Computational Political Science

Session 5

David Broska
Zeppelin University
March 2, 2021

Outline for today

1. **Evaluating classifier performance**
    - Precision, Recall, F1
    - Accuracy
2. **Coding exercise**
    - Classifying movie reviews
    - Which letters are most predictive of female and male names?
3. **Wordscores model**
    - How it relates to Bayes theorem
    - How it is implemented
4. **Wordscores coding example**

Evaluating classifier performance

Supervised machine learning

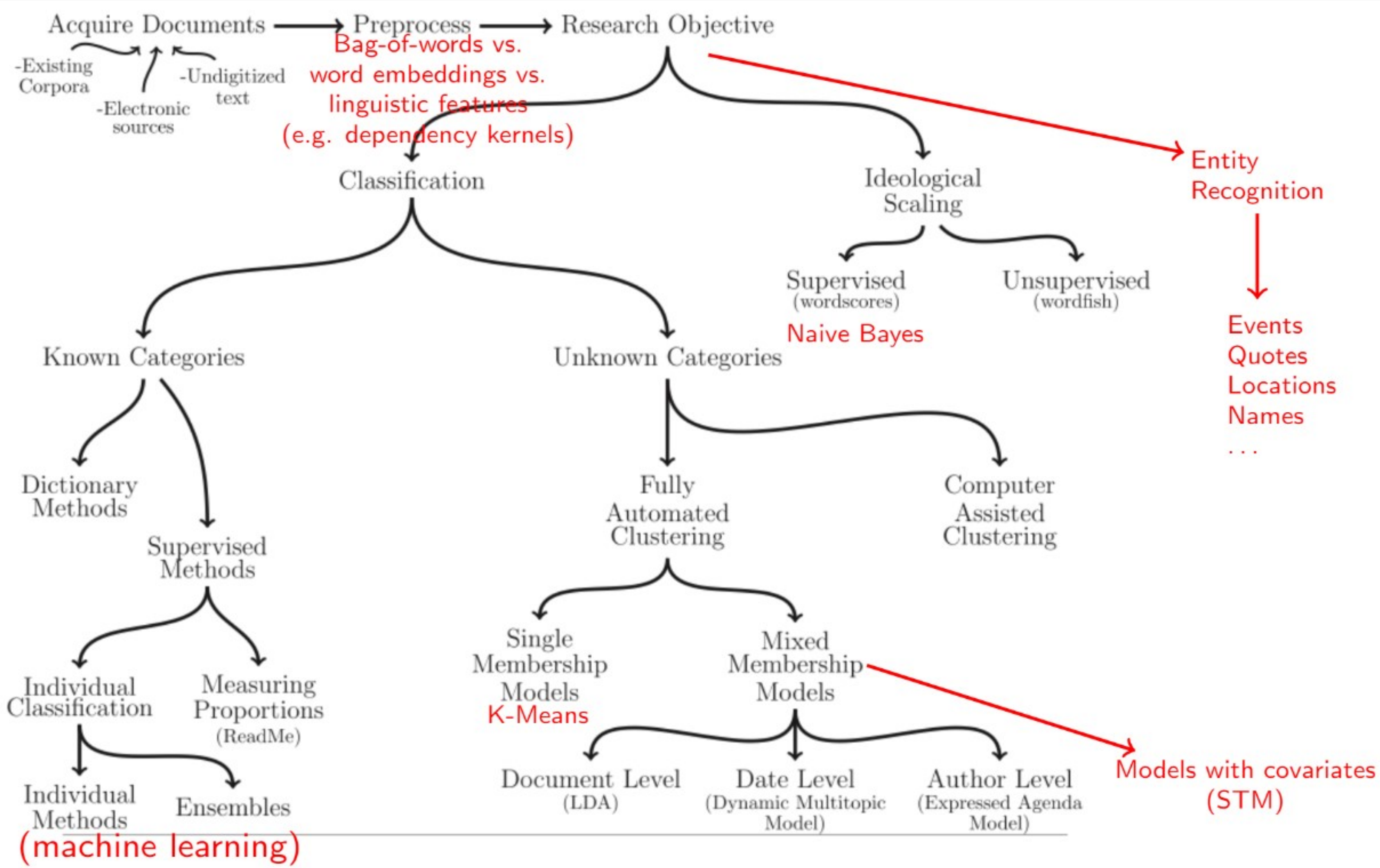


Fig. 1 in Grimmer and Stuart (2013)

Supervised machine learning

The goal is classify documents into pre existing categories.

For example, authors of documents, sentiment of tweets, ideological position of parties based on manifestos, tone of movie reviews...

What we need

- Hand-coded dataset (labeled), to be split into:
  - *Training set*: used to train the classifier
  - *Validation/Test set*: used to validate the classifier
- Method to *extrapolate* from hand coding to unlabeled documents (classifier):
  - Naive Bayes, regularized regression, SVM, CNN, ensemble methods, etc.
- *Performance metric* to choose best classifier and avoid overfitting:
  - Confusion matrix, accuracy, precision, recall...
- Approach to *validate* classifier: cross-validation

Principles of supervised learning

Generalization

A classifier or a regression algorithm learns to correctly predict output from given inputs

Crucially, it predicts correctly not only in previously seen samples but also in previously unseen samples.

