a binary simple answer, but not a good answer because a retrieval model that only missed one relevant document shouldn't be scored as harshly as a retrieval document that missed more than one. But how do you assign a value for precision when it missed all of the documents. In the standard average precision formula it would return a 0, however the binary solution of giving a 1 if there was no relevant documents and 0 otherwise is likely the simplest answer without finding some ratio regarding total misses in the whole collection to misses on this query.


Precision vs. Recall
Precision
Recall