“there are […] unknown unknowns
— the ones we don’t know we don’t know”
(not originally Donald Rumsfeld)

“the main dangers lie in the ‘unknown knowns’
—the disavowed beliefs […]
we pretend not to know about”
(Slavoj Žižek)

Most search engines try to find text that exists, but what about searching for text that doesn’t?

Lest someone hands me a random text generator, I should qualify that question – what about searching for text that isn’t written but perhaps should be?

Why would you want such a thing? To educate yourself or others perhaps. To help find the “unknown unknowns” and “known unknowns” quoted above. To escape the social media echo chamber. And so on (Paul Graham’s essay on what you can’t say seems relevant here). Even new scientific paradigms could be characterised as being on a journey through the murky fields of unknown-known [1].

There is of course, a lot of work that tries to address this aim by comparing the coverage of different media sources. But I’d like to demonstrate something more radical. Can we algorithmically analyze a single source and find topics it unnaturally avoids covering? What I’ve documented here is an attempt to do this. Call it a first draft, and feel free to improve upon it. The code is on github.

Does “cuba” make it more likely that “minister” will appear?

<table>
<thead>
<tr>
<th>Residue</th>
<th>Precedent</th>
<th>Consequent</th>
<th>Predicted prob, lift</th>
<th>Actual prob, lift</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuba</td>
<td>-&gt; minister</td>
<td>0.70, 11.33</td>
<td>0.09, 1.41</td>
<td></td>
</tr>
</tbody>
</table>

algorithm says yes, 11.33 times more likely than usual
in the real world, not nearly as likely... why is that?

The example shown here was an example result from processing the Reuters-21578 corpus, a
collection of about 19,000 news bulletins from the 1990s. The meaning of all the red and blue figures is explained in more detail below, along with the algorithm. The short story is that the algorithm thinks that when Cuba is discussed, then ministers should be discussed as well. In reality that doesn’t happen, so the algorithm flags it up.

Why did the algorithm make that prediction in the first place? Because there are lots of words which link Cuba to ministers. Some of these are so general as to be meaningless (“here”, “there”, “next”) but plenty aren’t, and that’s how probabilistic approaches work:

<table>
<thead>
<tr>
<th>Lift contribution</th>
<th>Link 1 - prob-lift -&gt;</th>
<th>Linking word</th>
<th>Link 2 - prob-lift -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.39</td>
<td>-0.35-20.26-&gt;</td>
<td>brazil</td>
<td>-0.25-4.01-&gt;</td>
</tr>
<tr>
<td>1.30</td>
<td>-0.26-4.09-&gt;</td>
<td>economic</td>
<td>-0.31-4.98-&gt;</td>
</tr>
<tr>
<td>1.20</td>
<td>-0.26-4.83-&gt;</td>
<td>countries</td>
<td>-0.28-4.59-&gt;</td>
</tr>
<tr>
<td>1.13</td>
<td>-0.35-3.59-&gt;</td>
<td>trade</td>
<td>-0.20-3.25-&gt;</td>
</tr>
<tr>
<td>1.10</td>
<td>-0.48-4.44-&gt;</td>
<td>years</td>
<td>-0.14-2.30-&gt;</td>
</tr>
<tr>
<td>1.08</td>
<td>-0.26-7.19-&gt;</td>
<td>here</td>
<td>-0.26-4.14-&gt;</td>
</tr>
<tr>
<td>1.05</td>
<td>-0.35-22.53-&gt;</td>
<td>soviet</td>
<td>-0.19-3.03-&gt;</td>
</tr>
<tr>
<td>1.01</td>
<td>-0.39-4.77-&gt;</td>
<td>month</td>
<td>-0.16-2.58-&gt;</td>
</tr>
<tr>
<td>0.90</td>
<td>-0.39-4.17-&gt;</td>
<td>added</td>
<td>-0.14-2.31-&gt;</td>
</tr>
<tr>
<td>0.90</td>
<td>-0.22-2.66-&gt;</td>
<td>foreign</td>
<td>-0.25-4.12-&gt;</td>
</tr>
<tr>
<td>0.84</td>
<td>-0.35-3.82-&gt;</td>
<td>against</td>
<td>-0.15-2.43-&gt;</td>
</tr>
<tr>
<td>0.84</td>
<td>-0.17-5.35-&gt;</td>
<td>conference</td>
<td>-0.30-4.84-&gt;</td>
</tr>
<tr>
<td>0.80</td>
<td>-0.22-3.77-&gt;</td>
<td>world</td>
<td>-0.23-3.70-&gt;</td>
</tr>
<tr>
<td>0.80</td>
<td>-0.37-4.46-&gt;</td>
<td>sugar</td>
<td>-0.09-1.41-&gt;</td>
</tr>
<tr>
<td>0.79</td>
<td>-0.17-3.58-&gt;</td>
<td>nations</td>
<td>-0.28-4.55-&gt;</td>
</tr>
<tr>
<td>0.78</td>
<td>-0.17-1.69-&gt;</td>
<td>government</td>
<td>-0.28-4.47-&gt;</td>
</tr>
<tr>
<td>0.75</td>
<td>-0.22-2.97-&gt;</td>
<td>next</td>
<td>-0.21-3.47-&gt;</td>
</tr>
<tr>
<td>0.75</td>
<td>-0.09-110.39-&gt;</td>
<td>envoy</td>
<td>-0.53-8.63-&gt;</td>
</tr>
<tr>
<td>0.74</td>
<td>-0.13-7.86-&gt;</td>
<td>policies</td>
<td>-0.35-5.68-&gt;</td>
</tr>
<tr>
<td>0.74</td>
<td>-0.26-2.76-&gt;</td>
<td>there</td>
<td>-0.17-2.82-&gt;</td>
</tr>
</tbody>
</table>

