Traitor - Associating Concepts using the World Wide Web

Lesley Wevers
l.wevers@utwente.nl

Oliver Jundt
s0213500@student.utwente.nl

Wanno Drijfhout
s0166960@student.utwente.nl

January 15, 2013
University of Twente

1 Idea

What do Apple, Samsung, Microsoft, IBM etc. have in common, if anything? What does Java have that PHP does not have? Which extra terms could refine the search query "`lance armstrong`"? What is the context of a sentence, such as "*Food, games and shelter are just as important as health*"? How can animals be organized in taxonomies?

You may know the answers to these questions or know where to look for them - but can a computer answer these questions for you too? We suspect this could be the case if the computer had a data set of associated *concepts*. An example to illustrate why we have this suspicion: if we know that the concepts of the two companies Apple and Samsung are both frequently associated with the concept of a "phone", we can conclude that "phone" is something they have in common.

Our idea is to test our suspicion with Traitor[1], a tool we developed that has a database of associated concepts and understands questions similar to our examples.

2 Method

In the following we describe how the two essential parts of Traitor, the association database and query interfaces, were created and work.

2.1 Part 1: Associated Concepts

We use Common Crawl's 25TB big data set of web pages to extract associated concepts. For this we have implemented a map-reduce program in Java for Hadoop. This program runs on SARA's Hadoop cluster and processes the web pages. The program depends on two assumptions we make:

1. A word represents a concept;
2. If two words co-occur in a sentence, the concepts they represent are associated by humans.

As we will discuss later (see section 4.1), assumption 1 has only a limited validity.

2.1.1 Mapper

In a few steps (see figure 1 or Javadoc[2]) our mapper transforms the raw HTML responses from the Common Crawl data set to a list of key value pairs (k, v), where the key k is a word pair (w_1, w_2) that co-occurred in some sentence on the page, and the value v denotes the number of pages on which this co-occurrence has been found. We consider the 'count' of co-occurring word pairs as a measure of the association's *strength*.

In sequence, the Traitor mapper performs the following steps:

1. discard documents that are not HTML-formatted or do not have status code 200 (OK);
2. extract plain text content from the HTML page using the BoilerPipe[3] library;
3. split the plain text into sentences using a regular expression;
4. discard sentences containing source code;

[1] See `http://evilgeniuses.ophanus.net` for a demo.

5. discard non-alphabetic tokens from sentences, such as numbers[2]

6. discard sentences that are "probably" not English[3]

7. count each distinct co-occurring word pair in the page's sentences once and emit the tuple $(w_1, w_2, 1)$ such that $w_1 \leq w_2$ (lexicographically).

We have chosen sentences as the semantic unit (rather than the same paragraph or document) for generating word pairs for two reasons. A technical reason is to constrain the result size. Pairing every non-trivial word with every other produces results in the order$O(n^2)$. The second logical reason is based on human language semantics. We assume that word co-occurrences within a sentence are more likely to represent actual concept associations than co-occurrences of words that are spread over a whole document.

2.1.2 Reducer

In the reduction phase we sum the counts of the word pairs produced by the mapper. Because the distribution of word pairs follows a Zipf distribution (see figure 2), we find that the majority of the resulting pairs have a very low count. To reduce storage cost of the final output, we can discard pairs with a count less than some N; e.g. if $N = 2$, we discard all pairs that only occur once. Essentially, this allows us to reduce the output to any size that we can handle in the presentation layer. Unfortunately, pairs with rare words are cut indiscriminately due to this policy, even if these co-occurring words are still usable associations.

2.2 Part 2: Query Interfaces

The resulting tuples from the map-reduce step are imported into a SQLite3-database. As a front-end to this database we have built a web application with two query interfaces using CherryPy and Jinja2.

[2]Consequently, Traitor does not process years, product model numbers and IDs etc.

[3]The heuristic to determine whether a sentence is English, is rather crude and imprecise: we check if a sentence contains at least two 'common typically English words'.

2.2.1 Association search with set notation

Users can enter search queries, similar to set notation, to do tasks like finding similarities and differences between two concepts.

A sequence of words, separated by spaces, denotes the conjunction/intersection and yields the words that are associated with all words in the sequence. A sequence of words, separated by plus-symbols, denotes the disjunction/union and yields the words that are associated with any word in the sequence. Words, preceded by a minus-symbol, denote the negation/complement.

To illustrate: the query `(a + b) -c` would yield all words that are associated with `a` *or* `b`, *and not* with `c`.

2.2.2 Visualizing similarity relationships of concepts

A graphical interface allows users to visualize similarity relationships of concepts in a *force directed graph* using D3.js[1].

Users can enter a list of words which become labeled nodes in the graph. The graphical size of the nodes indicates how many associations a concept has. The more associations a concept has, the bigger is the node.