Overfitting

A classifier or a regression algorithm learns to correctly predict output from given inputs in previously seen samples.

However, it fails to do so in previously unseen samples. This causes poor prediction/generalization.

The goal is to maximize the frontier of precise identification of true condition with accurate recall

Confusion matrix

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Precision: Does the classifier identify only my content?
% of documents that are predicted positive that are indeed positive

Recall: Does the classifier identify all my content?
% of positive documents that are predicted positive

Accuracy: How correctly is the classifier's identifications?
% of documents that are correctly predicted

Measuring performance example

Assume

- We have a corpus where 80 documents are really positive (as opposed to negative, as in sentiment)
- Our method declares that 60 are positive
- Of the 60 declared positive, 45 are actually positive

Exercise

1. Please draw a confusion matrix with the given numbers and compute precision and recall

2. Compute accuracy for the following two scenarios

   a) 10 true negatives
   b) 100 true negatives

How do you interpret the result? How does accuracy relate to precision and recall?

Measuring performance example

Let's fill in the blanks with the given numbers on the classified documents

60 are predicted positve whereas 45 are truely positive leaving 15 false positives

		True condition		
		Positive	Negative	
Prediction	Positive	45		60
	Negative			
		80		

Since we have 80 truely positive documents there are 35 false negatives

		True condition		
		Positive	Negative	
Prediction	Positive	45	*15*	60
	Negative	*35*		
		80		

Precision

$$\frac{\text{TP}}{\text{TP} + FP} = \frac{45}{45 + 15} = \frac{3}{4} = 75\%$$

Recall

$$\frac{\text{TP}}{\text{TP} + FN} = \frac{45}{45 + 35} = \frac{9}{16} = 56.3\%$$

Measuring performance example

Accuracy with **10** true negatives

		True condition		
		Positive	Negative	
Prediction	Positive	45	*15*	60
	Negative	*35*	***10***	***45***
		80	***25***	***105***

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{45 + 10}{105} = 52\%$$

Accuracy with **100** true negatives

		True condition		
		Positive	Negative	
Prediction	Positive	45	*15*	60
	Negative	*35*	***100***	***135***
		80	***115***	***195***

$$\text{Accuracy} = \frac{\text{sum of diagonal}}{\text{sum of all cells}} = \frac{45 + 100}{195} = 74\%$$

While precision and recall remain constant accuracy increases as the true negatives increase

A model can achieve high classification accuracy but it might be useless in solving the problem!

Measuring performance

Combined metric for precision and recall

$$
\begin{aligned}
F1 &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
&= \frac{0.75 \times 0.52}{0.75 + 0.52} \\
&= 0.64
\end{aligned}
$$

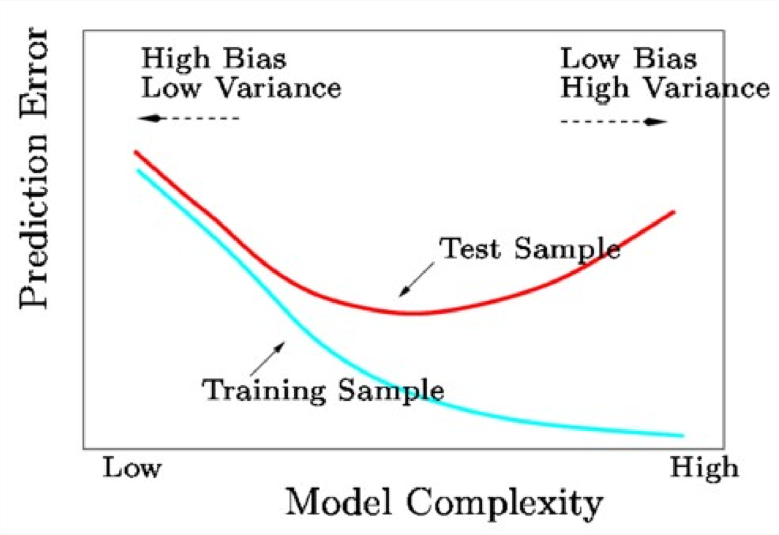
F1 is a good measure when the goal is to seek a balance between Precision and Recall

Additional remarks

- Precision and recall can be reported separately for each category
- Precision and recall (or F1) should be reported alongside accuracy. Why?
- There is generally a trade-off between precision and recall. Why?

