Lab 5: Information Access

TDT4275: Natural Language Interfaces

Jonas Myrlund

April 23, 2013

1 Written Assignments

A Using either web searches or a large corpus, devise a query using a polysemous word.

I want to find out more about large cranes – the bird. I start out with the following query: "large crane".

See figure 1 for the initial results using Google[1].

B Then, devise two or three alternate queries which help to disambiguate the polysemous word and repeat the search.

The query "large crane" could be less ambiguous in the following ways:

1. "large crane bird"
2. "large crane feathers"

As figure 2 exemplifies, these queries result in only documents relating to the *bird sense* of the word *crane*.

C For each query, classify each of the top 10 search results as correct or incorrect.

Again, I will be using Google for the queries.

C.1 Query: "large crane"

The query yields only one correct document: the second search result. The others documents relate to the wrong sense of the word. For details, see figure 1.

[1] `https://google.com/search?q=large+crane`

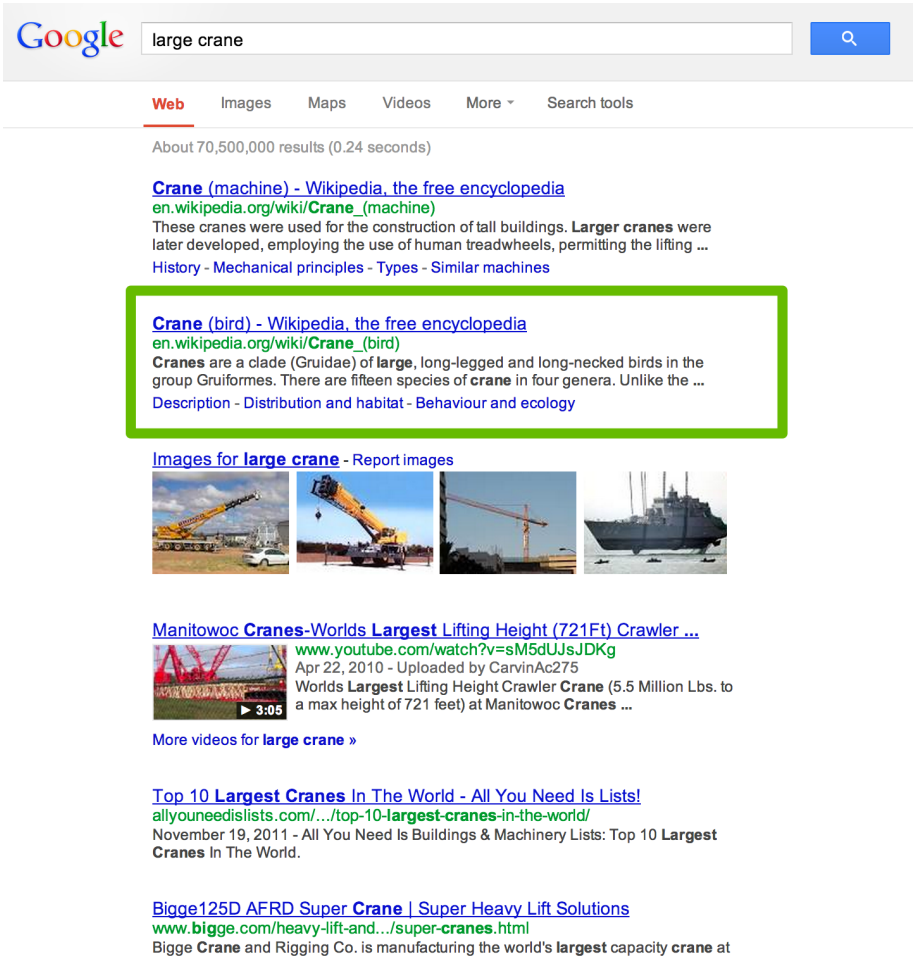

Figure 1: Google yields only one relevant search result when searching for "large crane".