For each word in the query the system retrieves the 50 most strongly related concepts. One could interpret this top 50 ranked list as an estimate of the concept's context.

For each word pair we apply the RBO metric[4] to estimate the similarity. A link is created between two nodes if the similarity is more than 5%. The length and thickness of the link indicate the similarity. If two nodes are connected by a short thick line the corresponding concepts share a very similar top 50 ranking. Conversely, distant nodes and nodes without any link between them have very few (or no) concepts in common.

3 Result

In the following we try to assess the quality of the Traitor's query results for our introductory questions and additional samples. The association database that we used for the query results contains over 43 million extracted distinct word pairs.

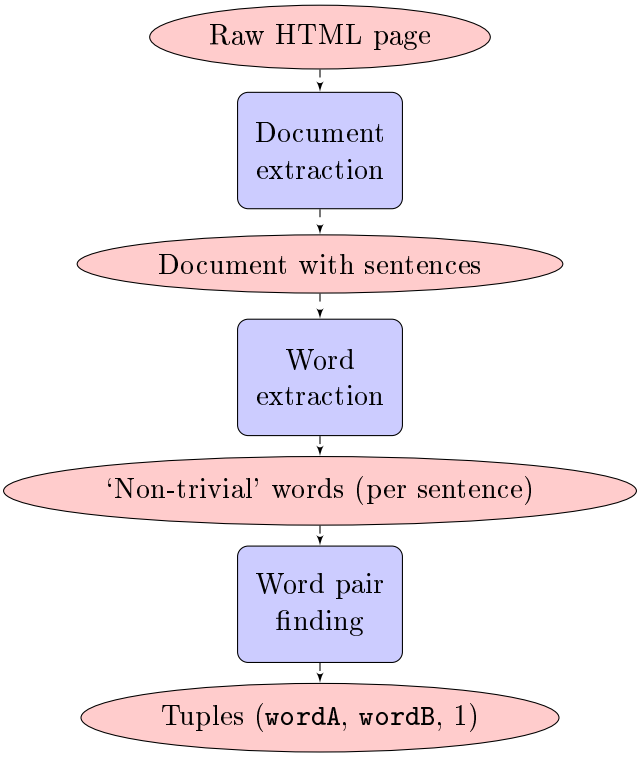


Figure 1: Description of the TRAITOR mapper workflow.

