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 classified data there are 1925 cases:

- **703** non-election articles.
- **796** election violence articles.
- **426** 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_docf

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

| | rowname | 0 | 1 | id |
|---|---|---|---|---|
| 12432 | roughly | 0.0274481 | 0.9725519 | 12432 |
| 12433 | riotously | 0.0310748 | 0.9689252 | 12433 |
| 12409 | smashing | 0.0340764 | 0.9659236 | 12409 |
| 12431 | bicycle | 0.0358056 | 0.9641944 | 12431 |
| 12435 | pontypool | 0.0358056 | 0.9641944 | 12435 |
| 11726 | smashed | 0.0476850 | 0.9523150 | 11726 |
| 12436 | bedminster | 0.0479798 | 0.9520202 | 12436 |
| 12408 | p.c | 0.0544608 | 0.9455392 | 12408 |
| 12418 | roughs | 0.0555324 | 0.9444676 | 12418 |
| 12169 | stoned | 0.0602765 | 0.9397235 | 12169 |
| 12429 | missile | 0.0602765 | 0.9397235 | 12429 |
| 12434 | abersychan | 0.0602765 | 0.9397235 | 12434 |
| 12437 | lofts | 0.0602765 | 0.9397235 | 12437 |
| 12386 | rioters | 0.0641103 | 0.9358897 | 12386 |
| 9213 | staves | 0.0752830 | 0.9247170 | 9213 |
| 11863 | supt | 0.0766460 | 0.9233540 | 11863 |
| 12080 | foley | 0.0766460 | 0.9233540 | 12080 |
| 12430 | polioe | 0.0810481 | 0.9189519 | 12430 |
| 12223 | panes | 0.0834444 | 0.9165556 | 12223 |
| 11931 | detective | 0.0842750 | 0.9157250 | 11931 |

# 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 = 700)

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 | 0.7697842 | naive bayes |
| Recall | 0.8560000 | naive bayes |
| F1 | 0.8106061 | 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 = 700)
```

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: 700 documents, 1,675 features (99.2% sparse).

print(both_dfms$test)

## Document-feature matrix of: 1,225 documents, 1,675 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 = 700)

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 | 0.8750000 | naive bayes |
| Recall | 0.7754491 | naive bayes |
| F1 | 0.8222222 | 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 = 700)
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 | 0.6708683 | naive bayes |
| Recall | 0.9373777 | naive bayes |
| F1 | 0.7820408 | 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 $Q_RT$. 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 <- 700
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 | 0.0323522 | 0.9676478 | 3 |
| 14 | legible | 0.0332999 | 0.9667001 | 14 |
| 2921 | chartist | 0.0343049 | 0.9656951 | 2921 |
| 5 | optical | 0.0365084 | 0.9634916 | 5 |
| 8 | technology | 0.0365084 | 0.9634916 | 8 |
| 13 | deciphering | 0.0365084 | 0.9634916 | 13 |
| 2501 | p.c | 0.0443407 | 0.9556593 | 2501 |
| 2671 | detective | 0.0470963 | 0.9529037 | 2671 |
| 13000 | gutted | 0.0513514 | 0.9486486 | 13000 |
| 9243 | bicycle | 0.0564516 | 0.9435484 | 9243 |
| 2271 | roughs | 0.0594015 | 0.9405985 | 2271 |
| 1079 | cobden | 0.0626767 | 0.9373233 | 1079 |
| 9441 | supt | 0.0626767 | 0.9373233 | 9441 |