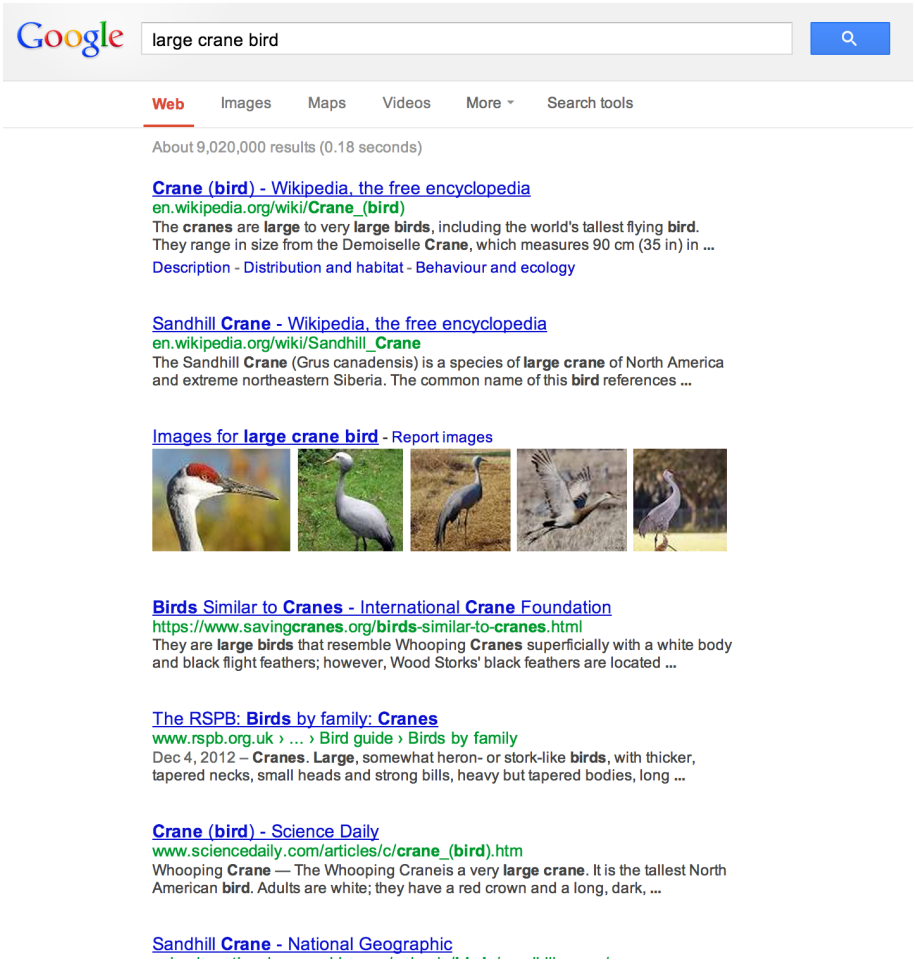

Figure 2: When adding the term "bird" to the query, all the top results are relevant.

C.2 Queries: "large crane bird" and "large crane feathers"

For both these queries, all the first 10 search results relate to the right sense of the polysemous word in Google.

Yahoo! and Bing, however, both give three documents relating only to feathers, which is not what is wanted.

D Create an interpolated precision-recall curve comparing the accuracy of each of the queries.

The recall values for the first query, "large crane", stagnates in that it gives only one correct document. Thus, the scales of the precision-recall curves do not overlap to a degree that makes it viable to plot them together.

However, the curves quite clearly show the improvement adding a disambiguating term yields, when the precision axis is normalized to one. See figure 3, 4 and 5.

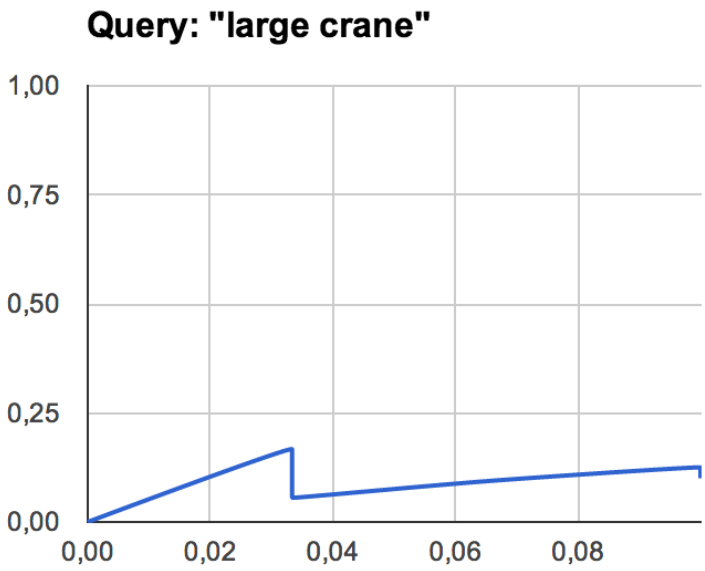

Figure 3: Precision-recall curve for "large crane".

