CS47300 Fall 2020 Assignment 6 and Solutions

1 Search Engine Optimization

Google recommends having a single URL refer to each page. Is this for their benefit, or for yours (as a web site designer trying to get your page seen)? It would make sense that having more ways to refer to the page would make it easier to find (just like products may have many different sizes to give them more shelf space in the supermarket.)

Given what you know, analyze this question. Assume that you can have either a single url for your page (e.g., `https://www.cs.purdue.edu/~clifton/cs47300/assign6.pdf`), or two (the above, and `https://www.cs.purdue.edu/~clifton/cs473/assign6.pdf`). Furthermore, assume you have defeated any duplicate detection, so the search engine will separately index both pages if you have two URLs.

Analyze if you would expect your search result ranking to be higher or lower with two URLs pointing to the page. Do this for each of:

1. A content-based ad-hoc retrieval approach (e.g., the vector-space model),

   Both documents would get the same score, so would presumably appear next to each other in a ranking. Neither would be higher than if you just had one.

   The one possible change is that the document frequency of terms in the documents would increase (since it now appears in the duplicate as well as the original), so the IDF would decrease. This could result in our documents getting lower scores, and documents using terms that match the query that AREN'T in our documents to be ranked higher.

2. A network structure based model (e.g., PageRank), and

   Assume that we originally had two pages pointing to our page. When we create the duplicate, the authors of one of those pages may find the duplicate, and the other the original, so each of our pages would get half the pagerank of what would have happened if we had duplicate. So it could substantially lower the pagerank.

3. A collaborative filtering approach.

   If we were lucky, people who didn't like the page may see the original, and people who did like it saw the duplicate. So the duplicate could be highly recommended. However, since fewer people saw it, there would be fewer people similar to those people, so the high recommendation would probably go to fewer people than the average recommendation it would get if we only had a single page.

   So for collaborative filtering, it could both help and hurt. It would make the "bootstrapping" problem worse, as fewer would see each one. But one of them would likely end up with a higher score (just through random chance of who saw it), which could lead to more strongly positive recommendations for those fewer people who it is recommended to.

You should be able to come to a conclusion for each, based on a mathematical analysis, if you are better off having one URL or two.

2 Metric Limitations

We often talk of improving information retrieval with respect to some metric (recall, precision, mean average precision, F1 score, false positives / false negatives, etc.) However, there may be different ways to achieve a high score that can lead to some serious ethical issues.

For example, assume we have a "fake news detector" that has a 20% false positive rate, in other words, it identifies 20% of true articles as fake. This might seem high, but if we had an influx of, say, spam messages on

facebook it might seem worthwhile if this is what it took to get the amount of spam shown (false negatives) to reasonable levels.

Imagine two different scenarios. In one, the false positives are randomly distributed across all types of true articles. In the second, all true articles critical of the ruling party are falsely deemed to be fake news. I think most would consider the second situation to be much worse that the first.

1. Which do you consider more likely, randomly distributed false positives, or false positives that fall on a particular topic/viewpoint? Hint: Think of this as a text categorization problem, and frame your answer in terms of the way a text categorization method would work.

   I would imagine false positives that fall on a particular topic or viewpoint would be more likely. First of all, your labels for fake news and true news has to come from somewhere. On media topics that are highly divided, naturally those topics will have a wider mix of true and false claims, so prediction will most likely become less accurate.

2. Come up with another situation, where two different outcomes might have the same result with respect to some metric, but ethically there is a significant difference in which seems acceptable and which does not. This should be a situation other than fake news detection. Give a specific example, stating what you are using to measure "success" and two different ways that achieve the same score where you would consider one acceptable and the other unacceptable.

   One situation that parallels that of the last situation is maybe a survey analysis tool for predicting some mental illness. A metric such as precision or recall may report acceptable statistics overall, but upon conditioning the results on fields such as race or sex, one might discover the model has a tendency to predict abundantly more false positives on a particular subgroup. If the false positives were evenly distributed, this would be acceptable as everyone would have reasonable statistics for whether they have some particular mental illness or not. But having all the errors centered on one subgroup could cost that subgroup a lot of money and a lot of discrimintation could be the product.

3. See if you can come up with a different situation that is a different type of information retrieval task (e.g., if in 2 you used a collaborative filtering example, perhaps use ad-hoc retrieval here.) You don't need to give much detail for this, just the idea.

Suppose there is a situation where a content-based filtering system attempts to recommend articles to individuals. This model is evaluated according to feedback from users, specifically the model attempts to assign a predicted score and recommend to a user off of that, and the model is rewarded the closer its prediction is to the user's assigned score. This method of evaluation may show effective performance at a high level, and is a fairly good idea if there is a guarantee that every article will be treated and recommended fairly. But does not take into account an unfair system where a document or group of documents might not be recommended to anyone. This could result in some articles very infrequently being recommended to any users, and may even isolate some specific subgroups of individual authors, but the metric would not be able to detect this unless further conditioning of the results is done.

3 Sentiment Analysis

We have the set of opinion words and sentiments: "must-have" (+1), "terrible" (-1), "amazing" (+1), "awful" (-1), "small" (0)

Given the example sentence: "These headphones sound amazing."

1. Give a sentiment estimate of the first sentence using aggregate opinion and the provided opinion words.

   The sentiment should be positive.

Now, given the sentence: "This premium handkerchief, this $10,000 used napkin, is a must-have."

2. Give a sentiment estimate of the second sentence using aggregate opinion and the provided opinion words.

   The sentiment should be positive.

3. Does the second statement's contents seem to match the sentiment assigned by the aggregate opinion?
   No, it doesn't, the statement seems to be sarcastic. "used napkin" indicates that the sentiment is conflict to positive.