And why doesn’t it play out in practice? I don’t know. Maybe chance. Or could it be that, back in the 1990s, US media wanted to avoid words associated with democracy when discussing the Cuban government? Whatever your own political persuasion, I’m sure you’d agree that US-Cuba relations were pretty stony at the time. I’d like to think that the algorithm flagged up this case because that was reflected in media coverage. That would be quite exciting.

The rest of this article describes how the algorithm works, and provides a whole load more results for you to explore at leisure.

**The Algorithm**

Suppose we have a large corpus of text, divided into articles, such as a bunch of news reports, or Wikipedia pages. If an article is to be selected at random, every word in the corpus has a certain probability of appearing in the selected article.

But suppose we know the article already contains a specific word, such as “trade”. Now, the probability for all other words appearing in the article changes. We might expect to see words like “dollars”, “balance” and “export” appear more often than usual in articles that contain “trade”.

So, based on the corpus we have, we can compute conditional probabilities for all pairs of words. These are the blue figures shown above (though some are predicted not measured – I’ll come back to that). I’ll notate these as P(i|j), which translates as “probability of I appearing in the article, given that j appears in the article”. Computing all such probabilities results in a matrix of size \( n \times n \) where \( n \) is the number of unique words in the corpus [2].

Now imagine for a moment that we don’t know \( P(ij) \) for a certain pair of words. Can we predict it from...
the other information we have? Maybe. For each other word \( k \), we know the probability \( P(k|j) \) that \( k \) appears if \( j \) appears, and the probability \( P(i|k) \) that \( i \) appears if \( k \) appears. You could try multiplying and adding these for every possible other word \( k \) to get a guess at the probability which I will call \( P' \):

\[
P'(i|j) = \sum_k P(i|k) P(k|j)
\]

But wait! We can’t sum probabilities like that, because it’s possible that more than one of the other words could appear at once. How can we solve that problem? Remembering a hack you were probably taught at school, that “at least once” means the same as “not zero times”, let’s work out the probability of \( i \) appearing with \( j \) through association with \textit{at least one} intermediate word \( k \):

\[
1 - P'(i|j) = \sum_k 1 - P(i|k) P(k|j)
\]

This formula still isn’t right, because it assumes that the other words \( k \) occur independently of one another, which isn’t true either. Hold that thought, I’ll come back to it later. First let’s take a step back, remember that \( P' \) was only meant to be a naive guess in the first place, then ask ourselves if it’s a good enough guess.

