Media portrayal of terrorist events: Using computational text analysis to link news items to the Global Terrorism Database

by Kasper Welbers, Wouter van Atteveldt, and everyone who wants on it from the ResTeCo team

Development note

This vignette uses the development version of RNewsflow. I will update it on CRAN once I'm certain the new version has every feature needed for this vignette.

```
devtools::install_github('kasperwelbers/RNewsflow')
```

Current status

Close to done, with one hurdle:

- The precision and recall of the matches seems to be pretty high, and certainly high enough to work with.
- However, currently the attacks in London do not match with the threshold that gives the highest F1 score (according to the two validation approaches). The reason is probably that normally location is an informative indicator, but for The Guardian the UK and London are common terms, and so weight much less in the matching scores. Another reason could be that for events with relatively much coverage, the very rare combinations of terms that describe it best will also weight less overal.

The current version of this vignette reports some results as is, but keep in mind that this bias needs to be resolved.

Introduction

```
library(restecode)
```

This file demonstrates the method used for matching the terrorist events in the Global Terrorism Database to news articles. Both the GTD data and the articles from the Guardian used in the paper [@@] are available online, and instructions are included for replicating the analysis.[1] It is also possible to directly pick up our results (and trust us on the data preparation and matching algoritm), which are included in the package since these are not copyright sensitive. In that case you can skip the code until the validation section.

Certain steps of this analysis require a fair amount of memory. It is recommended to use a computer with at least 16Gb. This vignette can be performed step-by-step, but it will take a while to download the required data via the Guardian API. Furthermore, some steps take a while to compute, so it is advised to make back ups. We therefore first make a subdirectory (by default in the current working directory) to download the data and prepare the text corpora.

[1] It is possible that results differ if the GTD data or Guardian data has changed. We do not include the original data due to copyright.

```
dir.create('GTD_matching')
```

Downloading and preprocessing the data

GTD data

For this vignette we use the GTD data from 2000 to 2015. While the GTD data is openly available, it has to be requested from the GTD website, which involves filling out a form with basic usage information. Therefore, we do not include the GTD data in this package, but instead provide instructions and helper functions for preparing the GTD data for use our analyses.

The full data on the GTD website is provided as an .xlsx file, which can be read into R with the `read.xlsx` function from the `openxlsx` package.

```
library(openxlsx)
gtd = read.xlsx(xlsxFile = '[path to GTD file]')
```

The data can now be preprocessed into a Document Term Matrix (DTM). Here we use the `dfm` (document feature matrix) format from the `quanteda` package. The function `prepare_gtd` is provided in this package to take all the necessary steps. Here we prepare the data, using the fromdate argument to only include events after 2000-01-01.

```
gtd = prepare_gtd(gtd, fromdate = '2000-01-01', todate='2015-12-31')
```

The Guardian data

The Guardian has an API that provides free access to full texts. To access this API you first need to register for an API key.

```
api.key = "[paste your own api key here]"
```

We can access the API through the `GuardianR` package. For convenience we have made a wrapper function that downloads the data in batches to a given folder, and then creates the DTM when finished. This prevents having to download the data twice, which can take quite a while. The function can also be interupted. As long as the same query and from/to date are used, it will resume from the last downloaded batch.

The `download_guardian` function requires a query, or a character vector of query terms. For matching the GTD data we provide a list of terrorism query terms. The list is intentionally very broad. The goal is to have high recall (i.e. find all events that mention terrorist events) and we are not that concerned about precision (i.e. finding only events that mention terrorist events) in the data collection phase.

The following code downloads about 500k articles, which might take around a day to finish .

```
gua = download_guardian(terrorism_terms, api.key,
                        fromdate = '2000-01-01', todate='2015-12-31',
                        path = 'GTD_matching')

## Here we only use the articles from publication "The Guardian" (+/- 160,000)
gua = gua[gua$publication == 'The Guardian',]
```

The data can now be preprocessed into a DTM with the `prepare_guardian` function. Here we also have the `first_n_words` arguments, which can be used to only include the first `n` words of each articles in the DTM. We used n = 200 in our analysis, assuming that articles that cover new terrorist events mention the event early due to the inverted pyramid structure of news (i.e. core message on top). This also reduces biases due to article length.

```
gua = prepare_news(gua, doc_col='id', date_col = 'webPublicationDate',
                   text_cols = c('headline','body'),
                   docvars = c('webUrl','sectionName'),
                   first_n_words = 200)
```

Matching the GTD events to the news articles

Our task is to find out whether a news article covers a **new** terrorist attack (i.e. that happened only recently), and if so match it to the corresponding GTD event. To do this, we measure whether news articles are sufficiently similar to GTD events that occured within seven days before the article publication.

We measure the similarity of a GTD event and news article by looking at overlapping words and word combinations, weighted for how informative these words and word combinations are for distinguishing events. We adopt an established approach for measuring document similarity, in which documents are positioned in a vector space based on the occurence of word and word combinations, and the similarity of their positions is calculated.

In this section we describe additional preprocessing steps for creating a good (and memory-efficient) vector space, and our method for calculating the similarity. The required functions are published in the RNewsflow package that we developed, and that we have updated for this event matching task.

```
library(RNewsflow)
```

additional preprocessing for better matching

We use several additional preprocessing steps to prepare the DTMs for the similarity calculation. While some techniques have more state-of-the-art alternatives that might yield better results on the matching task, we have purposefully restricted ourselves to methods that can be performed directly within R and with DTMs as input.

Cluster terms with similar spelling

In a DTM, terms that have very similar spelling are nonetheless regarded as completely separate columns. To some extent, this is accounted for by preprocessing tech“niques such as stemming (applied in our case) or lemmatization. For instance, with stemming,”attacking“,”attack" and “attacks” are all transformed to “attack”. However, some words–in particular namems–can have different spellings, such as “Gaddafi” or “Gadhafi”. Moreover, some datasets have spelling mistakes. In the GTD data, there are quite many spelling mistakes and slightly different spellings for words, often related to differences between countries (though the prescribed language is English).

