Text Technologies for Data Science

Qais Patankar - s1620208 - 17th November 2019

IR Evaluation

First we opened and parsed the `qrels.txt` file. This was extremely simple, especially since the tuples did not have a space after the comma, allowing us to just split by spaces. We dealt with the head `` `1:` `` and tail `` `(9090,3) (6850,2)` `` separately. We asserted that the head ended with a colon to ensure that the input file was not malformed. Similar assertions were made when parsing the tuples.

Reading and parsing each result file (`S1.result`, etc) was also simple — we leveraged use of a `defaultdict(list)` to make appending of query results simpler. As before, we split by space, and used assertions (comparing certain columns against *"0"*) — the latter allowing us to ensure that the input file is not malformed.

We created a `get_scores(retrieved, relevant)` function that computes scores for a single query, returning a dictionary with all the relevant keys: "P@10", "R@50", "r-Precision", "AP", "nDCG@10" and "nDCG@20".

This function calculated recall (at 50) on its own, but relied on other functions to compute the other metrics. We created generic functions for cut-offs at k, so the only other functions were: `precision_at_k`, `average_precision`, and `ndcg_at_k`.

Best performing system

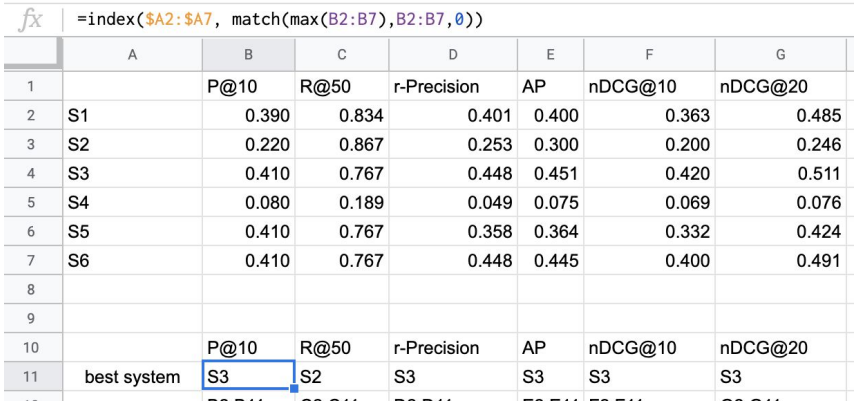
Here we attempt to find out whether the best system for each metric is statistically significantly better than the second system (S2).

	P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
best system	S3	S2	S3	S3	S3	S3
best system mean	0.410	0.867	0.448	0.451	0.420	0.511
S2 mean	0.220	0.867	0.253	0.300	0.200	0.246
p-value	0.073		0.271	0.354	0.195	0.109
significant?	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

As you can see, the results are not significant. Therefore, each metric is not statistically significantly better than the second system.

This is because all p-values are greater than our significance level of 5%.

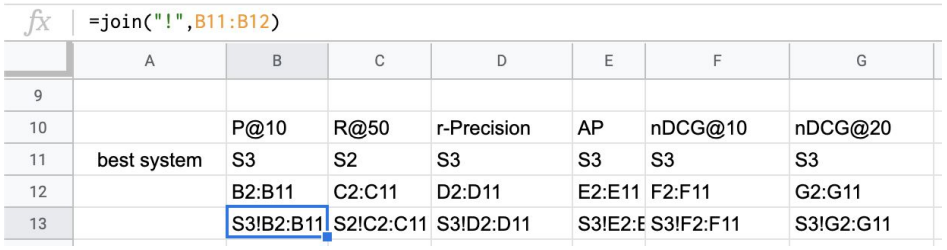
To calculate this we imported all eval files to Google Sheets, each file as its own sheet (tab). We used the following formula to find the name of the system that had the maximum average for each column.



```
=index($A2:$A7, match(max(B2:B7),B2:B7,0))
```

	A	B	C	D	E	F	G
1		P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
2	S1	0.390	0.834	0.401	0.400	0.363	0.485
3	S2	0.220	0.867	0.253	0.300	0.200	0.246
4	S3	0.410	0.767	0.448	0.451	0.420	0.511
5	S4	0.080	0.189	0.049	0.075	0.069	0.076
6	S5	0.410	0.767	0.358	0.364	0.332	0.424
7	S6	0.410	0.767	0.448	0.445	0.400	0.491
8							
9							
10		P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
11	best system	S3	S2	S3	S3	S3	S3