How do we test whether one variable can accurately predict another? With a regression model. But let’s not regress \( P \) against \( P' \), because conditional probabilities aren’t actually very interesting. The biggest values of \( P'(i|j) \) will be for common words that appear everywhere. Instead, let’s convert probabilities \( P \) and \( P' \) to \textit{lifts} \( L \) and \( L' \). \( L(i|j) \) means the factor by which \( j \) increases the probability of \( i \), and is computed as \( P(i|j)/P(i) \). These are the red figures given above – again, as you now understand, both predicted and actual.

Also for the sake of simplicity, let’s ignore all other words \( k \) apart from the 10 that are most likely to link \( j \) to \( i \). In other words, the 10 largest values of \( P(i|k) P(k|j) \). This may seem odd, but as we go on it will make the results more comprehensible to us.

Bearing in mind that a lot of words don’t appear together at all – that is, our matrix of \( P(i|j) \) contains a lot of zeros – the most appropriate regression form might be a \textit{hurdle model}. That means two models: one to predict whether a word pair will appear at all, and a second model to predict how often it appears, for the times when it does appear.

Testing these models on the Reuters dataset, the first model doesn’t work well (discrimination is actually negative – I have no explanation for this!). The second model works alright, however: the correlation \( \rho \) between predicted and actual lift, given that the words do appear together in the first place, is \( 0.60 \).

So, it turns out that \( L' \) is actually a reasonably good predictor of \( L \). But how does this help us data mine for the taboo, or “what isn’t being said”? Well, let’s define “taboo” as a combination of words that the algorithm logically thinks should appear often, but doesn’t appear often in practice [3]. These combinations are \textit{outliers} in the regression model, so let’s look at the biggest outliers. To help us learn why they are outliers, let’s also look at words the algorithm used to link them. (This is why restricting the algorithm to 10 links is useful).

\textbf{Naive algorithm results}

There are a bunch of ways we could define “biggest outliers”, so I’m going to use the following two:

- Linear regression outliers: word pairs furthest from the regression curve
  - Underoccurring in practice (click to open)
  - Overoccurring in practice (click to open)
- Nonparametric regression outliers: word pairs which fall in a low quantile of \textit{L} but a high quantile of \textit{L'}
  - Underoccurring in practice (click to open)
  - Overoccurring in practice (click to open)
The results are tantalizing, but I don’t think I have sufficient knowledge of finance, nor business, to make sense of them. Maybe you can! I will note that a lot of “underoccurring” outliers are for word pairs that only occur once in practice: the results might be more intuitive if we insisted on a higher threshold, i.e. only considered word pairs that appear several times. But then again, if we were data mining for what isn’t said, aren’t the results likely to be unintuitive?

Conclusions
How can this be improved on? Firstly, the maths. Earlier I mentioned some issues with the assumption of independence of $k$, and said “hold that thought”. Ok, I actually wrote an extensive footnote about that, see [4] below.

Secondly, rather than looking for links between words, we could use other software to extract concepts from the text and find links between concepts rather than words. Intuitively that would filter out some noise from the results, as some results are obviously due to a confusion of words and concepts (“Puerto” and “Rico” appearing together more than expected, for example). You would, however, have to be careful how the concept extracting process works: if “concepts” are also defined through a matrix of conditional probability such as this one, then some odd interactions between concept extraction and taboo mining may happen.

Third, you could apply this to a different corpus of text. Funnily enough, I already have … on the entire English language version of Wikipedia. I’m pleased to report that the first part of the hurdle model works considerably better there, and is able to predict whether word pairs appear at all with a discrimination of about 22%. For pairs which do appear the correlation $\rho$ is 0.53. As to the results, I can’t understand them. Maybe it would help to split apart the fictional and factual content then have another go.