Trade-off

- Generally we want to apply method to new data, e.g. predict class of unseen documents
- However, the classifier is trained to maximize in-sample performance
- Danger: **overfitting**



- Model is too complex, describes noise rather than signal (Bias-Variance trade-off)
- Focus on features that perform well in labeled data but may not generalize (e.g. “inflation” in 1980s)
- In-sample performance better than out-of-sample performance

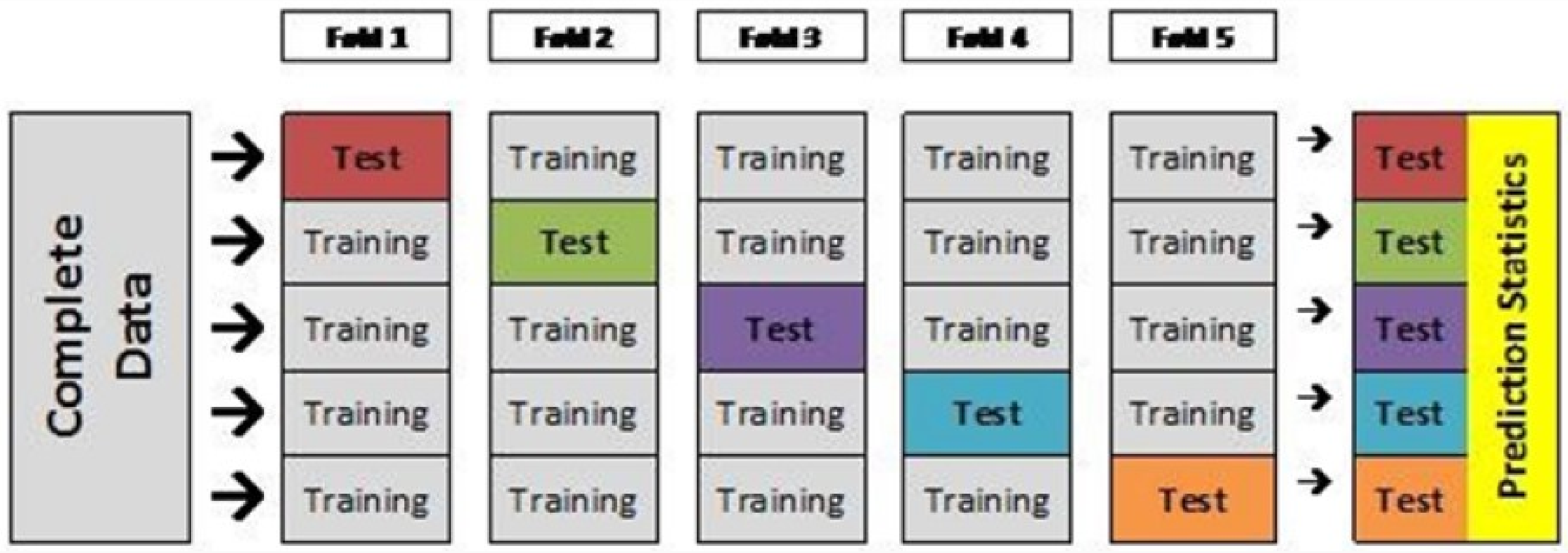
Solutions

- Randomly split dataset into training and test set
- Cross validation

Cross validation

Intuition

- Create K training and test sets ("folds") within training set
- For k in K, run the classifier and estimate performance in test set within fold
- Choose best classifier based on cross-validated performance



Coding exercise

Wordscores

From classification to scaling

Machine learning focuses on identifying classes (*classification*), while social science is typically interested in locating things on latent traits (*scaling*), for example:

- Policy positions on economic vs social dimension
- Inter- and intra-party differences
- Soft news vs hard news
- ...and any other continuous scale

But the two methods overlap and can be adapted - will demonstrate later using the Naive Bayes classifier

In fact, the class predictions for a collection of words from Naive Bayes can be adapted to scaling

Wordscores

Analogous to a "training set" and a "test set" in classification, the Wordscores method by Laver, Benoit, and Garry (2003) uses two sets of texts:

Reference texts

- texts about which we know something (a scalar dimensional score)