Finally, if we take the sentence "The new, glorious $20,000 toothbrush is a must-have." and know that it is labeled with an overall negative sentiment:

4. What does that tell us about our set of opinion words?
   The set of opinion words is not enough for the sentiment analysis. There can be many other sentiments, e.g. sarcastic.

4 Evaluating Non-linear Results

We often assume that the user examines search results from top to bottom and in sequential order. However, some applications give results in a grid view (this is common in product search, e.g., hotels, or in searches where the result is shown as an image rather than text.) Users might go either left to right (column-wise) or top to bottom (row-wise) in perusing the results. Design a rank-based evaluation metric for such a search result. You can modify an existing metric such as MAP or propose a new one.
We can analyze this question in several ways.
A simple solution would be to make assumptions on user behavior. For example, in a grid view, users may examine results from left to right, one row at a time, or top to bottom, one column at a time. We can generate an ordered list based on this assumption. Then we can apply any existing rank-based evaluation metrics (e.g., MAP, MRR) on the order list.
We can take a hybrid approach to evaluate assuming left to right or top to bottom, and combine both evaluations. We can sum those values to generate a single number. Alternatively, we can get a weighted sum based on some weights. We can learn these weights from the user behavior analysis. For example, how frequently the user scan results left to right or top to bottom.
We can also consider some mixed user examining behavior. For example, in the grid view, we can consider top left result as the most important place and take distance from that cell to all other cells and order the results list based on the increasing distance from the top-left result. One way to compute the distance is to use the Manhattan distance between the top-left and the target result cell.

5 Question Answering and NLP

Given the query "When is the CS47300 final exam?", run through steps a question-answering system would use to answer this as a question (rather than just returning a web page such as `https://www.cs.purdue.edu/~clifton/cs47300/` or `https://roomschedule.mypurdue.purdue.edu/Timetabling/gwt.jsp?page=exams`). For example, NLP tasks in parsing the question, search for pages containing the answer, etc. Most of this follows from the last 10 mintes of the lecture on question answering, but you are welcome to use methods from other sources (textbook, NLP lecture, etc.)

1. What is the type of query? (Informational, Navigational, Transactional). Explain briefly how this might be determined for this specific query.

Informational.
A navigational query is a search query entered with the intent of finding a particular website or webpage. Informational queries are queries that cover a broad topic (e.g., colorado or trucks) for which there may be thousands of relevant results. A transactional search query is a query that indicates an intent to complete a transaction, such as making a purchase. This query is asking for information: "when" is the CS47300 final exam, and the search engine will return multiple webpages. Thus, it's informational.

2. How might you identify appropriate sources for the information? Assume there isn't a pre-defined structured knowledge base, but you have to use web search.

   I will check if the source is reliable or not. I'm finding the time of CS47300 finel exam, so I'll check if the source comes from some official domain like `cs.purdue`. Secondly, I'll check if the page is up to date. I assume we are tring to fine the time of final exam of Fall 2020. Thus those webpages for past semesters are not relevant.

3. How might you go from a web page to identify specifically what in that web page answers the question?

   I'll search on "final", see if there's information on the final. The I'll check whether it contains information of the time of the final.

4. How might you go from a fragment of text (say, a line or sentence containing the answer) to something that actually sounds like an answer? Hint: If you are more specific in your answer to part 1 than just informational, transactional, or navigational (which should be easy to figure out from word in this query), you can determine a ¡q¿template¡/q¿ for what an answer would look like.

   CS4700 final exam time: mm/dd/yyyy, HH:MM - HH:MM

You should be able to answer each of these in a few sentences.

6 Big Data methods

Sketch out a means to use a Map/Reduce style system to build an inverted index supporting TF/IDF queries. In other words, you would like to produce a term, document frequency, and list of documents with their term frequencies.

A simple option would be to view this as producing two indexes, one with term/document/term frequency, one with term/document frequency. A better option would be to produce something that combines all the information. Don't worry about producing a "file" - simply collecting and emitting the appropriate sums is fine.

You may use the pseudo-code syntax from the 11/18 lecture, or spark syntax (java or python) from 11/20. You do not need to produce something even remotely executable, the goal is to capture the basic logic used in a map-reduce framework.

If you do feel like creating something executable, we do have a spark cluster you can try it on. We did a project on this last year, and it is still available:
`https://www.cs.purdue.edu/homes/clifton/cs47300/Project3.sxhtml`
While this isn't the same task, it does give a simple example program in both java and python. A caveat - this is not a production environment. In particular, I've see situations where students with a lot of python cruft (unwanted software) from prior courses that was auto-installed by scripts run as part of those courses can result in simple programs (even the samples provided) failing, and sometimes spawning multiple processes and bringing the whole system down. So feel free to try it, but if something doesn't work, don't keep trying the same thing.

Term frequencies are just the "count" we discussed in class, but the key is the term/document pair, rather than a term. But this isn't very interesting as a map-reduce problem, as we can easily scan through a single document to produce this. Better is to make this part of the

"map" phase, scanning the entire document and emitting the term, and the value would be a data structure with two things: 1 (the document frequency derived from counting the term in that document), and the document/term frequency for that term.

Document frequencies are a bit more difficult, as we need to make sure we only count each document once. Using the above as our map phase, for each term, our input key would be the term, and the value would be two things: a current count and a set of document/frequency pairs. We then iterate over the values, adding each count to our count total, and unioning the document/frequency pairs into a running set for that document. When done, we emit the term, the count we've arrived at for document frequency, and the set of documents contributing to that.