Finally, the obvious criticism here is that looking at the results is a bit like reading a horoscope, or worse, a Rorschach test! Nebulous concepts are vaguely suggested, and how the reader interprets them reveals more about the reader than the text. How can you prove this algorithm is any better than that? The fact that there is significant correlation between predicted occurrence of word pairs, and their actual occurrence, is certainly encouraging. Ultimately to prove the algorithm’s worth, though, a blind test is needed in which people read either algorithm output or randomly selected word pairs, and attempt to infer conclusions from them.

I wish I had more time for this, but alas I don’t, so having read this far – are you up for taking over the project? Unless you’re a cultural genius, you’ll probably think that the algorithm needs improving a bit to become useful, but if you can make the right tweaks, it would be very exciting.

If you want to discuss it with me please do, otherwise, feel free to dive into the source and get on with it. If not, well, thank you for reading this far nonetheless!

FOOTNOTES
[1] Please accept my apologies for a gross oversimplification of Kuhn.
[2] Actually we discard very frequent words (appearing in more than 12% of articles) because they don’t tend to relate to specific topics. We also discard very infrequent words (appearing fewer than 5 times overall) because they’re a sort of noise in the data. And to keep computation manageable, we trim the remaining word list down to the 20,000 most frequent.
[3] As we are using hurdle models we restrict ourselves to combinations that do appear somewhere, but don’t appear as often as we think they should.
[4] The assumption of all other words $k$ occurring independently is (I think) more formally stated.
as ∀x,y \{x,y\} \cap \{i,j\} = \emptyset \Rightarrow P(x|y)=P(x).

How can we fix this? An exhaustive calculation would need to sum the probability of every combination of other words k appearing. There are \(2^{(n-2)}\) such combinations so clearly this isn’t possible, but I imagine you could sample the space of combinations of k at random to get an estimate. This means computing the probability of each \(P(k_1 n k_2 n k_3 n \ldots|j)\) and each \(P(i|k_1 n k_2 n k_3 n \ldots)\). If you tried to measure these directly from the data, i.e. ditch the conditional probability matrix and see whether the combination actually exists, I imagine you would overfit the model, in the sense that most multi-word combinations would probably appear in either 0 or 1 articles. This means you would make great predictions but have no residuals from which to infer what is taboo.

I actually tried an alternative approach using a greedy algorithm. It works like this, for a maximum number of linking concepts \(m\):

1. Initialize prediction \(P'(i|j)\) to 0.
2. Make a temporary copy of \(P(k|j)\) for all k, let’s call this \(Q(k|j)\).
3. Repeat \(m\) times:
   1. Find the most probable word \(w\) to link \(j\) to \(i\), i.e. pick \(w\) to maximize \(x = P(i|w) Q(w|j)\).
   2. Add \(x\) to our prediction \(P'(i|j)\).
   3. Modify \(Q(k|j)\) to make it conditional on \(w\) not occurring, in other words set \(Q(k|j) = (1-x) Q(k|j)\) for all \(k\).
4. \(P'(i|j)\) now contains our prediction.

This is identical to the naive algorithm discussed above for \(m=1\), and I think (though haven’t proven!) that it’s mathematically correct for \(m=2\). As \(m\) gets higher, we over-correct for higher order terms e.g. the probability that the first two choices of \(w\) occur together. Still, it’s an estimate, not a precise calculation: think of words as appearing in groups, and if one word is used to predict a link from \(i\) to \(j\), then all words it occurs with get scored down to make their reuse as a predictor of an \(i\)–\(j\) link less likely.

The crucial question, of course, is how well the greedy algorithm for estimating \(P'\) performs overall? I tried on the Reuters corpus for for \(m=2, 3, 10\) and \(20\); in all cases performance is similar to the naive algorithm, sometimes worse, but for \(m=10\) it is better at \(\rho=0.62\).

Here are the greedy algorithm results, for the record.

- Linear regression outliers: word pairs furthest from the regression curve
  - Underoccurring in practice (click to open)
  - Overoccurring in practice (click to open)
- Nonparametric regression outliers: word pairs which fall in a low quantile of of \(L\) but a high quantile of \(L'\)
  - Underoccurring in practice (click to open)
  - Overoccurring in practice (click to open)