What AI Engineers Should Know about Search

JUNE 25TH, 2024

Things AI Engineers Should Know about Search. At least 50 of them :).

I probably don't need to discuss bi/cross-encoders, etc. A lot of great content is out there on those topics, especially folks like Pinecone (https://www.pinecone.io/blog/), targeting the AI / LLM / RAG crowd. But maybe you to quickly get some high-level, historical, lexical search context… Well I'm here for ya! You might be new to all this. Hired into an "AI Engineer" role and suddenly needing to index a lot of search content :). Welcome to this brave world, there's a lot you can contribute!

Some things to know:

1. The fanciest solutions don't matter as much as getting a good evaluation framework setup to evaluate the quality of search results
2. Your choice in Vector Database probably doesn't matter as much as how you fit all the pieces of a retrieval solution together - lexical, vector, reranking, query understanding, etc. More people seem to be aware of this now, and we've probably passed peak vector DB (https://softwaredoug.com/blog/2024/01/24/are-we-at-peak-vector-db).
3. There are many ways to label search results as relevant / not. Whether crowdsourcing human labelers, to processing clickstream data, to having an LLM evaluate search results using a prompt. They all have their pros / cons.
4. Search labels are often called judgments (https://softwaredoug.com/blog/2021/02/21/what-is-a-judgment-list) - because historically

because a “judge” would give a “grade” (relevant or not) to a search result for a query back in the first Information Retrieval competitions like TREC.

5. Evaluation methods all have different biases (https://jamesrubinstein.medium.com/measuring-search-a-human-approach-acf54e2cf33d). In human evaluators we have issues dealing with non-experts, people getting exhausted, people preferring earlier results as more relevant than later results, etc

6. In clickstream based judgments, we have other issues! We only get labels for results the search system returns in the top N (stuff that can be clicked on). And we deal with people’s lizard brains clicking on irrelevant, spicy pictures.

7. Presentation bias (https://softwaredoug.com/blog/2022/07/16/what-is-presentation-bias-in-search) in search UIs is a big problem. Basically users only click on what they see!

8. Overcoming presentation bias involves an active or reinforcement-learning mindset - where we “exploit” the current knowledge of search relevance, but find ways to “explore” new results. A great resource for overcoming this is Chapter 12 of AI Powered Search (https://livebook.manning.com/book/ai-powered-search/chapter-12/v-20)

9. Click models can turn clickstream data into judgments. There’s a great book Click Models for Web Search (https://clickmodels.weebly.com/uploads/5/2/2/5/52257029/mc2015-clickmodels.pdf) available for free that can guide your work understanding user search result clicks and conversions.

10. A lot of metrics exist for measuring the quality of a query - if query “zoolander” returns some search results, we can reference the judgments, to see whether or not we gave relevant results. Statistics like (n)DCG, ERR, MAP, Precision, Recall, F-Score are well understood in the search industry

11. Search comes with a precision / recall tradeoff (https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_652#:~:text=Precision%20%3D%20Total%20number%20of%20document If you cast a wide net, you’ll get more relevant results in the mix, but also likely be showing the users lots of irrelevant ones too!

12. Search people talk about an “Information Need” some informal specification of the information a user wants. They express that information need with multiple queries

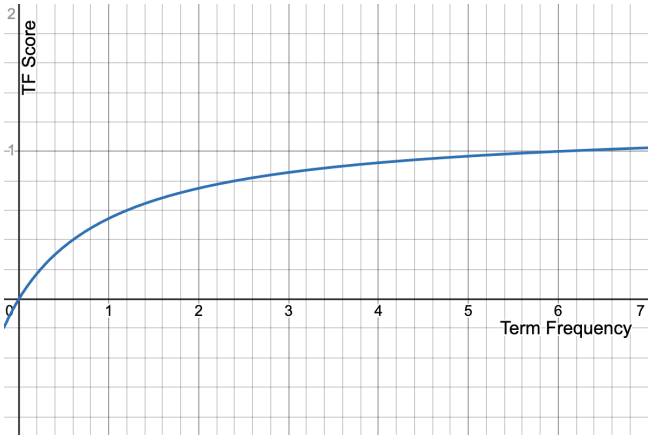
13. Information needs can be quite diverse - from looking for an exact item by ID to comparing / contrasting products to performing in-depth research on one topic and gathering notes, they each can require unique treatment in the ranking and UX

14. How users search is only getting more complicated. Users have very different expectations of question answering systems, social media sites, e-commerce, RAG, etc.