We therefore first use a technique for clustering words with similar spelling together as single columns. The steps are as follows. We first extract all the unique column names (i.e. words) from the gtd and news DTMs, which constitutes our *vocabulary*. We then apply the `term_char_sim` function to calculate which terms are sufficiently similar. By default, `term_char_sim` looks at overlapping tri-grams (i.e. three consequtive characters) with some additional restrictions.

```
voc = union(colnames(gtd), colnames(gua))
simmat = term_char_sim(voc)
```

The result is an adjacency matrix of the words in the vocabulary, which is 1 if words are sufficiently similar to be merged. We can now use this matrix to cluster all similar terms together, for which we provide the `term_union` function. Note that words are also clustered together if they are related indirectly. That is, if A and B are related, and B and C are related, Then a cluster ABC is created even if A is not related to C.

```
gtd = term_union(gtd, simmat)
gua = term_union(gua, simmat)
## (be carefull not to run this twice, because it overwrites gtd/gua)
```

This clustering approach is intentionally greedy. The vector space that we're creating is extremely sparse (as detailed below), so the chances that a different but similarly spelled concept causes a false positive are slim.[2] Still, it is always good practice to manually check how the clustering performs by looking over some of the new columns. The new column names contain all unique terms, connected with the | (OR) operator.

```
cnames = colnames(gua)
head(cnames, 20)
```

Compute term combinations

For our matching task, we want to use the features (i.e. terms) that are most informative, i.e. that most clearly distinguish events. Very common terms do not tell us much about whether news articles cover the same event as described in a GTD entry. However, some terms such as "London" and "attack" are not very informative on their own, but very informative combined: the London attacks.

Simple vector space calculations of similarity do not take into account that the combination of "London" and "attack" together in the same article is more informative than the sum of their individual information. We therefore provide a function to add term combinations to the DTM. This is not a common thing to do, because it greatly increases the number of unique terms in the DTM (a quadratic increase) which is often not feasible due to memory limitations. However, it is possible if we impose a few assumptions regarding our specific matching problem and filtering the term combinations accordingly.

assumptions:

- We drop all columns of which the document probability (docprob) is higher than 0.01. That is, we ignore all terms and term combinations that occur in more than 1/100 documents.
- We only keep combinations if terms if their observed frequency is more than 1.2 times the expected frequency. The expected frequency is the number of times we would expect two terms to co-occur based on sheer probability. We are not interested in term combinations that only occur due to sheer probability of co-occurence.

Note that these values can easily be altered, and which values are best for different use cases is an open question. We have chosen the current values based on common sense and experimentation. As discussed below, there is a good way to get quick validation measures for our type of event matching task, and this can be used to get good estimates.

```
sf = create_queries(gtd, gua, max_docprob = 0.01, min_obs_exp = 1.2,
                    min_docfreq = 1, use_dtm_and_ref = F, verbose=T, weight = 'tfidf')
```

The output of `create_queries` is a list with the revamped gtd and gua DTMs, now called `query_dtm` and `ref_dtm` (reference). The query dtm is in this case the gtd data. Both DTMs contain the same columns. For the `query_dtm` the values are weighted using a tfidf weighting scheme (as specified in `create_queries`) and normalized so that the max value is 1. For calculating the idf only the document frequency scores of the reference dtm (i.e. news articles) are used, because we do not want to weigh down common terrorism query terms in the GTD. The values in the `ref_dtm` are binary: does the term or term-combination occur in a document?

The reason for this way of formatting the data is that we can now sum up the weights of the query terms (GTD) that occur in documents in the reference DTM (Guardian). In the `compare_documents` function that we use below for calculating the similarity scores for GTD and Guardian documents, we provide the

[2]Note that later on we will weight the DTM based on overall term frequencies, so terms in very greedy clusters with and/or with common words will mostly become less informative, rather than result in false positives.

query_lookup measure. This is an assymetric similarity score that is not normalized by the total sum of query terms or the sum of terms in the reference documents. This is important, because we are only interested in whether at least a minimum amount of weighted query terms occur.

Performing the comparison

The compare_documents function is a specialized function for calculating the inner product of DTMs. It has several important benefits, but two are of particular importance for the task of matching event data to news articles.

- Events and news occur over time. We can leverage this to only compare events and news within a given time window. For large datasets this is critical, because it greatly reduces the number of comparisons. In our current case there are 87,338 GTD events and 161,049 Guardian articles, amounting to 14065697562 comparisons. Even if time is not an issue, you would need a huge amount of memory. The compare_documents function uses a custom built matrix multiplication algorithm that incorporates an efficient sliding window approach. The hour_window function takes a numeric vector of length two that specifies the left and right borders of the window.
- Even with fewer comparisons, the total number of comparisons quickly amounts to a number of non-zero results that would require a huge amount of memory. For our current task we can set a threshold for the minimum similarity score, which cuts of the vast majority of these results. The compare_documents function performs the calculation of the similarity measure and applies the given threshold (min_similarity) within the matrix multiplication algorithm.