| 2952 | conser | 0.0663342 | 0.9336658 | 2952 |
| 3425 | smashing | 0.0704449 | 0.9295551 | 3425 |
| 3470 | inciting | 0.0804111 | 0.9195889 | 3470 |
| 9449 | schoolroom | 0.0804111 | 0.9195889 | 9449 |
| 10758 | lodgers | 0.0804111 | 0.9195889 | 10758 |
| 13297 | gladstone's | 0.0804111 | 0.9195889 | 13297 |
| 1995 | shutters | 0.0865322 | 0.9134678 | 1995 |

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 |
|---|---|---|---|---|
| 8565 | horden | 0.0152274 | 0.9847726 | 8565 |
| 1208 | protectionist | 0.0186758 | 0.9813242 | 1208 |
| 2335 | dump | 0.0210604 | 0.9789396 | 2335 |
| 9323 | suffragists | 0.0219967 | 0.9780033 | 9323 |
| 8621 | motor | 0.0256448 | 0.9743552 | 8621 |
| 9243 | bicycle | 0.0341371 | 0.9658629 | 9243 |
| 9137 | truncheons | 0.0430465 | 0.9569535 | 9137 |
| 9404 | rowdyism | 0.0471482 | 0.9528518 | 9404 |
| 9449 | schoolroom | 0.0471482 | 0.9528518 | 9449 |
| 9530 | telegraphed | 0.0471482 | 0.9528518 | 9530 |
| 9120 | batons | 0.0490164 | 0.9509836 | 9120 |
| 2417 | workers | 0.0507769 | 0.9492231 | 2417 |
| 9702 | nationalist | 0.0521138 | 0.9478862 | 9702 |
| 9261 | railings | 0.0540100 | 0.9459900 | 9261 |
| 9325 | barricaded | 0.0540100 | 0.9459900 | 9325 |
| 8852 | unionism | 0.0550107 | 0.9449893 | 8852 |
| 2921 | chartist | 0.0582486 | 0.9417514 | 2921 |
| 9168 | booing | 0.0582486 | 0.9417514 | 9168 |
| 8742 | drafted | 0.0582486 | 0.9417514 | 8742 |
| 8556 | nationalists | 0.0602760 | 0.9397240 | 8556 |
| 9966 | socialist | 0.0602760 | 0.9397240 | 9966 |
| 8867 | industries | 0.0611270 | 0.9388730 | 8867 |
| 8534 | hartlepool | 0.0618915 | 0.9381085 | 8534 |
| 8876 | candidature | 0.0618915 | 0.9381085 | 8876 |
| 8566 | sacked | 0.0643045 | 0.9356955 | 8566 |
| 3425 | smashing | 0.0660205 | 0.9339795 | 3425 |
| 2632 | unionist | 0.0673033 | 0.9326967 | 2632 |
| 10244 | socialism | 0.0690934 | 0.9309066 | 10244 |
| 3076 | asquith | 0.0742795 | 0.9257205 | 3076 |
| 1993 | witness's | 0.0761856 | 0.9238144 | 1993 |

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

| | rowname | 0 | 1 | id |
|---|---|---|---|---|
| 10442 | mankind | 0.9348514 | 0.0651486 | 10442 |
| 12687 | essence | 0.9348514 | 0.0651486 | 12687 |
| 16592 | regency | 0.9348514 | 0.0651486 | 16592 |
| 10466 | relaxed | 0.9358848 | 0.0641152 | 10466 |
| 13348 | i1 | 0.9358848 | 0.0641152 | 13348 |
| 16565 | berchem | 0.9358848 | 0.0641152 | 16565 |
| 13432 | spoliation | 0.9368860 | 0.0631140 | 13432 |
| 4914 | forts | 0.9378563 | 0.0621437 | 4914 |
| 11212 | stupid | 0.9378563 | 0.0621437 | 11212 |
| 16087 | talleyrand | 0.9378563 | 0.0621437 | 16087 |
| 16374 | honored | 0.9387973 | 0.0612027 | 16374 |
| 7535 | batteries | 0.9392572 | 0.0607428 | 7535 |
| 14632 | wvill | 0.9397102 | 0.0602898 | 14632 |
| 14892 | unfettered | 0.9397102 | 0.0602898 | 14892 |
| 13005 | prussia | 0.9403038 | 0.0596962 | 13005 |
| 16672 | despised | 0.9405963 | 0.0594037 | 16672 |
| 8124 | scheldt | 0.9418776 | 0.0581224 | 8124 |
| 6584 | belgium | 0.9427826 | 0.0572174 | 6584 |