15. Maybe you’ve heard of BM25 (https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/). It is “just” TF*IDF (https://en.wikipedia.org/wiki/Tf%E2%80%93idf) that’s been heavily tuned and studied over the years.

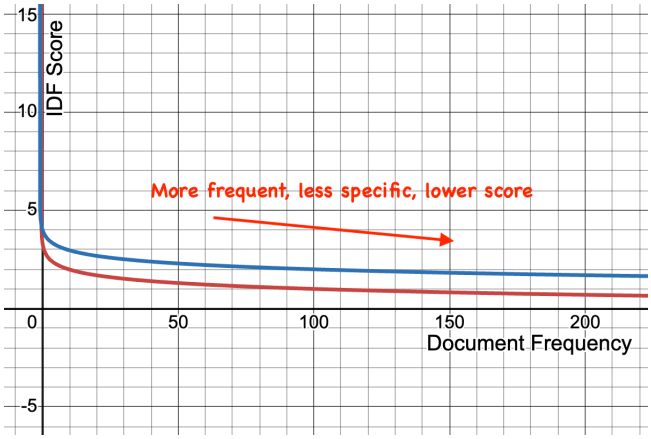
16. BM25 saturates the term-frequency component of TF*IDF, the intuition being that relevance and term frequency aren’t linearly

related, but after enough mentions of the term, it doesn't suddenly get "more about" that topic



17. IDF - or Inverse Document Frequency (1/doc freq) - measures the term's specificity the rarer the term, the more important it is to the user's intent. In a search for [luke] OR [skywalker] - skywalker is rarer and more specific so it scores higher.

18. IDF is not actually raw IDF - 1 / docfreq, but each similarity system has its own IDF curve that saturates non linearly



19. BM25 also biases towards shorter fields. Because the probability of you mentioning "Luke Skywalker" in a tweet is much much less than it happening to crop up once in a long textbook.

20. BM25 can be tuned with parameters k1 and b (https://stackoverflow.com/questions/38071877/how-to-choose-the-okapi-bm25-parameters-b-and-k1). The parameter k1 controls saturation of term frequency - such as very fast for short snippets of text, or very slow for longer snippets

21. The parameter b changes the influence of field length on the score. Set b very high and above average fields will be punished more (below average punished less).

22. BM25 is just one of a bazillion lexical similarity scores. Elasticsearch, for example, has many options (https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html) - including scripting the similarity!

23. BM25F is a variant of BM25 that accounts for matching in different fields (with their own term statistics). Field here meaning different text attribute like title or overview or tags or somesuch.

24. In many lexical search engines, you may be surprised when a common term in one field, is actually rare in another, screwing up BM25 calculations and suddenly pushing a seemingly irrelevant document to the top.

25. There are tools to compute the "true" specificity of a term beyond direct IDF, such as blending IDF across fields (https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html), or merging all the text into one big field.

26. Combining and weighing different fields is important. Fields in search play different roles, and need to be tokenized and scored differently. Like prioritizing a title match over something buried deep in a body

27. Search exists in an ecosystem. Some fields

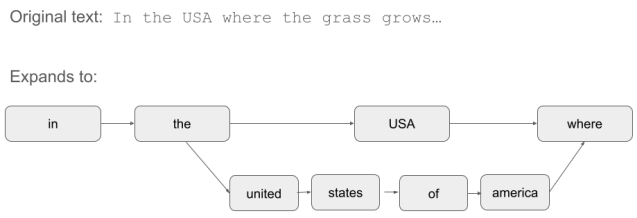
may be prone to abuse by users (ie keyword stuffing). Some fields have strong incentives to be reasonably non-spammy (like the title). How you score each can be very dependent on how much we can trust the curation of that content.

28. Search exists in an ecosystem - one of the best search hacks is to convince our users to SEO their content for search. They'll tune their content towards our search, rather than us needing to tune their algorithm for their bad content.

29. Search exists an an ecosystem - one of the worst search hacks is when our users realize they can SEO their content for our search, and they tune / abuse their content to be spammy and nearly malicious

30. Tokenization matters a lot. Not just in lexical search, but embeddings too! Tokenization (n-gram, word-piece, whole tokenization) and related topics like stemming, dealing with punctuation, etc can dramatically change how well search performs

31. Lexical token streams are really graphs (https://blog.mikemccandless.com/2012/04/lucenes-tokenstreams-are-actually.html) - in any position an equivalent token (synonym, or just a different way of expressing tokenization) is just a branch on this graph. And different branches might have different lengths! (IE USA vs United States of America).