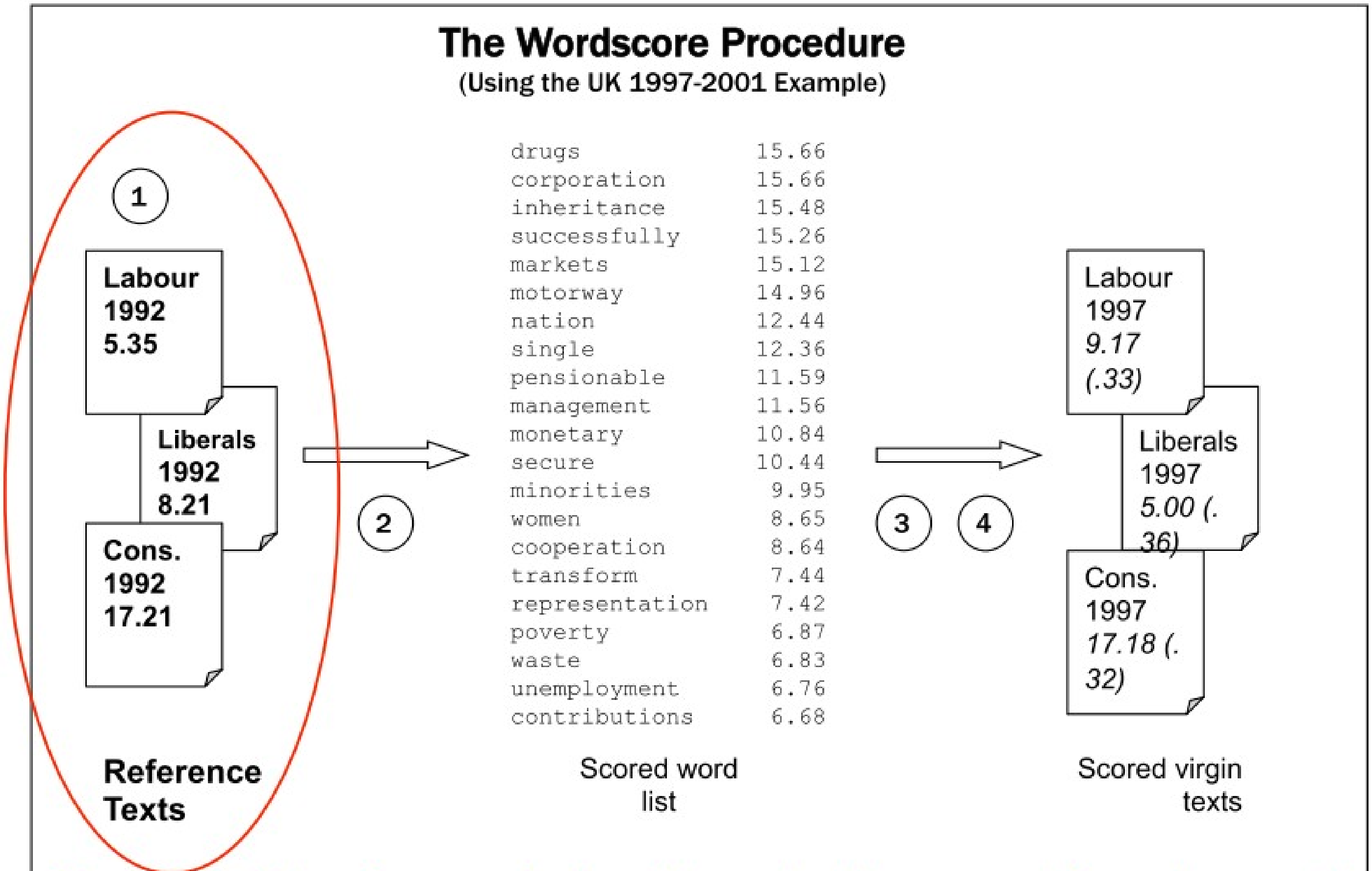
Virgin texts

- texts about which we know nothing (but whose dimensional score we'd like to know)

Basic procedure

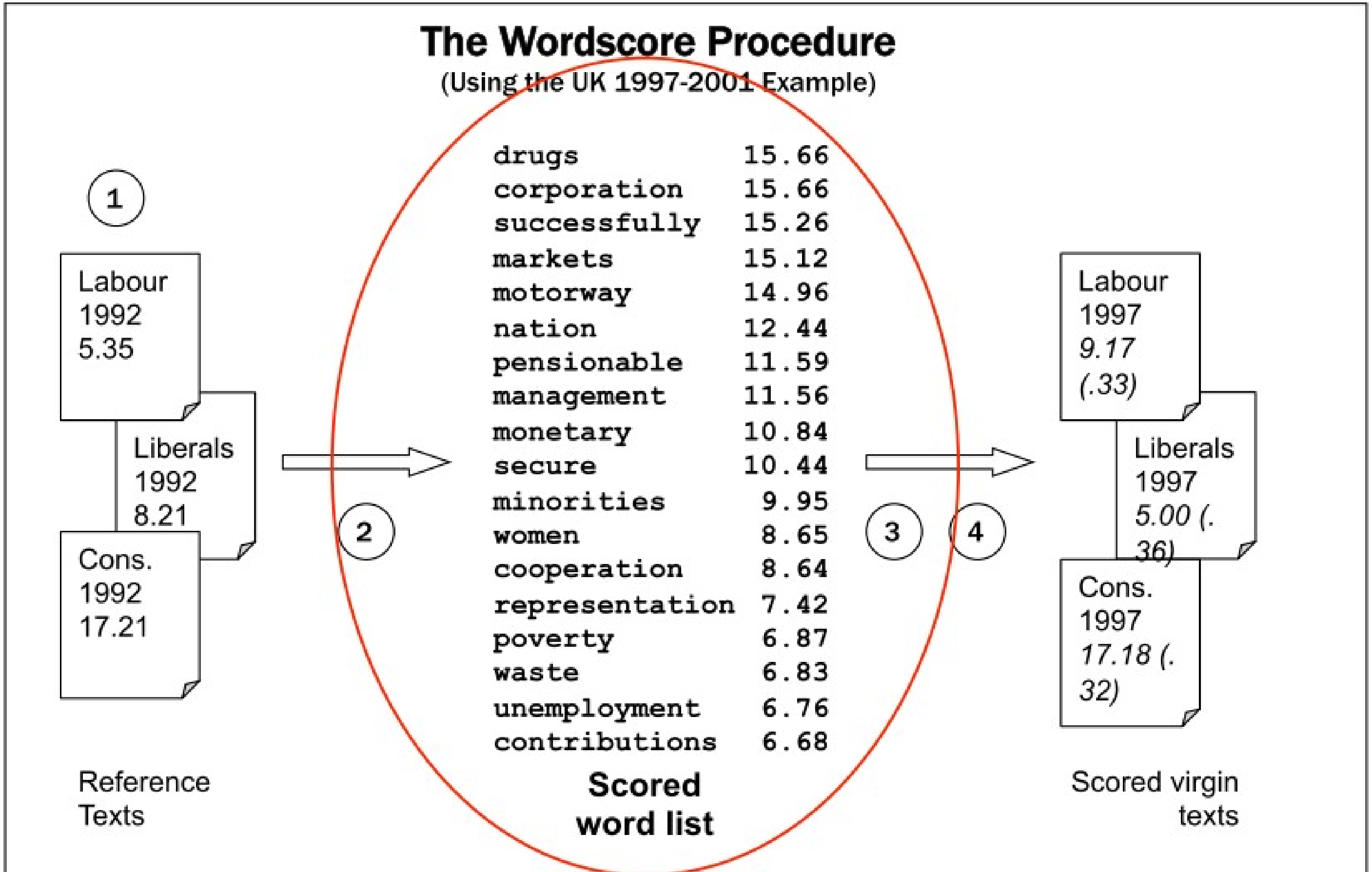
1. Analyze reference texts to obtain a single "score" for every word
2. Use word scores to score virgin texts