The first two arguments are the two DTMs that are to be compared, in our case the query_dtm (GTD) and ref_dtm (Guardian). We also specify the time window in hours. Although we are only interested in matches within seven days after the GTD event, we also look for all matches within a window of 14 days before the event. This allows us to do a preliminary validity check, as discussed below. The min_similarity of 1 is a very low threshold, as shown in the validation analysis, to have a high recall. As the measure we use the query_lookup as discussed above.

```
g = compare_documents(sf$query_dtm, sf$ref_dtm,
                      date_var='date', hour_window = c(-14*24,7*24),
                      min_similarity=1, measure = 'query_lookup', verbose=T)
```

The resuls of the comparison are not copyright sensitive, so they are provided as data in the package. Here we load these results (overwriting g) so that they can be used for the remainder of the vignette.

```
g = gtd_comparison_results
```

The output contains an edgelist data.table d in which each row is a match, with columns indicating the GTD event id, Guardian article id, weight of the match, and the hour difference. It also contains the document meta information and certain additional attributes of the document similarity comparison. This is stored separately for from and to matches, in our case GTD events (from_meta) and Guardian articles (to_meta).

```
head(g$d)
```

from	to	weight	hourdiff
201108140006	world/2011/aug/14/afghan-governor-taliban-raid	171	0
200912150002	world/2009/dec/15/hamid-karzai-afghanistan-corruption-conference	162	0
201304180001	world/2013/apr/19/boston-lockdown-armed-police-hunt-suspect	155	24
201411180004	world/2014/nov/18/jerusalem-synagogue-attack-witnesses-scenes-horror	154	0
201501090002	world/2015/jan/11/paris-gunman-amedy-coulibaly-allegiance-isis	150	72
201501090001	world/2015/jan/09/charlie-hebdo-attack-suspects-gunman-killed-dammartin-en-goele-port-de-vincennes-paris	149	24

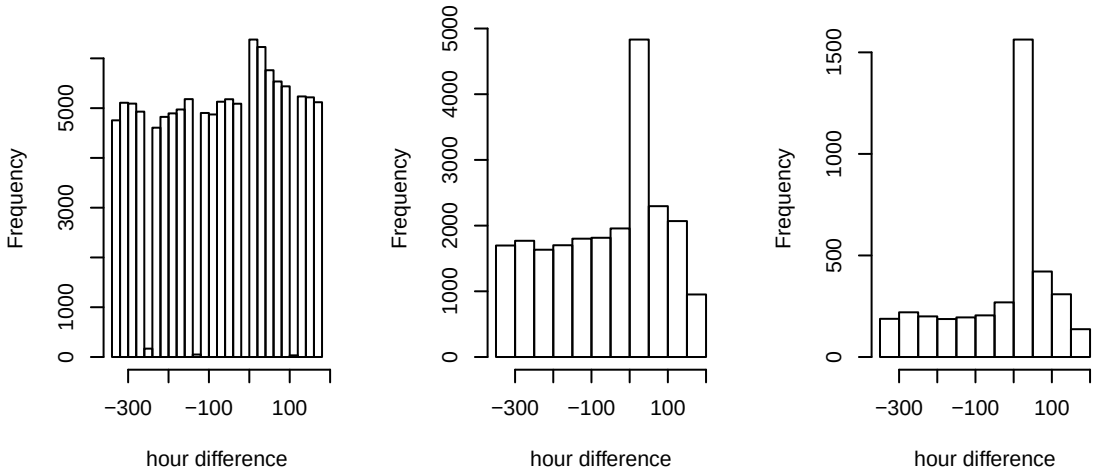
```
head(g$from_meta[,c('document_id','date','geo','country','type')])
```

document_id	date	geo	country	type
200001020001	2000-01-02	de	Germany	Armed Assault
200001020002	2000-01-02	dz	Algeria	Armed Assault
200001020003	2000-01-02	co	Colombia	Bombing/Explosion
200001020004	2000-01-02	co	Colombia	Bombing/Explosion
200001030001	2000-01-03	na	Namibia	Armed Assault
200001030002	2000-01-03	lb	Lebanon	Armed Assault

Validation 1: inspect matches over time

To get a first indication of the validity of the matches, we can look at the histogram of the hour differences. In particular, we can look at the number of matches before the event to get an estimate of the number of false positives. That is, if a GTD event matches a news article that was published before the actual event, we can be certain that the match is a false positive. Here we plot three histograms for different similarity weight thresholds to see how this affects the matches.

```
par(mfrow=c(1,3))  ## plot side by side
for (thres in c(5,10,20))
  x = hist(g$d$hourdiff[g$d$weight > thres], right = F, main='', xlab='hour difference')
```



In all three cases we see a peak after the an hour difference of zero, which indicates that the algorithm picks up true positives. However, we also see that there are false positive, especially for the lowest threshold. This is a good sign, because it means that we get a better precision if we increase the threshold. If we increase the threshold to a point where there are only a few matches before the zero hour difference point, we can be confident that we have a high precision.

However, if our threshold is too high, this will greatly harm the recall. If we look at the second and third histogram, we see that the peak decreases from to 4832 to 1563. While a portion of this decrease is likely to be due the loss of false positives, it is very likely that we have lost many true positives as well.

If we assume that the probability of false positives is the same for matches before and after the event, we can calculate a rough estimate of the precision and recall. While the exact estimates are not very accurate, we can them to compute which similarity threshold gives the highest F1 score. The estimation works as follows.

Estimating P/R

Let all possible event-to-news pairs within the comparison window be denoted as N, which we split into pairs where the news is published before the event N_{before} and news published after the event N_{after}. Based on a matching algorithm we then count the matches N, which we also split into matches before the event M_{before} and matches after the event M_{after}. We can now calculate the probability of a match before as:

$$P(M_{before}) = \frac{M_{before}}{N_{before}}$$

Based on the fact that matches before the event are false positives, and the assumption that the probability of a false positive match is the same for matches after the event, we can estimate the number of false positives in matches after the event. We consider this to be the estimate of false positives $\hat{FP}$ in our results.

$$\hat{FP} = P(M_{before}) \times N_{after}$$

Now, the difference between the observed matches and estimated false positives gives the estimated true positives $\hat{TP}$, and so we can also estimate the precision $\hat{P}$.

$$\hat{TP} = M_{after} - \hat{FP}$$
$$\hat{P} = \frac{\hat{TP}}{M_{after}} \times 100$$

To estimate the recall, we first use our lowest similarity threshold to estimate $\hat{TP}$ for a set of matches of which we are confident that the recall is close to 100. We then use this estimate as the upper limit TP_{limit}. We can then calculate the recall estimate $\hat{R}$ as the percentage of this limit for the results of higher thresholds.