32. It's not just about term matching - phrases matter a lot. Phrases correspond to important entities users often search for, or tags people

stick on things, or many other things we expect. This really matters in very technical disciplines like science, engineering, or legal research - where that phrase has some very specific meaning.

33. Phrases require encoding term positions - It's not feasible to build a giant lexical term dictionary of every term combination. So a lexical search system has to encode positions of terms in a data structure like a roaring bitmap (https://softwaredoug.com/blog/2024/01/21/search-array-phrase-algorithm)

34. As phrases need to be tokenized - and might produce equivalent terms at a given position - they are also graphs! If you expect USA to be a synonym of United States of America, that's a graph! A good query parser has to treat a search for USA as actually a search for "USA" OR "United States of America"

35. Some term-bigrams statistically go together. Think "Palo Alto" or "Nacho Cheese". These are known as collocations (https://opensourceconnections.com/blog/2019/05/16/unreasonable-effectiveness-of-collocations/) and can be great "poor man's" entities.

36. Lexical search isn't just about the BM25, but building phrase search in a way that respects the graph-based nature of the token stream

37. Usually search systems are a series of pre-processing (query/intent understanding) and post-processing (reranking / post-filtering) stages. With one or more retrieval engines in between.

38. Query Understanding (https://queryunderstanding.com/) comes in a lot of flavors. From broad, coarse-grain category classification (this is a search for an

image) vs extracting entities from a query (this is a search about "San Francisco") vs mapping a query to a vector for vector retrieval

39. Query Relaxation is a technique where you first try with a strict, high precision query. Then if you get no / poor results you reissue the query with a relaxed query. A simple version is switching from an OR query (match any terms) to an AND query (match all terms). But its an active area of research - here's an interesting talk on the subject. (https://haystackconf.com/2019/query-relaxation/)

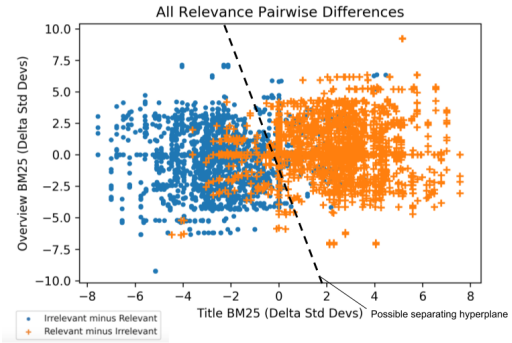
40. Related to query relaxation is the idea of relevance 'tiers' (https://livebook.manning.com/book/relevant-search/chapter-7/156). High precision queries are boosted higher, and the lower precision / higher recall strategies are given less weight. If the high precision query doesn't match, you effectively fall back to the lower quality/broader results.

41. You can use filter queries (https://www.elastic.co/guide/en/elasticsearch/reference/current/query-filter-context.html) to express what definitely is NOT relevant so that you can exclude those from the results. This helps eliminate false positives from creeping in top N.

42. Applying machine learning to optimize ranking is often called "Learning to Rank". As they involve a list-wise ranking relevant results above irrelevant results, they can be a bit of a different type of machine learning than classification or regression.

43. Learning-to-Rank functions have a long history of study. SVMRank (https://docs.google.com/presentation/u/0/d/1UI4KHWDP0hONl21i1tyvvDyC\ is a simple support vector machine that classifies relevant vs irrelevant pairs using a binary classifier SVM. LambdaMART

(https://www.microsoft.com/en-us/research/uploads/prod/2016/02/MSR-TR-2010-82.pdf) attempts to optimize a list-wise relevance metric (like NDCG) using gradient boosting