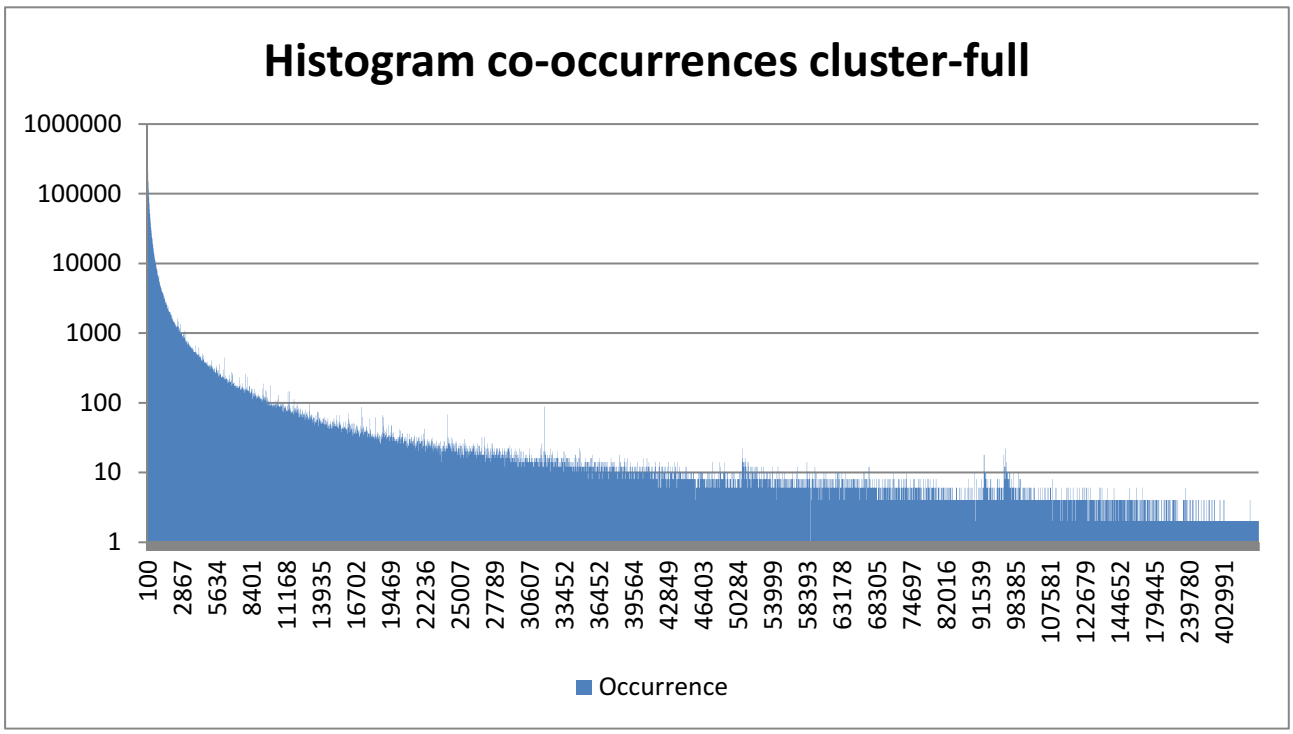


Figure 2: Histogram of word pair occurrences in the data set. Rare word pair occurrences (fewer than $N = 100$ times) are excluded.

3.1 Commonalities and recommending extra search query terms

The first question from the introduction translates in TRAITOR to "`apple samsung microsoft ibm`" (i.e, what do those companies have in common?) and yields as top ranked concepts: *software*, *windows*, *system*, *mobile*, *computer* etc. This is a reasonable result.

Recommending extra search query terms is similar to finding commonalities. For the query "`lance armstrong`", TRAITOR returns *tour*, *cancer*, *france*, *foundation*, *team* and (further down) *livestrong*, *race*, *cycling* and *yellow*. These are rather good results compared to Google Trends and allow the user to narrow down on a particular aspect of "`lance armstrong`", such as "`cancer`", eventually reaching *foundation* via "`lance armstrong cancer`".

Other queries of this type have results of varying quality, such as "`amber alert`" (yields *red* but also *girl* and *issued*, later on), "`political party`" (yields *information* but also *democratic* and *republican*), "`vehicle`" (yields *car*, *motor*, *insurance* etc.) and "`music`" (yields *video*, *live*, *download*, *world*, *play* etc.). We reckon that names and categories (like "`music`") generally yield better result than elements thereof ("`blues music`" yields *rock*, *band*).

3.2 Differences

The second question from the introduction translates to "`java -php`" (i.e., what associations does Java have that PHP does not?). The results are interesting: *applet(s)*, *alden*, *jvm*, *coffee*, *marketplace*, *east*, *concentration*, *flashcards*, *servlet*, *indonesia* etc. Some results are associated with the JAVA programming language (but not PHP!), the Java coffee brand and the Java island (East-Java).

If we extend the query to "`java coffee -php`", results include *cup*, *bean(s)*, *tea*, *espresso* etc. These results are reasonable answers to the query "concepts that are associated with Java and coffee but not PHP".

Other queries have results of varying quality; e.g., "`wii -xbox`" (contains *balance*, *kart*, *nunchu(c)k*) is good; "`gates -jobs`" (contains *melinda* as well as *wrought*, *driveway*), "`twente -holland`" (contains *fc*, *mcclarens*, *enschede* as well as *domestic*, *huge*) slightly worse. We reckon more words that are both commonly used and unambiguous (e.g., Wii) yield better results.

3.3 Deducing contexts of sentences

The association search interface can also be used for deducing contexts of sentences. For example, the query "`food + games + and + shelter + are + just + as + important + as + health`" produces the union of each word's associations: *care*, *insurance*, *information*, *play*, *good*. Taking the union[4] corresponds to the intuition that each word contributes extra meaning to the sentence.

For another query "`haskell + is + better + than + python + or + java`" TRAITOR yields reasonable results too: *code*, *language*, *application(s)*, *programming*, *software*.

Each word has its own associations. For any such association to emerge as 'context' of the entire sentence, the association must be very strong for either that single word, or for multiple words combined. We reckon the strongest associations describe the context of the sentence best.

3.4 Taxonomies

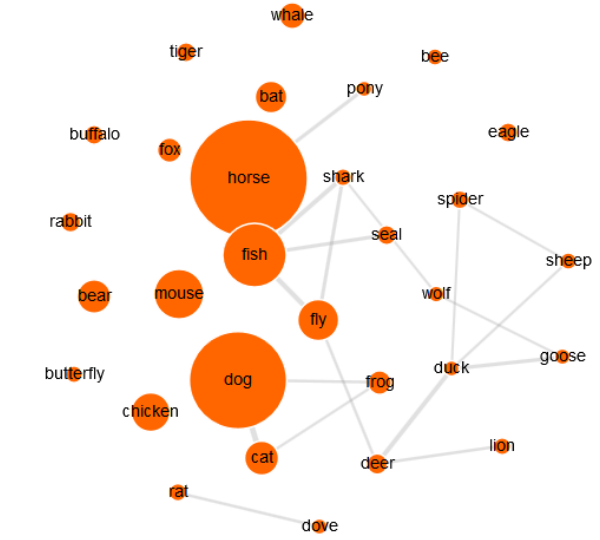
We created the visualization interface for finding taxonomies. This type of question seems to be the most complex one with widely varying quality of results. For the introduction example of animals (see figure 3a) the visualization is little insightful. However, the visualization of a set of brands (see figure 3b) illustrates the economic ties (e.g., competitors, similar sectors) between the brands quite well. The visualization of various countries (see figure 3c) resembles geographical, economic or historical relationships between them, but it requires knowledge of the user to understand what the visible clusters actually mean.