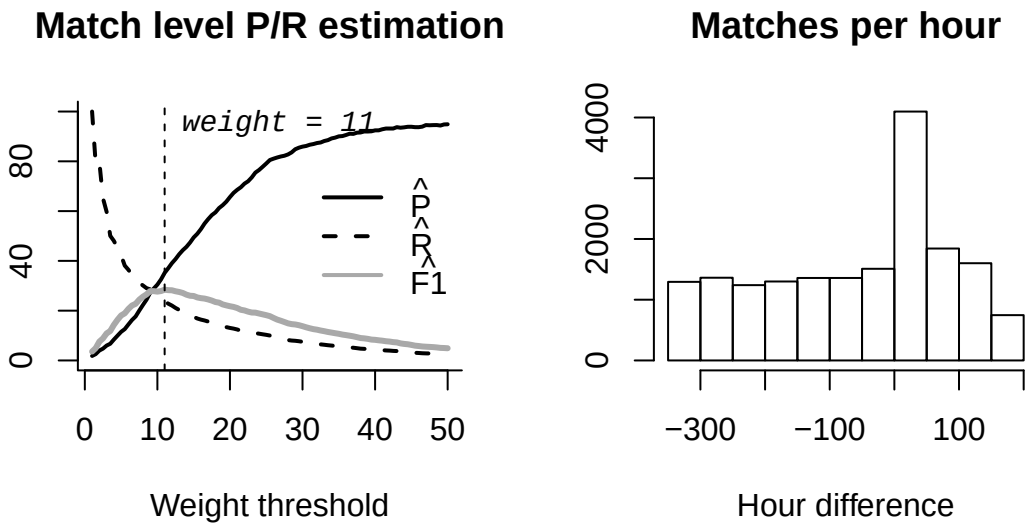
$$\hat{R} = \frac{\hat{TP}}{TP_{limit}} \times 100$$

Finally, the estimated F1 $\hat{F1}$ is the harmonic mean of the $\hat{P}$ and $\hat{r}$.

$$\hat{F1} = \frac{\hat{P} \times \hat{R}}{\hat{P} + \hat{R}}$$

The following function estimates the precision, recall and F1 scores for a range of similarity thresholds.

```
res = estimate_validity(g)
```



As an initial validation, this methods shows that the matching algorithm is able to identify correct matches, and confirms that by increasing the threshold the precision increases. Note that the exact values cannot be interpreted as the true precision and recall of the matching task (our manual validation indicates that these these are much higher). Nevertheless, they provide a way to compare different settings, such as the weight threshold, for finding good results.

Validation 2: Gold Standard

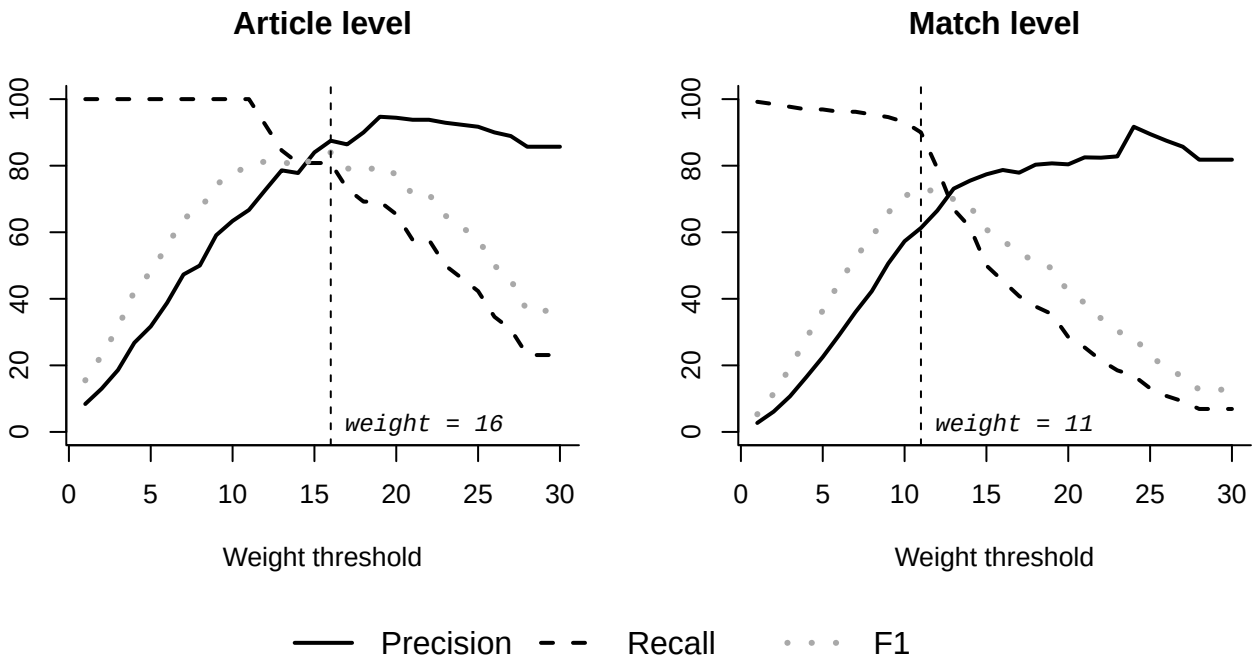
As a second method of validation, we therefore also created a gold standard. We manually coded 500 random articles from The Guardian between 2000 and 2015 for the broad terrorism query. We first looked whether the a news article covered a `new` terrorist event. If this was the case, all GTD events within 7 days before the article publication date and that took place in the same country as mentioned in the article were manually checked for a match.