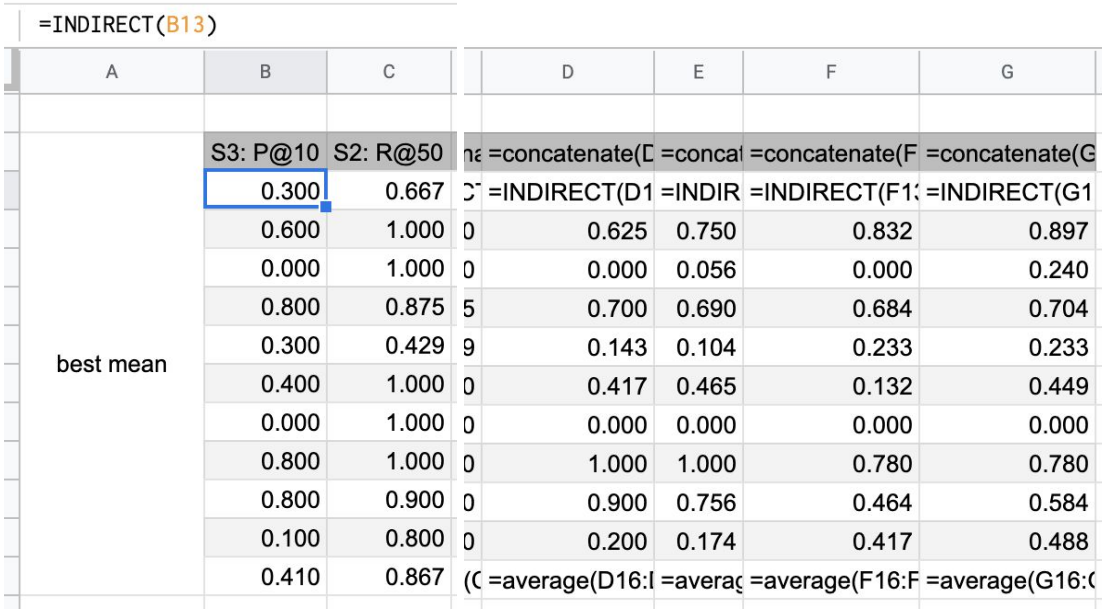
Then, in this same sheet, we compose a cell reference by string, ranging from query 1 to query 10 in the best system (for each column).



```
=join("!",B11:B12)
```

	A	B	C	D	E	F	G
9							
10		P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
11	best system	S3	S2	S3	S3	S3	S3
12		B2:B11	C2:C11	D2:D11	E2:E11	F2:F11	G2:G11
13		S3!B2:B11	S2!C2:C11	S3!D2:D11	S3!E2:E	S3!F2:F11	S3!G2:G11

Then, again in the same sheet, we use INDIRECT to import that “best range” from the other sheet, into this sheet. Additionally, the tile is concatenated to make this clear.

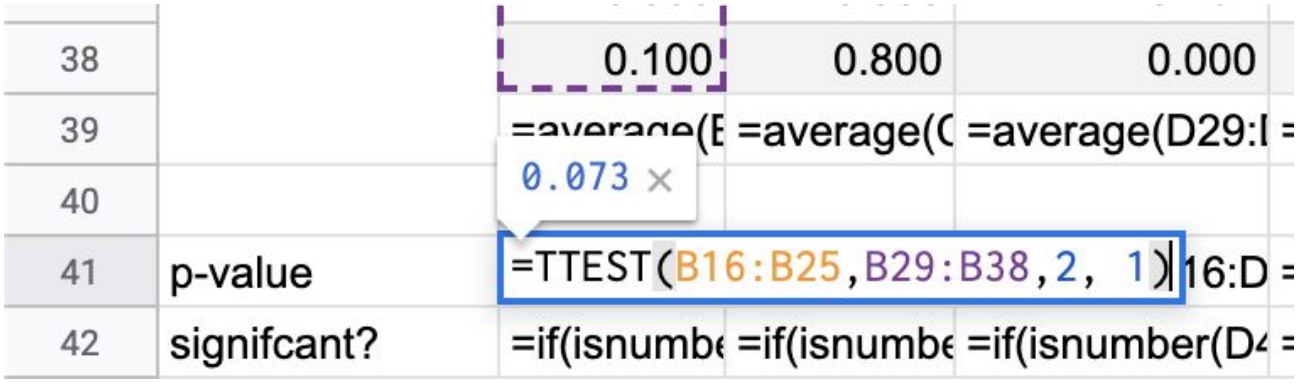


```
=INDIRECT(B13)
```

	A	B	C	D	E	F	G
		S3: P@10	S2: R@50	=concatenate(D	=concat	=concatenate(F	=concatenate(G
		0.300	0.667	=INDIRECT(D1	=INDIR	=INDIRECT(F1	=INDIRECT(G1
		0.600	1.000	0.625	0.750	0.832	0.897
		0.000	1.000	0.000	0.056	0.000	0.240
		0.800	0.875	0.700	0.690	0.684	0.704
		0.300	0.429	0.143	0.104	0.233	0.233
	best mean	0.400	1.000	0.417	0.465	0.132	0.449
		0.000	1.000	0.000	0.000	0.000	0.000
		0.800	1.000	1.000	1.000	0.780	0.780
		0.800	0.900	0.900	0.756	0.464	0.584
		0.100	0.800	0.200	0.174	0.417	0.488
		0.410	0.867	=average(D16:I	=averag	=average(F16:F	=average(G16:(

