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()
# just some candidate document for speed/space reasons
unclassdocs <- durhamevp::get_candidate_documents(3000:6000)
```

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

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

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	0.7842105	naive bayes
Recall	0.8370787	naive bayes
F1	0.8097826	naive bayes

```
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)
```

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).
```

3.1 More realistic use case

Exclude non-election cases which would not be returned by our keywords:

```
description_corpus<-quanteda::corpus(classdocs[classdocs$election_article==1,c("fakeid", "de
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	0.9019608	naive bayes
Recall	0.7076923	naive bayes
F1	0.7931034	naive bayes

```
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)
```

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	0.6456212	naive bayes
Recall	0.9031339	naive bayes
F1	0.7529691	naive bayes

5 King Algorithm: implementation

5.1 Incrementally Define R and S

R is our 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 the issue of how to define a search set in large data sets that do not fit into memory all at once or may not even be able to be retrieved all at onece. is the BNA.

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

To partition S into T and $S\backslash T$, we first define a 'training' set by sampling from S and R. Since R is typically much smaller than S our test set for our classifiers is all all of S, we often use the entire R set and a sample of S as our training set.

```
R <- classdocs[classdocs$EV_article==1,]
R$R <- 1
R$in_sample<-1
S <- unclassdocs
S$R <- 0

n_sample_S <- 1000
S$in_sample<-0
S$in_sample[sample(1:nrow(S), n_sample_S)]<-1

R_S <- dplyr::bind_rows(R, S)
R_S$fakeid<-1:nrow(R_S)



R_S_corpus<-quanteda::corpus(R_S[,c("fakeid", "ocr", "in_sample", "R")], text_field = "ocr")
R_S_dfm <- preprocess_corpus(R_S_corpus, stem=FALSE, min_termfreq=20, min_docfreq = 20)

king_train_dfm<-quanteda::dfm_subset(R_S_dfm, quanteda::docvars(R_S_dfm, "in_sample")==1)
king_nb <- quanteda::textmodel_nb(king_train_dfm, y=quanteda::docvars(king_train_dfm, "R"),
keywords<-nb_keywords(king_train_dfm, "R")
knitr::kable(head(keywords, 20))
```

	rowname	0	1	id
3	generated	0.0221304	0.9778696	3
14	legible	0.0227857	0.9772143	14
8	technology	0.0250073	0.9749927	8
13	deciphering	0.0250073	0.9749927	13
2919	chartist	0.0267458	0.9732542	2919
2670	detective	0.0337936	0.9662064	2670
2271	roughs	0.0350241	0.9649759	2271
9415	telegraphed	0.0353459	0.9646541	9415
12921	gutted	0.0370477	0.9629523	12921
2501	p.c	0.0449904	0.9550096	2501
1079	cobden	0.0458847	0.9541153	1079
3421	smashing	0.0495804	0.9504196	3421
10653	worker	0.0558817	0.9441183	10653
9331	schoolroom	0.0579874	0.9420126	9331
13218	gladstone's	0.0579874	0.9420126	13218
3510	ratepayers	0.0602579	0.9397421	3510
8408	telegram	0.0632289	0.9367711	8408
1167	disraeli	0.0653778	0.9346222	1167
5	optical	0.0714484	0.9285516	5
12923	pontypool	0.0787619	0.9212381	12923

After fitting the classifiers, we use the estimated parameters to generate predicted probabilities of R membership for all documents in S. Of cource, all the search set documents in fact fall within S but our interest in in learning from the *mistakes* these classifiers make.