| 8113 | lunette | 0.9435024 | 0.0564976 | 8113 |
| 6666 | ambition | 0.9436337 | 0.0563663 | 6666 |
| 14407 | belgian | 0.9455937 | 0.0544063 | 14407 |
| 8114 | laurent | 0.9468458 | 0.0531542 | 8114 |
| 14963 | engraved | 0.9482080 | 0.0517920 | 14963 |
| 15385 | carolina | 0.9488633 | 0.0511367 | 15385 |
| 6939 | combat | 0.9501253 | 0.0498747 | 6939 |
| 4130 | innovation | 0.9579148 | 0.0420852 | 4130 |
| 11224 | senate | 0.9595975 | 0.0404025 | 11224 |
| 15979 | frontiers | 0.9607738 | 0.0392262 | 15979 |
| 8121 | bombs | 0.9696190 | 0.0303810 | 8121 |
| 8117 | besiegers | 0.9747445 | 0.0252555 | 8117 |

## 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

The King et al suggestion is to present the above lists to humans who then choose additional keywords from these lists. That might be fine if the final goal is develop keywords. However, this does not work so well if the final goal is to obtain sets of documents based on those keywords. Some problems with this approach for obtaining documents

1. the set of documents will keep on growing without limit (with many irrelevant documents)

2. there is no guidance about which keywords to select from the lists presented

An alternative proposal is to continue with a classification approach. That is to treat the classifer as informative about two separate matters: keywords retrival as potentially informative

The stage II classifier is informative about the following issues:

1. terms which are in $S$ but not in $R$ which may be positively predictive of election violence.

2. terms which are very differently predictive of election violence between stage I and stage II classification.

3.

Idea 1: For keyword $t$ in keywords $1 \dots n$, $Q_t$ is the Boolean query which returns document $d$ from $S$ if $d$ contains $t$. $S\prime$ is the set of documents returned by the set of queries $Q_{T_1} \dots Q_{T_n}$. The probability that document $d$ is an example of the concept of interest is given by the set of queries which return that document. We run independent queries with each of the keywords. We have different sets returned from

Idea 2: King's stage II classification can be thought of of the first part of an EM process which augments information from labelled data with information from unlabelled data. See for a classic example Nigam et. al. 2000 - Semi-supervised parameter estimation: can learn from a combination of labeled and unlabeled data in a loop. First train a classifier using available labeled documents, and probabilistically label the unlabeled documents. Then train a new classifier using the labels for all the documents, iterate to convergence. This basic EM procedure works well when the data conform to the generative assumptions of the model. However, these assumptions are often violated in practice, and poor performance can result. Two extension improve classification accuracy (1) a weighting factor to modulate the contribution of the unlabeled data and (2) the use of multiple mixture components per class.