We do the same thing for the S2 sheet, like so: `=indirect(concat("S2!",B12))`

Now that we have the two ranges, we can use TTEST sheet function to get us the p-value to compare against the significance level (0,05).



We wrap **< 0.05** comparison checks with **isnumber** checks for p-value to cover the unique case where S2 is the best for R@50. They are the same, so we know it's not significant.

What was learned

In Python we learned that we can pipe regular print output to a file, instead of stdout. This allowed us to quickly our code to work for all results file, instead of just for a single results file. We could always print our debug messages to stderr, so we can see them in the terminal.

We also learned how to compute all the aforementioned metrics in a way that is easy to commit to memory (since it's "naturally computeable").

How to use INDIRECT and TTEST in Google Sheets.

Challenges faced

Possibly the most challenging part was computing the normalised dcg — this required a bit of thinking in deciding whether or not the *cut-off at k* should also trim the *relevant* list (since we only trim the *retrieved* list when computing `precision_at_k`).

Additionally, when computing the ndcg, enumerating through a subslice from **1 to k** (rather than **0 to k**) resulted in a lot of confusion and exposure to off-by-one errors. In the end we decided to enumerate from **0 to k**, and simply *continue* if **i** was equal to zero. As usual, we would then be able to just do `log2(i + 1)`, instead of the more confusing `log2(i + 2)`.

Improvements

In several places (`get_scores`, `precision_at_k`, `average_precision`) we recompute the same data and convert between lists and sets.

```
retrieved_docids = list(map(lambda d: d["doc_number"], retrieved))
relevant_docids = list(map(lambda t: t[0], relevant))
```

For large datasets this can be slow, so in the future, we should try to compute the data once and pass it around.
The entries of qrels.txt file were parsed into a tuple whereas entries of each results file were parsed into a dictionary. Coping with this inconsistency was challenging and meant that we had to be careful to access each data structure as appropriate (keying in by number for tuples, or by string). In the future we would use the same data structure — possibly a `namedtuple`. Additionally, we could use Python 3's typing feature.

Classification

- 1-2 pages on the work you did on classification, and how much improvement you could achieve over the baseline, and how it was achieved (new features? learning method? more training data? ... etc.)

First we tried lowercasing all the features, and this increased accuracy from 0.605 to 0.642. Alternatively we tried stripping all non-alphabet characters, and this increased accuracy from 0.605 to 0.645. Then we discovered that if we combine these two, the accuracy is raised to 0.664.

Then we included hashtags (in addition to the word without the hashtag), the accuracy is raised again to 0.686. Our next step was to include "cleaned hashtags", but this reduced the accuracy by a negligible amount, so we undid that change.

Challenges faced & future improvements.

We kept the domain name (with dot) of URLs as a token but, surprisingly, this did not affect the accuracy at all. Looking through all the URLs, this is due to the large number of twitter shortening links in these tweets.

We decided to use the unshortenit library (https://unshortenit.readthedocs.io/en/latest/) to convert these to real URIs but they were not changing to their original URIs. We then discovered that all twitter links were "invalid", and therefore had nothing to unshorten to. This turned out to be because twitter's shortening URLs are case insensitive, and we were downcasing URLs first. By the time we wrote a workaround for this, running the classifier was taking too long, so we did not get the chance to provide accuracy feedback on this.

This leads us to introducing parallelization in the future, as making our code concurrent (and run in parallel) would alleviate the time issues. Somewhere we use a hack to accumulate a list whilst also mapping (transforming) tokens. This is a blocker to the usual quick-fix solutions to parallelization (i.e. the multiprocessing library).

We still only use binary features, so we believe it would be of benefit to use non-binary features. We could visit links in each URL and include the following information:

- <title> of the webpage
- Description tag of the webpage
- The meta “tags” of the webpage

Ultimately we learned that just throwing more data at a classifier isn’t a sure-fire way to improve its accuracy — it needs to have useful, largely relevant tokens. We also learned that there are certain programming practices we should avoid doing to ensure that code can be parallelised in the future.