This gold standard data has later been improved based on results of the matching algorithm. We manually coded false positives of matches to see what went wrong, and found that in some cases the matches were actually correct (i.e. the article indeed covered the GTD event). In other words, there were matches that we did not find when manually creating the gold standard. The main reason is t hat there were some cases where a single news article could be matched to a large number of GTD events, and this was not always immediately obvious from the news article. An example is a news article that reported generally that there had been several bombings in the past week. While these adjustments improved the gold standard, it should be noted that this indicates that our process of manually creating the gold standard is not flawless. Despite our best efforts, the process of manually linking GTD events to news is not only time consuming but also prone to error.

In total, we found that 26 out of the 500 articles matched with a GTD event. Since the GTD documents each event separately, some articles matched multiple GTD events. In total, 130 matches were found. The gold standard data is included in the `restecode` package, under the name `gold_matches`.

The `pr_plot` function uses this data to calculate the precision, recall and F1 (weighted average of precision and recall) scores. The data frame `g` is subsetted so that it only includes matches of GTD events to news articles within 7 days after the event, and only includes news articles that are in the gold standard (N = 500). The precision, recall and F1 scores are calculated separately for the article level (does the article have at least one GTD match) and match level (each individual match of an article to a GTD event). Furthermore, the scores are calculated for different weight thresholds.

```
pr = gtd_pr(g)
```



The results are promising. For both the article level and match level the recall remains very high up to a threshold around eleven, at which point precision has sufficiently climbed up. For the match level the highest F1 score is found at a weight threshold of eleven. For the article level the highest F1 score is sixteen due to a steeper increase of precision, but the F1 score is almost as high at threshold eleven, and here the recall is notably better.

It is interesting that the estimated precision and recall based on matches over time also suggested a threshold of 11. While the exact same estimate should be attribute in good part to chance, this does strengthen our confidence that this is a good threshold for our data.

Improving matches with postprocessing

Given the results of `newsflow_compare`, we can still apply certain postprocessing steps to improve the results. Note that above te assigned the result of `gtd_pr` to `pr`. We'll use this below to see whether and how strongly the postprocessing improves the results.

Postprocessing 1: filtering on country

The matching algorithm already uses location information for calculating the similarity score. However, we can more strictly decide that a GTD event and news article need to match on the country level. In the current data preparation functions (`prepare_gtd` and `prepare_news`) we have added the `geo` column to the DTM document variables using the `newsmap` package and an adjusted country dictionary based on the locations used in the GTD. For the GTD data this is a two-letter code for the country[3].

```
head(g$from_meta[, c('document_id','geo','country','city')])
```

[3]If the data is not created with the `prepare_gtd` or `prepare_news` functions, you could use the `geo_tags` function to create the geo tags.

document_id	geo	country	city
200001020001	de	Germany	Erfurt
200001020002	dz	Algeria	El Omaria
200001020003	co	Colombia	Bogota
200001020004	co	Colombia	Unknown
200001030001	na	Namibia	Katima Mulilio
200001030002	lb	Lebanon	Beirut

For the news data the country is harder to determine, because a news article can mention other countries besides the country in which a terrorist event occured. Therefore, the `geo` column here contains the three top-ranked country matches per document.

```
head(g$to_meta[, c('document_id','geo')])
```

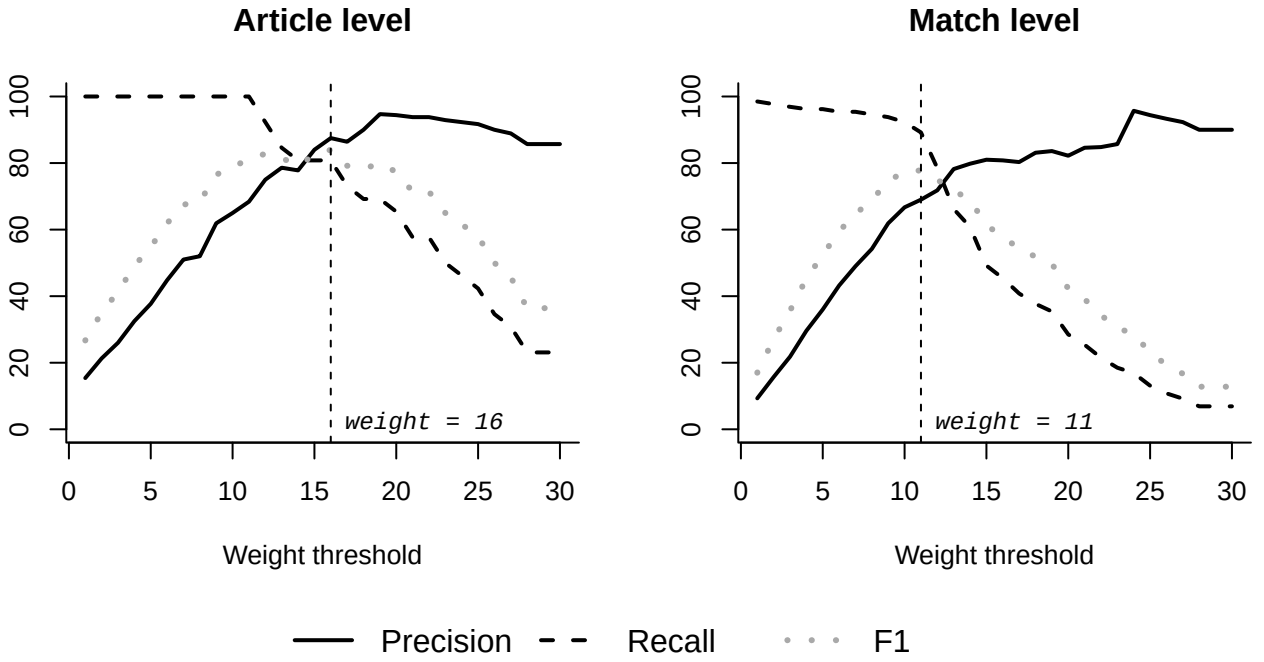
document_id	geo
world/2000/jan/07/warcrimes.germany	lt,us,ba
theguardian/2000/jan/07/features11.g2	in,us,br
film/2000/jan/07/culture.features	fr,cl,mx
society/2000/jan/05/futureofthenhs.health	NA
theguardian/2000/jan/04/features11.g23	se,us,af
books/2000/jan/01/books.guardianreview20	us,ie,za