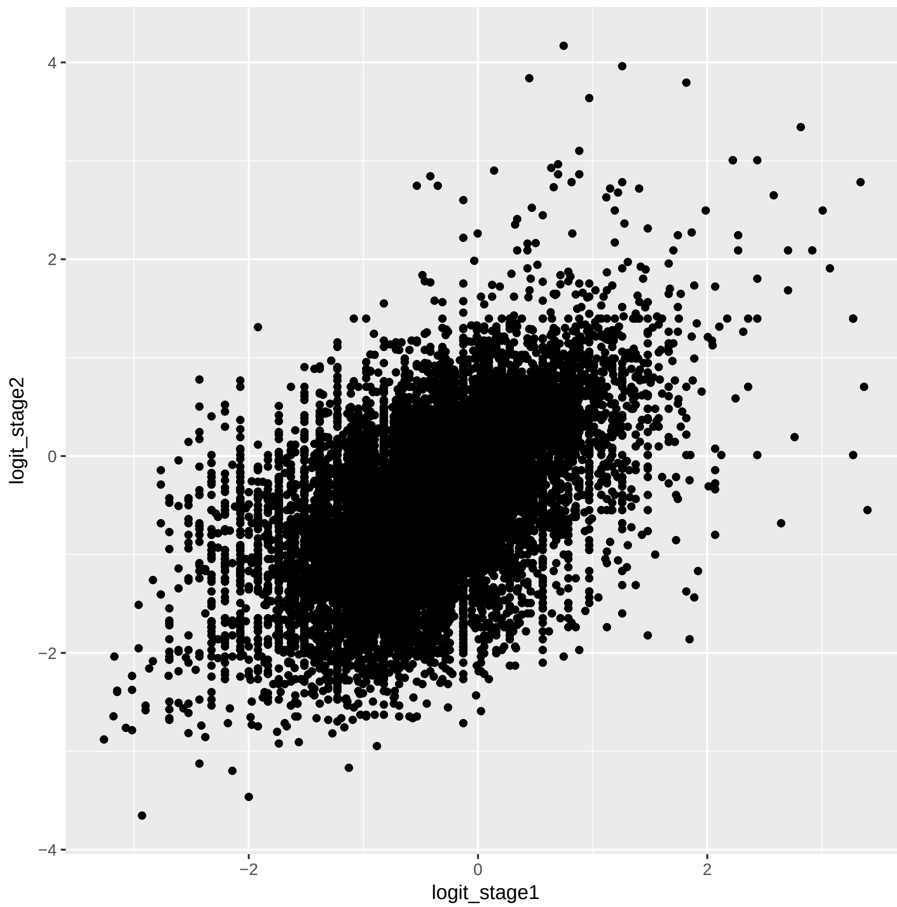
How can unlabeled data increase classification accuracy? Unlabelled data provide information about the joint probability distribution over words. Suppose that using only the labeled data we determine that documents containing the word 'riot' belongs to the election violence class. If we use this fact to estimate the classification of the many unlabeled documents, we might find the word 'bludgeon' occurs frequently with the word 'riot' and we might construct a more accurate classifier that considers both 'riot' and 'bludgeon' as indicators of positive examples.

Given a training set $(x^{(i)}, y^{(i)})$ for $i = 1 \dots n$, where $\theta$ is a paramter vector consiting of the values for all parameters $q(y)$ and $q_i(x|y)$ the log-likelihood function is:

$$
\begin{aligned}
L(\theta) &= \sum_{i=1}^{n} \log p\left(x^{(i)}, y^{(i)}\right) \\
&= \sum_{i=1}^{n} \log q(y^{(i)}) + \sum_{i=1}^{n}\sum_{j=1}^{d} \log q_j\left(x_j^{(i)}|y^{(i)}\right)
\end{aligned} \tag{1}
$$

Idea 3: to find a set of keywords searches which returns $T$ - when you have a set of keywords which returns $T$ - adding new keywords will change $S$ but not reveal more cases of $T$ - so will not see large changes in individual word predictivity of class in Naive Bayes classifier. encompasses the add search terms until revised data set $S\prime$ is not substantially changed by addition of new When new search terms are added this changes the nature of $S$ - at each iteration new information added - consider changes to $S\prime$

```
names(king_stage2)<-c("rowname", "stage2_0", "stage2_1", "stage2_id")
names(keywords)<-c("rowname", "stage1_0", "stage1_1", "stage1_id")
left_join(king_stage2, keywords, by="rowname") %>%
  mutate(logit_stage1 =log(stage1_1/stage1_0), logit_stage2=log(stage2_1/stage2_0)) %>%
  ggplot(aes(logit_stage1, logit_stage2))+
  geom_point()
```



4
2
0
−2
−4
logit_stage2
−2
0
2
logit_stage1