We reckon the set of brands is well-visualized because brands are often compared and reviewed on the web (e.g., product reviews, news reports).

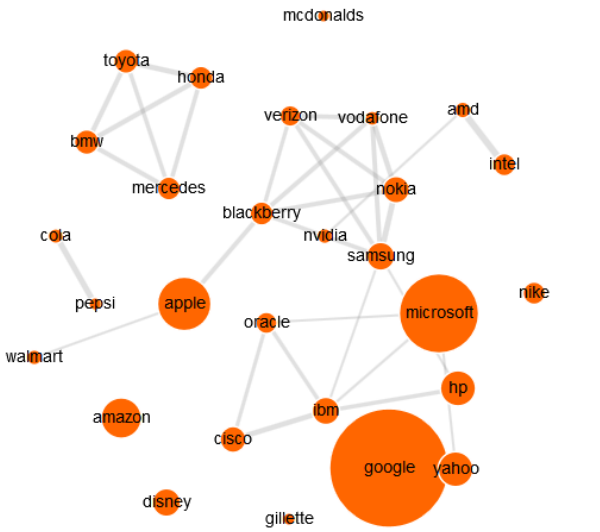
4 Conclusion & Discussion

Many of TRAITOR's results are indeed reasonable and as expected. However, as already discussed in the previous chapter, we observed cases where this is not the case. If we take a closer look at what would improve the result quality for most of these cases, we find the following problems that have to be solved.

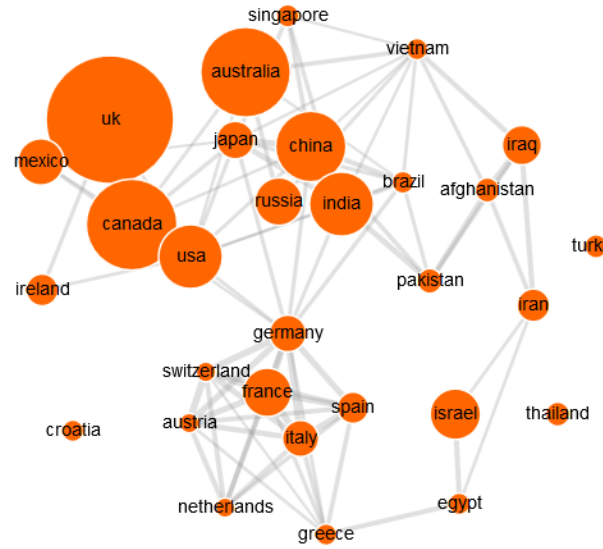
[4] The intersection would determine the words that are associated by all words in the sentence with the consequence that, if only one word is out of context, the resulting set of associations would be empty.



(a) Visualizing 30 animals.



(b) Visualizing 28 brands.



(c) Visualizing 30 countries.

Figure 3: Various demo visualizations

4.1 Ambiguous words

A weakness of Traitor is the assumption that each word corresponds to a single concept. Words often represent multiple concepts; e.g. "`apple`" refers to a company and a fruit. Traitor does not distinguish between different meanings of words and it consequently becomes very difficult to see which associations belong to which *meaning* in the query interfaces. In practice this is not a problem if there exists one dominant meaning of the word and the user is interested in exactly this meaning and only the most strongly associated concepts. Google Search has the same problem. If you search for "`apple`", you will have to click trough several result pages to actually find something about the *fruit*, because the company Apple is used much more often on the web. However, as with Google, combining concepts in a query can help to achieve the wanted results (e.g. "`apple fruit`").

4.2 Only one word to describe concepts

Another limitation of Traitor is that concepts can only be described (and identified) by single words. Sometimes, a proper concept description requires multiple words; e.g., "New York". This is a hard problem. We considered extracting bigrams and trigrams from sentences, but that would have drastically increased the database size. We also considered using predefined lists of n-grams that describe concepts, e.g. the Fortune 500 list of company names, but building a general and complete n-gram database turned out to be project by itself. Time constraints forced us to accept this limitation.

References

[1] Michael Bostock. D3.js. http://d3js.org/.

[2] Wanno Drijfhout, Lesley Wevers, and Oliver Jundt. Traitor documentation. http://evilgeniuses.ophanus.net/doc/.

[3] Christian Kohlschütter. boilerpipe. http://code.google.com/p/boilerpipe/.

[4] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4):20:1–20:38, November 2010.