Wordscores procedure (I)



Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

Wordscores procedure (II)



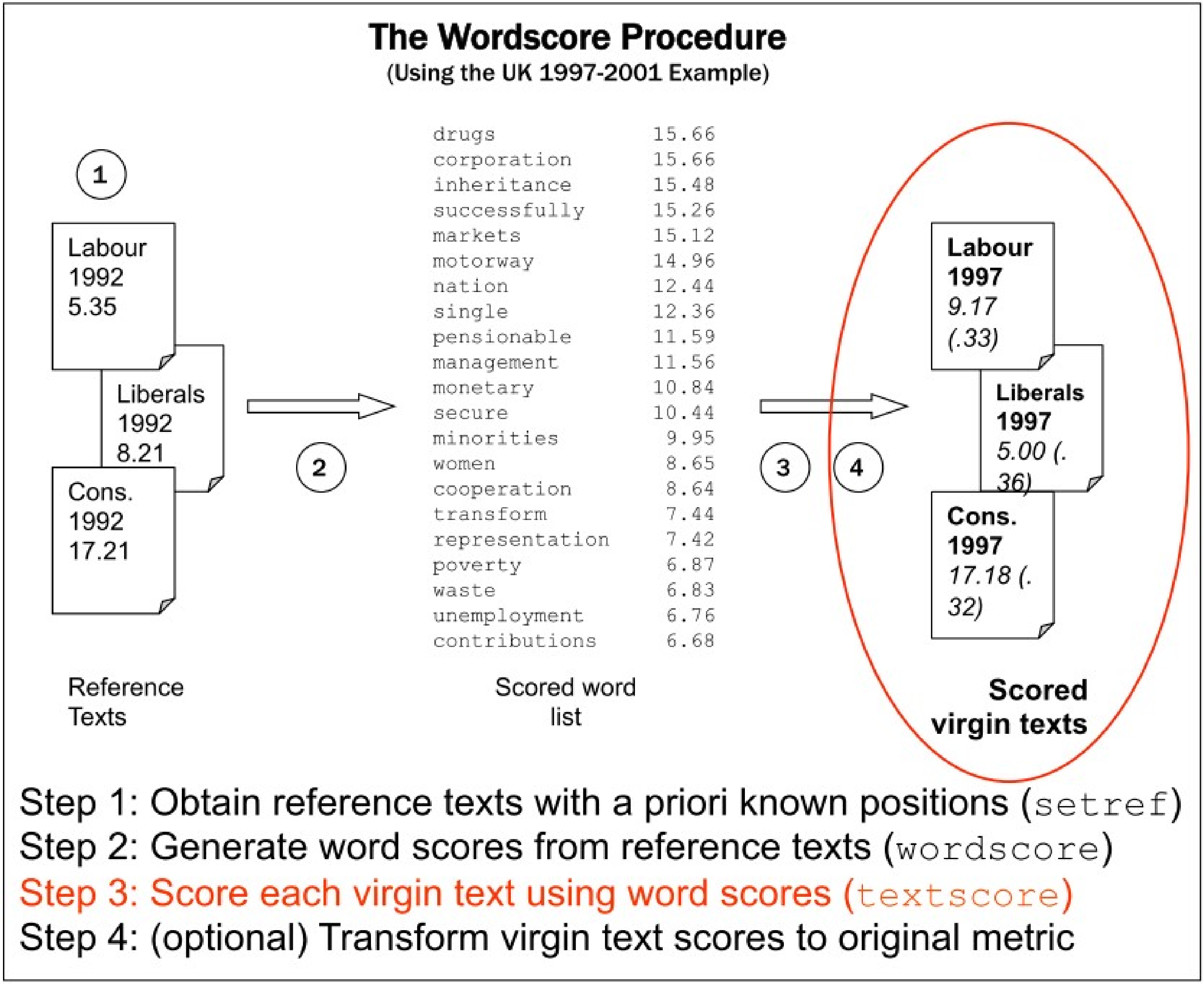
Step 1: Obtain reference texts with a priori known positions (`setref`)

