Rookie facets

January 2016

1 Rookie facet engine

When a user enters a query, Q, into the Rookie UI, Rookie executes a traditional document search using an information retrieval system[1] to find a set of query-responsive documents, D. In some cases, Rookie may filter D into a subset of documents which fall within a specified timespan, T. Regardless, Rookie passes either the set of documents or the set of time-filtered documents to its facet engine.

1.1 Index time

The Rookie facet engine makes use of a binary matrix, M, constructed at index time. The rows of M represent each part of part-of-speech filtered ngram type [2] that occurs at least 5 times across the entire corpus. The columns of M represent all documents in the corpus. A value of 1 at position $M_{i,j}$ indicates that the i^{th} ngram type occurs in the j^{th} document. (Similarly, a value of 0 indicates that the ngram does not occur in the document).

Rookie also creates a vector of "inverse document frequencies", $i\vec{df}$, for each ngram type at index time. Each component of the vector, idf_i, represents the inverse document frequency for a particular ngram, which is computed as follows:

$$idf_i = ln(N/df_i) \tag{1}$$

where N is the number of documents in the corpus and df_i is the number of documents in the corpus that contain the ngram type. For example, if there are 10 documents in the corpus and 2 contain the i^{th} ngram, the inverse document frequency for that ngram would be $ln(10/2)$.

1.2 Query time

At query time, Rookie creates a submatrix from M, picking those columns which respresent the documents in D. It then sums the rows of the submatrix to

[1] the current implmentation uses Whoosh

[2] Earlier versions of Rookie selected people, organizations and ngrams to ensure facet diversity. But this proved unnecessary

generate a summary vector of "term frequencies", $\vec{tf}$. The i$^{\text{th}}$ component of this vector, tf_i represents how many times a given ngram type occurs within the documents in D.

Element-wise multiplication of $\vec{tf}$ and $\vec{idf}$ produces a vector of tfidf scores, $\vec{S}$ for each ngram type.

$$\vec{S} = \vec{tf} \odot \vec{idf} \tag{2}$$

Rookie selects highest-scoring ngrams greedily from $\vec{S}$ until it reaches a fixed number of facets [3], passed as a parameter in the call to the facet algorithm. However, because top-scoring facets often use slightly different string tokens to refer to the same person, organization or concept (shown in table 1.2) [4] Rookie uses heuristics to clean up facets for presentation during its greedy search.

Table 1.2 shows the 15 highest ranking facets for Q="Mitch Landrieu", with and without heuristic cleanup. The top three raw facets are all short phrases which contain substrings of "Mitch Landrieu". Meanwhile, fully one fifth of the facets are slight varations of the string "Sheriff Marlin Gusman". Such overlap clutters the UI with repetitive information and requires time and energy from users.

Raw	Heuristic cleanup
Mayor Mitch	City Council
Mitch Landrieu	consent decree
Mayor Mitch Landrieu	Sheriff Marlin Gusman
City Council	Department of Justice
Mitch Landrieus	Police Department
New Orleans	Chief Administrative
Andy Kopplin	Mayor Mitch Landrieus
consent decree	Ryan Berni
staff writer	Orleans Parish Prison
Sheriff Marlin	Andy Kopplin
Orleans Parish	District Court
Landrieu administration	Landrieu spokesman
Sheriff Marlin Gusman	City Hall
Marlin Gusman	Landrieu Administration

Table 1: Top 15 facets for Q=Mitch Landrieu, in raw form and with heuristic cleanup

Thus, as Rookie picks the higest scoring facets using tfidf scores, it runs several heuristic checks to avoid duplicates and guess at the best string representation. Heuristic checks include: deleting facets that are identical except for those ending s and replacing a shorter facet (by string length) with a longer

[3] TODO: character budget

[4] TODO: Don't understand why exactly this is happening. POS tagging errors? Linguistic variation in language in articles? Refer to different things?

facet when the token-level Jaccard similarity between the facets is greater than 50%.

Facet problem	Example
Nested facets	Orleans Police & New Orleans Police
Split facets	Mayor Mitch & Mitch Landrieu & Mayor Landrieu
Query-facet overlap	Q=Mitch Landrieu, F=Mayor Mitch
Common term	Orleans Parish

Table 2: Common facet problems

1.3 Coreference

Combining facets in this way could be said to be an extremely lightweight form of coreference resolution, a large topic within natural language processing, philosophy [5] and linguistics. Rookie's goal is more modest: simply finding some string which represents some facet of underlying documents, without confusing the user with duplicates.

The motivation for this approach is twofold. First, this form of coreference resolution has proven suitable for this particular application. Second, coreference resolution methods are not currently fast enough to run run at query time in a user-facing application.

1.4 Performance, implementation & benchmarking

Rookie's facet algorithm is implemented in Python. It makes use of rapid matrix operations in Numpy. The speed of the facet algorithm varies with the size of D. When the size of D is around 300 (ex. Q=Mitch Landrieu) the algorithm runs in 60 milliseconds (ordinary clock speed) on a new laptop. When the size of D is 100 (ex. Q=levee) the algorithm runs in 30 milliseconds.

A speedy facet algorithm is crucial for Rookie as single query from the UI might require five or ten calls to the facet engine, one for each date bin. [6].

[5] Do "Obamacare" and the "Affordable Care Act" refer to the same thing?
[6] not discussed here