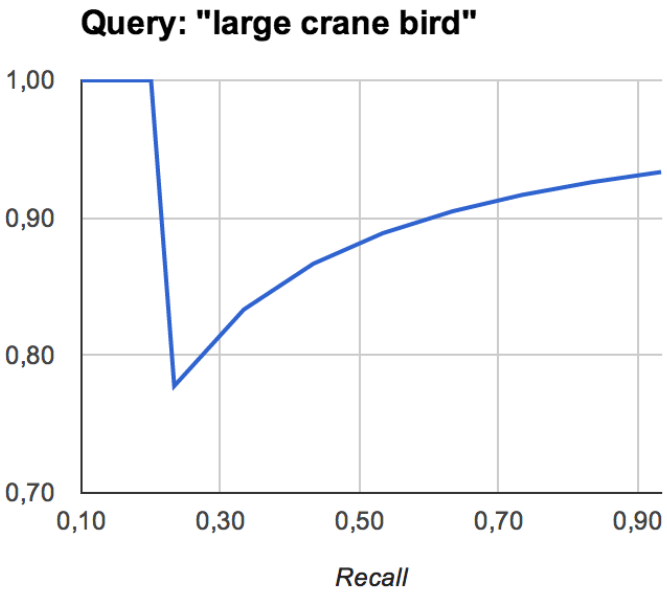

Figure 4: Precision-recall curve for "large crane bird".

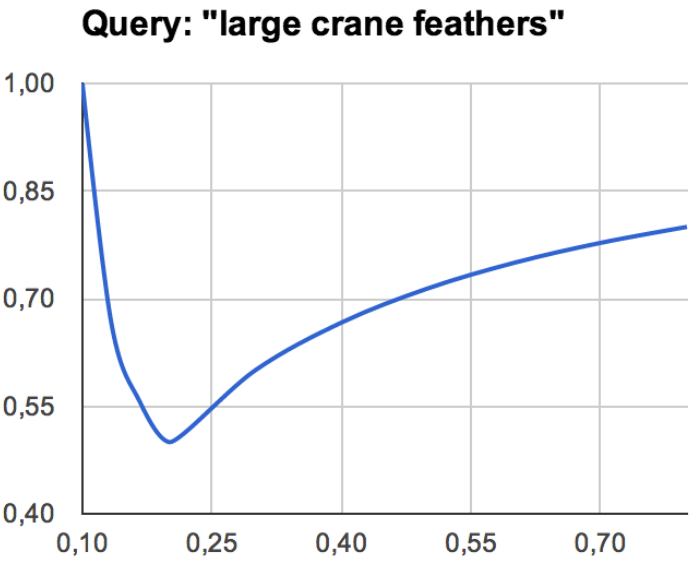

Figure 5: Precision-recall curve for "large crane feathers".

2 Information Retrieval

I went for a corpus traversed with Lynx from `http://www.sciencenews.org/`.

A The effects of the various steps described in the assignment.

Turn the corpus into text form. This really has no significant effect, as the corpus garnered from Lynx is already stripped of all HTML tags.

Turn the file into a list of words. Removes special characters, lowercases, and generally simplifies words into tokens, like this:

Before	Animals' cognitive shortcomings are as revealing as their genius.
After	animals cognitive shortcomings are as revealing as their genius

Build word frequency tables. This results in a series of *term frequency* files, of the following format:

```
0.00342987804878049     animals
0.00114503816793893     cognitive
0.00114503816793893     shortcomings
```

The number to the left denotes the term frequency of the word to the right.

Create your own stop list. This step is simply a matter of looking manually at some of the generated files and seeing which words are present in every document. These are manually extracted into a stop list.

The beginning of my stop list looks like the following:

```
a
all
the
of
and
in
for
on
or
```

Determine the terms' descriminative values. This step calculates the IDF – the inverse document frequency – of each word found in the tokenized document collection. The IDF routine simply outputs each word together with the inverse of the term frequency over the entire corpus.

Experiment with a simple search engine. Searching for "physical society" should probably lead to a document summarizing a recent meeting in the Physical Society.

Running `echo physical society | ../bin/search.perl -i idflist -e tfs` results in the following (please note that the numbers to the left are a result of some heavy playing around with weighting TF and IDF terms):

```
2430.451300474519204 lnk00000036.tfs
0.451300474519204 lnk00000037.tfs
0.204453275655133 lnk00000030.tfs
```

As expected, the raw document `lnk00000036.txt` is the document we were looking for.

Using a stoplist has no significant effect with the configuration used in the example above.

3 Text Summarisation

@TODO