```
S_dfm <- quanteda::dfm_subset(R_S_dfm, quanteda::docvars(R_S_dfm, "R")==0)
quanteda::docvars(S_dfm, "T")<-predict(king_nb, newdata = S_dfm, type="class")
```

5.3 Discovering Keywords to Classify Documents

After partitioning S into estimated target T and non-target set $S\backslash T$, we must find and rank keywords which best discriminate T and $S\backslash T$. King does this in three stages:

5.3.1 identify keywords in S

5.3.2 sort them into those that predict each of the two steps

My suggestion here is simply to use binary_dfm

```
S_dfm_binary<-quanteda::dfm_weight(S_dfm, scheme="boolean")
king_stage2 <- quanteda::textmodel_nb(S_dfm, y=quanteda::docvars(S_dfm_binary, "T"), distrib
king_stage2 <- nb_keywords(S_dfm, "T")
knitr::kable(head(king_stage2, 30))
```

	rowname	0	1	id
8722	unionism	0.0113223	0.9886777	8722
8429	horden	0.0141129	0.9858871	8429
1208	protectionist	0.0173133	0.9826867	1208
9203	suffragists	0.0203970	0.9796030	9203
9590	nationalist	0.0223910	0.9776090	9590
8485	motor	0.0237863	0.9762137	8485
10142	socialism	0.0290080	0.9709920	10142
8420	nationalists	0.0296335	0.9703665	8420
9858	socialist	0.0296335	0.9703665	9858
2335	dump	0.0302866	0.9697134	2335
9120	bicycle	0.0316832	0.9683168	9120
9011	truncheons	0.0399786	0.9600214	9011
3074	asquith	0.0427787	0.9572213	3074
8737	industries	0.0438013	0.9561987	8737
9286	rowdyism	0.0438013	0.9561987	9286
9331	schoolroom	0.0438013	0.9561987	9331
9415	telegraphed	0.0438013	0.9561987	9415
8994	batons	0.0455433	0.9544567	8994
8421	socialists	0.0460006	0.9539994	8421
2417	workers	0.0471852	0.9528148	2417
8747	candidature	0.0491752	0.9508248	8747
2631	unionist	0.0495718	0.9504282	2631
9138	railings	0.0502017	0.9497983	9138
9205	barricaded	0.0502017	0.9497983	9205
1993	witness's	0.0541586	0.9458414	1993
2919	chartist	0.0541586	0.9458414	2919
8609	drafted	0.0541586	0.9458414	8609
9043	booing	0.0541586	0.9458414	9043
8398	hartlepool	0.0575614	0.9424386	8398
8430	sacked	0.0598163	0.9401837	8430

```
knitr::kable(tail(king_stage2, 30))
```

	rowname	0	1	id
16564	despised	0.9361369	0.0638631	16564
97	fruits	0.9370575	0.0629425	97
16555	breaching	0.9370575	0.0629425	16555
5001	candidly	0.9379520	0.0620480	5001
16375	fortress	0.9379520	0.0620480	16375
4632	citadel	0.9381717	0.0618283	4632
16237	scrip	0.9388215	0.0611785	16237
7969	lunette	0.9392471	0.0607529	7969
4023	aspire	0.9420685	0.0579315	4023
7970	laurent	0.9428271	0.0571729	7970
13935	shells	0.9449882	0.0550118	13935
15303	carolina	0.9449882	0.0550118	15303
6762	combat	0.9463404	0.0536596	6762
7376	batteries	0.9504040	0.0495960	7376
4633	belgians	0.9537326	0.0462674	4633
14316	belgian	0.9556141	0.0443859	14316
7397	repose	0.9560607	0.0439393	7397
4619	antwerp	0.9562807	0.0437193	4619
11134	senate	0.9564985	0.0435015	11134
12926	prussia	0.9564985	0.0435015	12926
2897	gerard	0.9566425	0.0433575	2897
15893	frontiers	0.9577611	0.0422389	15893
4618	capitulation	0.9581658	0.0418342	4618
16545	lillo	0.9611442	0.0388558	16545
6394	belgium	0.9623721	0.0376279	6394
4623	chasse	0.9626404	0.0373596	4623
7977	bombs	0.9672625	0.0327375	7977
7981	scheldt	0.9679827	0.0320173	7981
7973	besiegers	0.9727743	0.0272257	7973
4625	forts	0.9769438	0.0230562	4625

5.4 rank them by degree of discriminatory power

We already seem to have done this in the step above

5.5 Human Input and Human-Computer Iteration

Now present