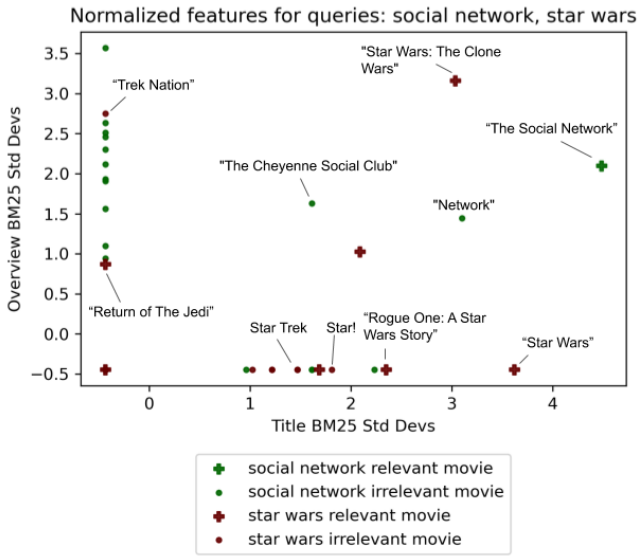
(Image from AI Powered Search (http://aipoweredsearch.com))

44. LambdaMART is one of those things like BM25 that's old school, but works, and you probably should learn. Here's a good blog post (https://softwaredoug.com/blog/2022/01/17/lambdamart-in-depth) complete with notebooks from scratch :)

45. Something like LambdaMART is great when there are a lot of lexical tabular features. Like statistics you want to use in ranking.

46. LambdaMART is great for reranking hundreds or thousands of results relatively inexpensively.

47. The relationship between relevance and the features are very non-linear. Some things like recency might matter if searching for news. Other things may matter when searching for canonical information. Even a simple relationship between relevance and title / overview BM25 for movie search below is complicated:



(Image from AI Powered Search (http://aipoweredsearch.com))

48. In Learning to Rank training data, a common hack is to sample other query's positive results as negative examples for this query - similar to contrastive learning (https://www.v7labs.com/blog/contrastive-learning-guide)

49. It's pretty common to have multiple reranking stages. Starting with dumb rerankers optimized for top 1K up to fancy cross encoders that rerank small amounts of search results.

50. Popular search systems (Solr, Elasticsearch, Vespa, OpenSearch) have differing levels of functionality for gathering Learning to Rank features, storing, and performing inference on ranking models

51. Good features in Learning to Rank systems are both orthogonal from existing features and add value. Orthogonal implies they measure different things, and probably come from different systems (like BM25 vs embeddings vs ??) almost guaranteeing there's no one size fits all solution to search

52. Ranking and similarity are quite different things! (https://x.com/HanchungLee/status/1805325078076473733) - Two pieces of text can be similar (measured with an embedding, BM25, or otherwise) but still be spammy, old, or otherwise low value.

53. Sometimes you don't need a reranker! A lot of people get by just by optimizing the scoring of the first pass using techniques like genetic algorithms or Bayesian optimization (https://haystackconf.com/us2022/talk-5/)

54. It's OK to cheat! A common technique is to build a collection, like a signals collection (https://livebook.manning.com/book/ai-powered-search/chapter-8/v-20) that memorizes engaging results for a query. Then when a user searches you can upboost highly engaging content (or downboost stuff that's had its chance).

55. Query feedback can be fed into your system! - you can learn a lot about your queries by following the types of content that are interacted with. Such as the category they tend to go with, or the embedding of an image that tends to get clicked on. Learn about Relevance Feedback (https://en.wikipedia.org/wiki/Relevance_feedback)

56. A lot of tools exist to prototype. From SearchArray (http://github.com/softwaredoug/searcharray) to BM25S (https://github.com/xhluca/bm25s) you can kick the tires of different lexical solutions to your problems!

57. Read Introduction to Information Retrieval (https://nlp.stanford.edu/IR-book/information-retrieval-book.html) by Manning et. al. It's the search bible and covers the foundational concepts of search and retrieval.

58. Be sure to check out the many resources about search and information retrieval, one great list is at Awesome Search (https://github.com/frutik/awesome-search) Github repo

Special Thanks to Han Lee (https://leehanchung.github.io/) and Felipe Besson (https://felipebesson.com/) for reviewing this post and giving substantive edits and feedback!

Doug Turnbull

🏠 More from Doug (/)
📎 Twitter (https://twitter.com/softwaredoug/) | 📎 LinkedIn (https://www.linkedin.com/in/softwaredoug/) | 📎 Mastodon (https://hachyderm.io/@softwaredoug)
💼 Doug's articles at OpenSource Connections (https://opensourceconnections.com/blog/author/doug-turnbull/) | Shopify Eng Blog (https://shopify.engineering/search?link_search=true&q=Doug+Turnbull)