Step 2: Generate word scores from reference texts (`wordscore`)

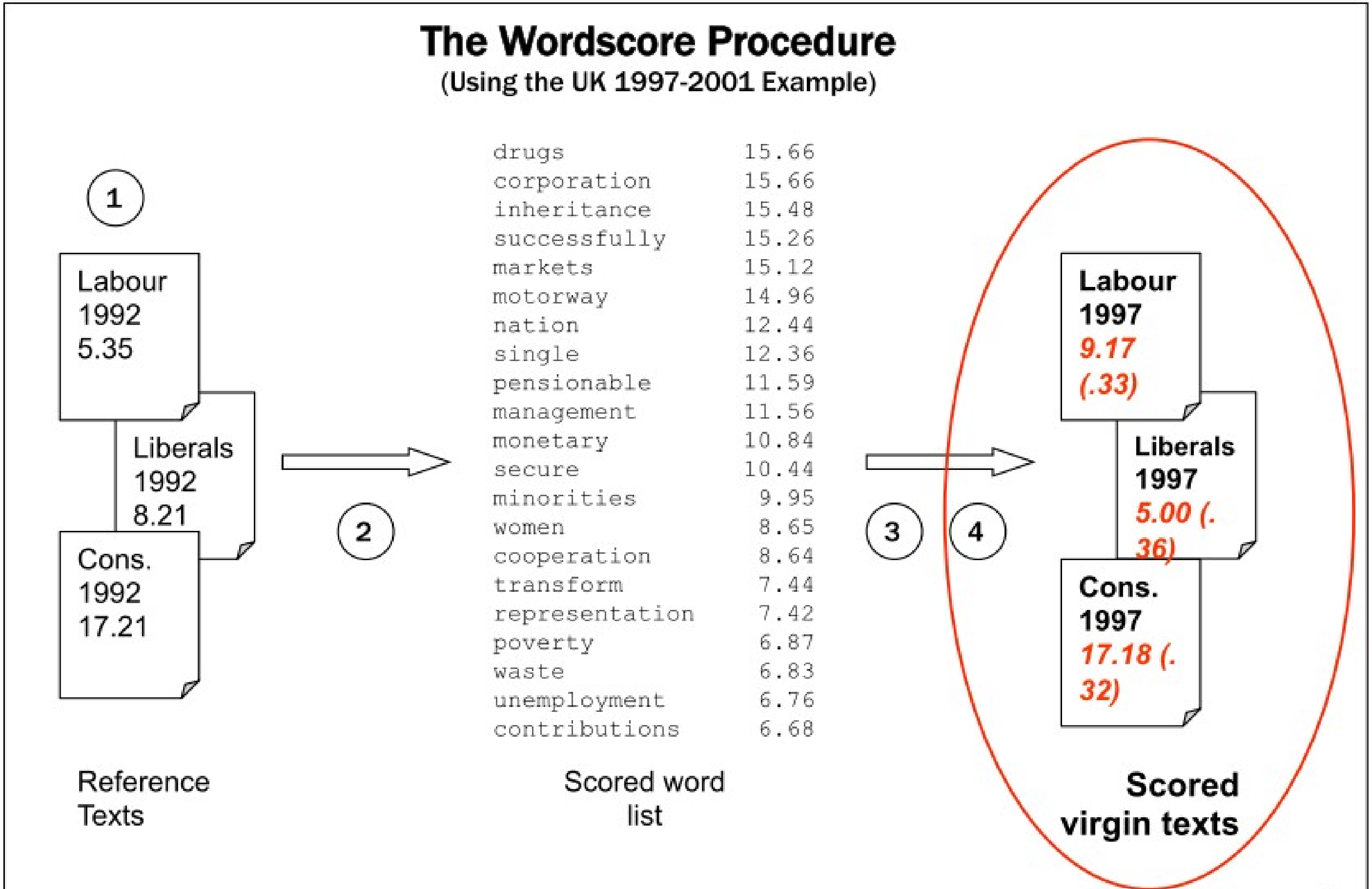
Step 3: Score each virgin text using word scores (`textscore`)