The `match_geo` function uses this information to determine whether the two documents in a match also match on their geo codes.

```
g$d$geomatch = match_geo(g)
```

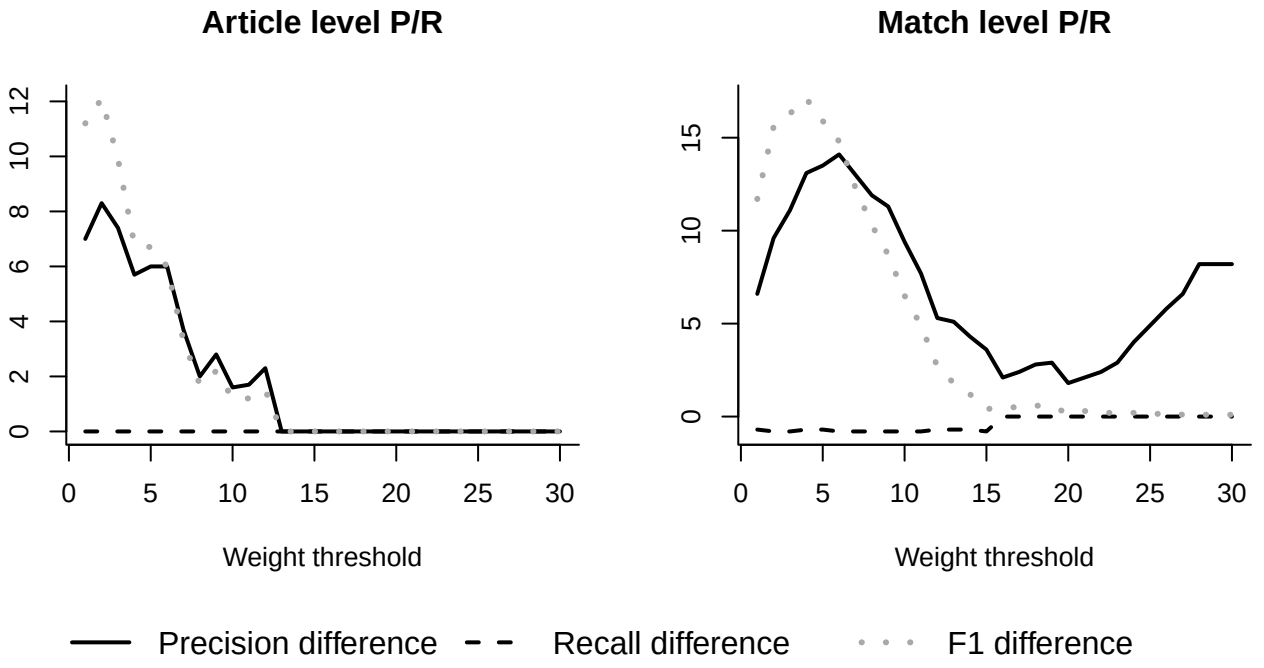
The `geomatch` column that has nog been added is a logical vector. We can now what the precision and recall look like if we filter on matches where geomatch is TRUE.

```
pr_geomatch = gtd_pr(g, filter=g$d$geomatch)
```



Eyeballing the plotted results shows that the precision increases and recall slightly decreases. We can look at this more specifically by comparing the `pr` and `pr_geomatch`. The `compare_pr` function

```
pr_comp = gtd_compare_pr(pr, pr_geomatch)
```



The increase in precision is in particular strong in the weight range from two to ten. If we use a weight threshold of eleven, the article level F1 score only increases by 1.2, and the match level F1 score increase by 4.9. Thus, in our case it is only a moderate improvemnt. However, matching on country level would be a very usefull strategy if the goal is to combine automatic matching with manual coding to achieve very high precision and recall, in which case it would make sense to use a lower threshold to get higher recall for the price of lower precision. In these cases, matching on country has a very strong relative improvement in precision, which means that much fewer document pairs need to be manually coded. For instance, with a threshold of eigth, the number of matches decreases by around 40%

```
sum(g$d$weight > 8)
```

```
## [1] 39094
```

```
sum(g$d$weight > 8 & g$d$geomatch)
```

```
## [1] 23375
```

postprocessing 2: clustering of GTD events

Terrorist attacks often encompass multiple events, such as multiple bombing targets on the same day. The GTD documents these events separately, but news media often report on a string of connected attacks. In some cases the number of separate events is rather large. In our gold standard, one news article matches 40 GTD events, and another matched 21 events. To better match connected attacks, we cluster the GTD data. If a GTD event in a cluster matches a news article, we then also match all other events in the cluster.

Still to be implemented and tested. Should increase recall for match level, which could make it possible to increase threshold for better precision.

Analysis

Gatekeeping function

Compare descriptives of GTD events that are and are not covered in the news.

