We only have access to a big data set via boolean search.

Sub-sets of the data set returned by a single key word on a single day are large.For example for the single search term 'election' for the single day of 1 August 1868 returns 1982 articles. The search term 'riot' for the same day (close to the 1868 General Election) returns 360 articles, 228 of which do not also contain the word election.

With even a small number of search terms it quickly becomes impractical to examine all the documents even for a single day.

Following King et al we define S - the search set of all documents in the British Newspaper Archive T - the target set of all documents in the British Newspaper Archive which are about election violence R - a reference set of documents which are about election violence

The task is to identify T from S in a form where T can be

It is trivial to define an algorithm which obtains a subset of S which contains T, because $S \subseteq S$ and $T \subset S$. Algorithms which aim to maximise the chances of obtaining all of T will tend to return S.

Our task is to find a good method for returning T from S in a form which we can analyse. By a *good* method we mean a method which returns a greater proportion of T, and a greater ratio of T to $\neg T$ than alternative methods. The main alternative method is manual searching by historians.

1 The data

```
classdocs <- durhamevp::get_classified_docs()
classified_corpus <- quanteda::corpus(classdocs[, c("fakeid", "ocr", "EV_article")],
    text_field = "ocr")
classified_dfm <- preprocess_corpus(classified_corpus, stem = FALSE, min_termfreq = 20,
    min_docfreq = 10)
```

In the data there are 1894 cases:

- **703** non-election articles.
- **769** election violence articles.
- **422** election (but not violent) articles.

2 Keyword Identification

Note it is important to make keyword identification somewhat selective of T from S otherwise even very good stage 2 & 3 selection processes the false positives will overwhelm the true positives.

Algorithm:

1. use classifier on R and S to identify two lists of keywords

2. generate probability from classifier parameters
3. add to keyword list based on probability

```
keywords<-nb_keywords(classified_dfm, "EV_article")
knitr::kable(head(keywords, 20))
```

	rowname	0	1	id
12299	roughly	0.0277149	0.9722851	12299
12300	riotously	0.0315492	0.9684508	12300
12280	smashing	0.0336163	0.9663837	12280
12298	bicycle	0.0347548	0.9652452	12298
12302	pontypool	0.0366149	0.9633851	12302
11621	smashed	0.0468791	0.9531209	11621
12303	bedminster	0.0499935	0.9500065	12303
12279	p.c	0.0528924	0.9471076	12279
12289	roughs	0.0548967	0.9451033	12289
12257	rioters	0.0579995	0.9420005	12257
12044	stoned	0.0585510	0.9414490	12044
12301	abersychan	0.0585510	0.9414490	12301
12304	lofts	0.0585510	0.9414490	12304
12295	missile	0.0611686	0.9388314	12295
11754	supt	0.0744892	0.9255108	11754
11959	foley	0.0744892	0.9255108	11959
11977	missiles	0.0776601	0.9223399	11977
9149	staves	0.0819225	0.9180775	9149
12094	panes	0.0835908	0.9164092	12094
12268	terrell	0.0835908	0.9164092	12268

3 Refinement Using Description

```
description_corpus<-quanteda::corpus(classdocs[,c("fakeid", "description", "EV_article")], t
description_dfm <- preprocess_corpus(description_corpus, stem=FALSE, min_termfreq=5, min_doc

both_dfms<-split_dfm(description_dfm, n_train = 1000)

nb<-quanteda::textmodel_nb(both_dfms$train, y=quanteda::docvars(both_dfms$train, "EV_article
prob_nb<-predict(nb, newdata = both_dfms$test, type="probability")
pred_nb<-data.frame(predict(nb, newdata = both_dfms$test, type="class"))
res<-data.frame(predict_nb=pred_nb, prob_nb)
names(res)<-c("predict_nb", "prob_notev", "prob_ev")
assess_classification(organise_results(both_dfms$test, res)) %>%
  filter(rowname %in% c("Precision", "Recall", "F1")) %>%
  kable()
```

rowname	value	model
Precision	0.7481663	naive bayes
Recall	0.8644068	naive bayes
F1	0.8020970	naive bayes

Note: this way of creating the dfm does make dfms terms equal because one overall dfm is created and then it is subset. You can see below that the number of features in both dfms is the same:

```
print(both_dfms$train)

## Document-feature matrix of: 1,000 documents, 1,647 features (99.2% sparse).

print(both_dfms$test)

## Document-feature matrix of: 894 documents, 1,647 features (99.2% sparse).
```

4 Refinement Using OCR

```
ocr_corpus<-quanteda::corpus(classdocs[,c("fakeid", "ocr", "EV_article")], text_field = "ocr
ocr_dfm <- preprocess_corpus(ocr_corpus, stem=FALSE, min_termfreq=5, min_docfreq = 2)

both_dfms<-split_dfm(ocr_dfm, n_train = 1000)
nb<-quanteda::textmodel_nb(both_dfms$train, y=quanteda::docvars(both_dfms$train, "EV_article
prob_nb<-predict(nb, newdata = both_dfms$test, type="probability")
pred_nb<-data.frame(predict(nb, newdata = both_dfms$test, type="class"))
res<-data.frame(predict_nb=pred_nb, prob_nb)
names(res)<-c("predict_nb", "prob_notev", "prob_ev")
assess_classification(organise_results(both_dfms$test, res)) %>%
  filter(rowname %in% c("Precision", "Recall", "F1")) %>%
  kable()
```

rowname	value	model
Precision	0.6867220	naive bayes
Recall	0.8642298	naive bayes
F1	0.7653179	naive bayes

5 King Algorithm: implementation

5.1 Incrementally Define R and S

R is out reference set. [King suggestions: could define R based on one simple keyword search].

5.1.1 Intermediate step

Take keywords in R, K_R, ranked by simple statistic such as document frequency or frequency-inverse document frequency. User examines elements of K_R apart from those used to define the set and chooses some keywords to define Q_S, which in turn generates a definition for S so that we can run the rest of the algorithm. The user can continue to add keywords from K_R into the final desired query $Q_R T$. This step also mitigates teh issue of how to define a search set in large data sets that do not fit into memory all at onece or may not even be able to be retrieved all at onece. is the BNA.

5.2 Partition S into T and $S \backslash T$

5.3 Discovering Keywords to Classify Documents

5.4 Human Input and Human-Computer Iteration