Step 4: (optional) Transform virgin text scores to original metric

Wordscores procedure (III)



Wordscores procedure (IV)



Step 1: Obtain reference texts with a priori known positions (`setref`)
Step 2: Generate word scores from reference texts (`wordscore`)
Step 3: Score each virgin text using word scores (`textscore`)
Step 4: (optional) Transform virgin text scores to original metric

Wordscore implementation

```r
# 4 texts with known and 3 texts with unknown category
txt <- c(k1 = "$ Win $",
         k2 = "$ Prize $",
         k3 = "Earn $ Easily",
         k4 = "Paypal 100 $",
         u1 = "$",
         u2 = "$ $",
         u3 = "Paypal 100 $ $")
x <- dfm(txt)
y <- c(1, 1, 1, -1, NA, NA, NA)
```

training dfm from references texts

	$	win	prize	earn	easily	paypal	100
k1	2	1	0	0	0	0	0
k2	2	0	1	0	0	0	0
k3	1	0	0	1	1	0	0
k4	1	0	0	0	0	1	1

training vector with known positions

y
1
1
1
-1

Wordscores

Compute probability of a reading document given a word

Start with a set of D reference texts, represented by an $D \times W$ document-feature matrix C_{dw} , where d indexes the document and w indexes the W total word types.

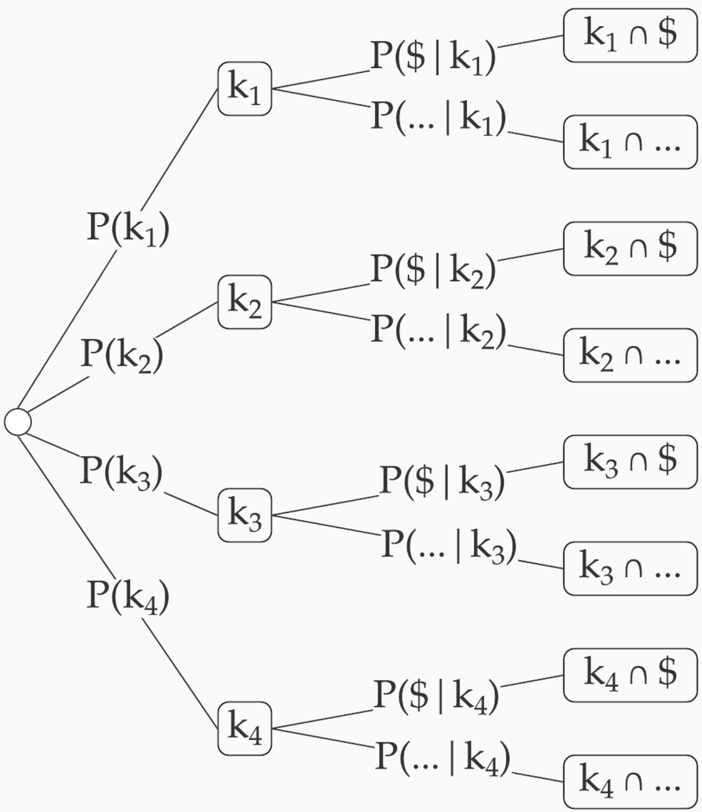
We normalize the document-feature matrix within each document by converting C_{ij} into a relative document-feature matrix (within document), by dividing C_{ij} by its word total marginals

Probability of word given the document

```
( PwGd <- dfm_weight(x[1:4,],scheme="prop") )
```

```
##      features
## docs    $  win prize earn easily paypal  100
##    k1 0.67 0.33  0.00 0.00   0.00   0.00 0.00
##    k2 0.67 0.00  0.33 0.00   0.00   0.00 0.00
##    k3 0.33 0.00  0.00 0.33   0.33   0.00 0.00
##    k4 0.33 0.00  0.00 0.00   0.00   0.33 0.33
```

P(k_1 | \$)



Uniform priors: $P(k_1)=...=P(k_4)=$ ¼

If we only read "\$" the probability of reading the document k_1 is ⅓.

Probability of word given the document:

	\$	**win**	**prize**	**earn**	**easily**	**paypal**	**100**
k1	0.67	0.33	0.00	0.00	0.00	0.00	0.00
k2	0.67	0.00	0.33	0.00	0.00	0.00	0.00
k3	0.33	0.00	0.00	0.33	0.33	0.00	0.00
k4	0.33	0.00	0.00	0.00	0.00	0.33	0.33

$$P(k_1|\$)$$

$$= \frac{P(k_1)P(\$|k_1)}{P(k_1)P(\$|k_1) + ... + P(k_2)P(\$|k_4)}$$

$$= \frac{P(\$|k_1)}{P(\$|k_1) + ... + P(\$|k_4)}$$

$$= \frac{\frac{2}{3}}{\frac{2}{3} + \frac{2}{3} + \frac{1}{3} + \frac{1}{3}} = \frac{1}{3}$$