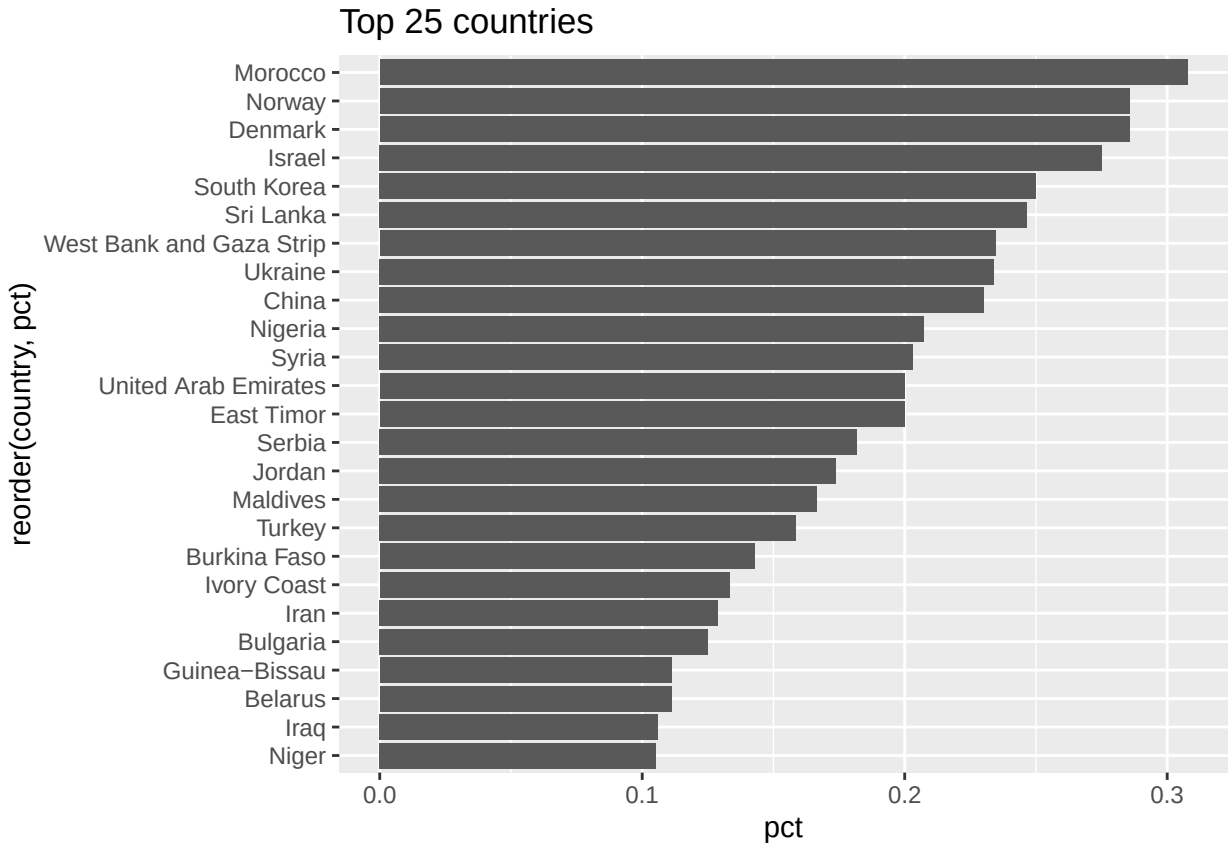
Get meta data for all GTD events used in the comparison (2000 to 2015, minus the a few for which no full comparison window was available). Aggregate edges to count how many news articles were matched per event.

(I will probably make functions for extracting the data to make it easier to see what I'm doing, but for now I'll put all the code here)

```
library(data.table)
meta = subset(g$from_meta, .complete_window)
e = subset(g$d, weight >= 11 & geomatch)
e = e[, list(N_news=length(weight), has_news=T), by='from']

e = merge(meta, e, by.x='document_id', by.y='from', all=T)
e$N_news[is.na(e$N_news)] = 0
e$has_news[is.na(e$has_news)] = 0
```

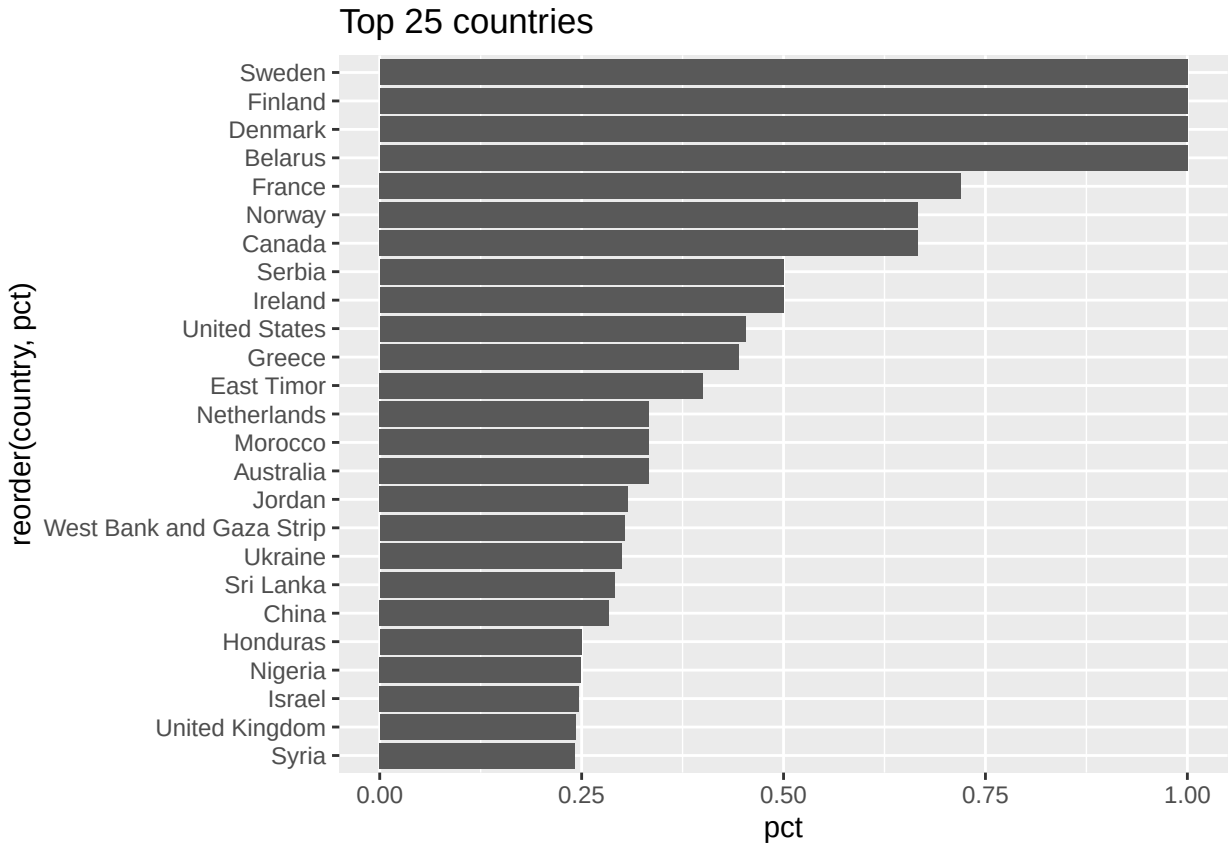
Lets start with the percentage of GTD events per country that received coverage

```
library("ggplot2")
e_c = e[,list(pct = mean(has_news)), by='country']
setorder(e_c, -pct)

ggplot(head(e_c, 25) ,aes(x=reorder(country,pct),pct)) +
  geom_bar(stat ="identity") + coord_flip() + ggtitle('Top 25 countries')
```