P(document | word)

Now let's compute all probabilities of reading a document given a word

```r
PwGd # recall our matrix containing all P(word | document)
```

```
##     features
## docs    $  win prize earn easily paypal  100
##   k1 0.67 0.33  0.00 0.00   0.00   0.00 0.00
##   k2 0.67 0.00  0.33 0.00   0.00   0.00 0.00
##   k3 0.33 0.00  0.00 0.33   0.33   0.00 0.00
##   k4 0.33 0.00  0.00 0.00   0.00   0.33 0.33
```

```r
# transpose PwGd matrix
( tPwGd <- t(PwGd) )
```

```
##          docs
## features    k1   k2   k3   k4
##   $       0.67 0.67 0.33 0.33
##   win     0.33 0.00 0.00 0.00
##   prize   0.00 0.33 0.00 0.00
##   earn    0.00 0.00 0.33 0.00
##   easily  0.00 0.00 0.33 0.00
##   paypal  0.00 0.00 0.00 0.33
```

```r
# P(document | word)
( PdGw <- tPwGd / rowSums(tPwGd) )
```

```
##          docs
## features    k1   k2   k3   k4
##   $       0.33 0.33 0.17 0.17
##   win     1.00 0.00 0.00 0.00
##   prize   0.00 1.00 0.00 0.00
##   earn    0.00 0.00 1.00 0.00
##   easily  0.00 0.00 1.00 0.00
##   paypal  0.00 0.00 0.00 1.00
```

Scoring words

Compute a J-length "score" vector S for each word j as the average of each document i's scores a_i, weighted by each word's P_{ij}:

$$S_j = \sum_{i}^{I} a_i P_{ij}$$

```r
# transpose matrix so we can multiply words of the document with the document score
t(PdGw) * y[1:4]
```

```
##      features
## docs      $ win prize earn easily paypal 100
##    k1  0.33   1     0    0      0      0   0
##    k2  0.33   0     1    0      0      0   0
##    k3  0.17   0     0    1      1      0   0
##    k4 -0.17   0     0    0      0     -1  -1
```

```r
# then, sum up the result column-wise
colSums( t(PdGw) * y[1:4] )
```

```
##      $    win  prize   earn easily paypal    100
##   0.67   1.00   1.00   1.00   1.00  -1.00  -1.00
```

Scoring words

We obtain the scored words *also* by using matrix multiplication. In matrix algebra,

$$\underset{1\times J}{S} = \underset{1\times I}{a} \cdot \underset{I\times J}{P}$$

```
PdGw # P(document | word)
```

```
##          docs
## features   k1   k2   k3   k4
##   $      0.67 0.67 0.33 0.33
##   win    0.33 0.00 0.00 0.00
##   prize  0.00 0.33 0.00 0.00
##   earn   0.00 0.00 0.33 0.00
##   easily 0.00 0.00 0.33 0.00
##   paypal 0.00 0.00 0.00 0.33
##   100    0.00 0.00 0.00 0.33
```

```
y[1:4] # documents scale
```

```
## [1]  1  1  1 -1
```

```
 # matrix multiplication with P(document|words) and scores
( ws <- PdGw %*% y[1:4] )
```

```
##      $    win  prize   earn easily paypal    100
##   0.67   1.00   1.00   1.00   1.00  -1.00  -1.00
```

Scoring texts

The goal is to obtain a single score for any new text, relative to the reference texts

We do this by taking the mean of the scores of its words, weighted by their term frequency

- Note that new words outside of the set J may appear in the K virgin documents – these are simply ignored (because we have no information on their scores)
- Note also that nothing prohibits reference documents from also being scored as virgin documents

```
# matrix multiplication with P(word | document) and obtained wordscores
dfm_weight(x, scheme="prop") %*% ws
```

```
##    k1    k2    k3    k4    u1    u2    u3
##  0.78  0.78  0.89 -0.44  0.67  0.67 -0.17
```

Does this result make sense in the context of the spam example?

k1 (s)	k2 (s)	k3 (s)	k4 (¬s)	u1	u2	u3
$ Win $	$ Prize $	Earn $ Easily	Paypal 100 $	$	$ $	Paypal 100 $ $

Using textmodel_wordscores()

For convenience we can use the quanteda function to obtain the above results

```
ws_mod <- textmodel_wordscores(x,y)
```

Wordscores

```
summary(ws_mod)
```

```
## textmodel_wordscores.dfm(x = x, y = y)

## (showing first 7 elements)
##      $    win  prize   earn easily paypal    100
##   0.67   1.00   1.00   1.00   1.00  -1.00  -1.00
```

Scaled documents

```
predict(ws_mod)
```

```
##    k1    k2    k3    k4    u1    u2    u3
##  0.78  0.78  0.89 -0.44  0.67  0.67 -0.17
```

Wordscore coding exercise