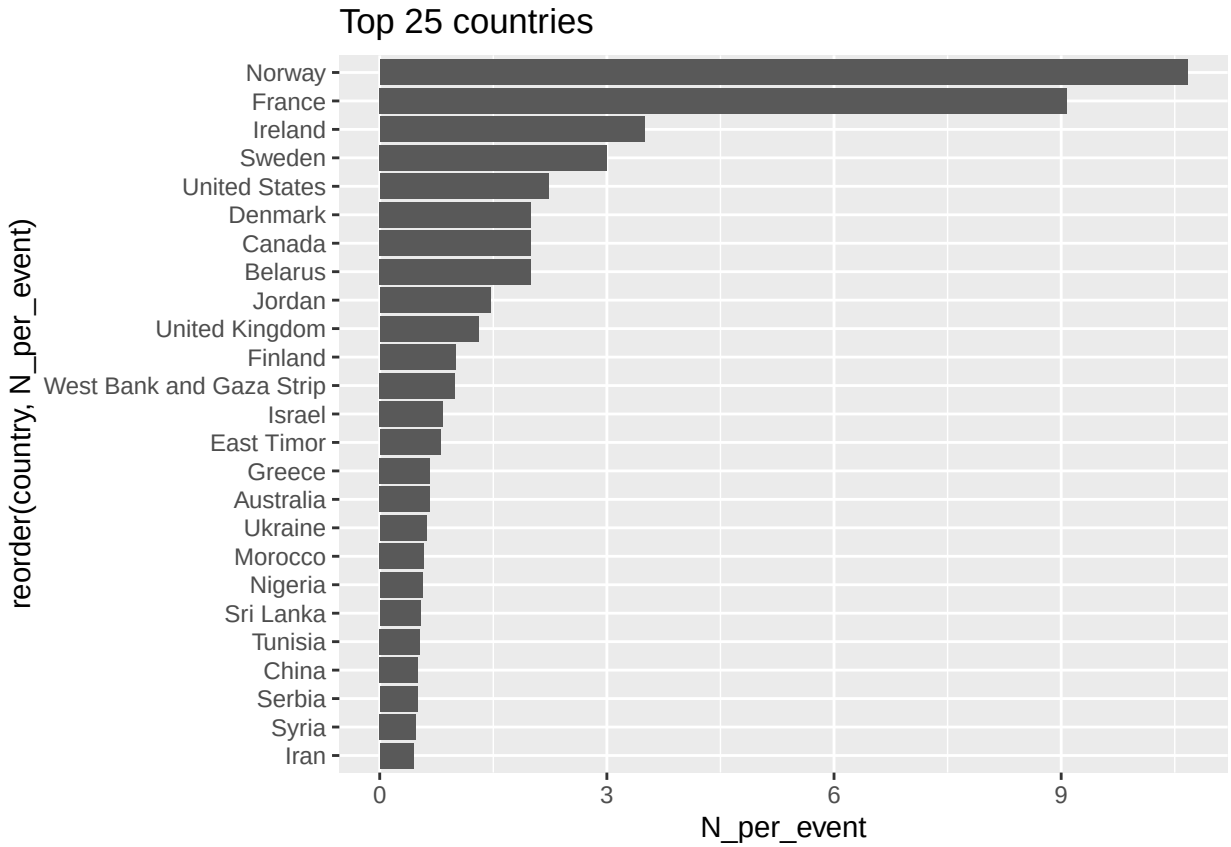
I would have expected to see US higher here, but we only find about 10% of US terror attacks This is probably related to what counts as a GTD events. We could instead look only at cases with >0 casualties.

```
ke = subset(e, killed > 0)
ke_c = ke[,list(pct = mean(has_news)), by='country']
setorder(ke_c, -pct)

ggplot(head(ke_c, 25) ,aes(x=reorder(country,pct),pct)) +
  geom_bar(stat ="identity") + coord_flip() + ggtitle('Top 25 countries')
```



Now this is more like it (sorry, that sounds reeeeaaallly dark given this context). Now lets take the amount of coverage into account. The following code shows the average number of news articles per event, again using only events with casualties.

```
ke_nc = ke[,list(N_per_event=mean(N_news)), by='country']
setorder(ke_nc, -N_per_event)

ggplot(head(ke_nc, 25) ,aes(x=reorder(country,N_per_event),N_per_event)) +
  geom_bar(stat ="identity") + coord_flip() + ggtitle('Top 25 countries')
```



There are some obvious cases here, but US and UK are still lower than I expected. I think this is due to the issue discussed above, that countries that generally receive more news (any news in the population query) are less likely to match due to the tf-idf weighting. I'll have to tinker around with this.

The semantics of "terrorism"

Compare descriptives of GTD events for which the news coverage does and does not contain the word "terrorism"

Compare word frequences in news coverage that does and does not contain the word "terrorism"

SImilarly, look at frequencies of the search terms used in the population query

Conclusion