Day 10: Classification and Scaling with Texts

ME314: Introduction to Data Science and Machine Learning

Jack Blumenau

25th July 2023

Day 10 Outline

Motivation

Supervised Classification for Text

Supervised Scaling of Texts

Unsupervised Scaling of Texts

Motivation

Motivation - Is this a curry?

Spiced Chickpea Stew With Coconut and Turmeric

By Alison Roman

YIELD 4 to 6 servings

TIME 55 minutes

Save to Recipe Box

Spiced chickpeas are crisped in olive oil, then simmered in a garlicky coconut milk for an insanely creamy, basically-good-for-you stew that evokes stews found in South India and parts of the Caribbean. While the chickpeas alone would be good as a side dish, they are further simmered with stock, bolstered with dark, leafy greens of your choosing and finished with a handful of fresh mint. When shopping, be sure to avoid low-fat coconut milk, coconut milk meant for drinking or cream of coconut: All are very different and would not be suitable here.

Featured in: Creamy, Hearty And (Sort Of) Virtuous.

Michael Graydon & Nikole Herriott for The New York Times. Prop Stylist: Kalen Kaminski.

Mediterranean, Soups And Stews, Chickpea, Coconut Milk, Ginger, Swiss Chard, Dinner, Lunch, Weekday, Weeknight, Main Course, Vegan, Vegetarian

Mark as **Cooked** | 16,962 ratings ★★★★★

Motivation - Is this a curry?

What is a curry?

Oxford English Dictionary: “A preparation of meat, fish, fruit, or vegetables, cooked with a quantity of bruised spices and turmeric, and used as a relish or flavouring, esp. for dishes composed of or served with rice. Hence, a curry = a dish or stew (of rice, meat, etc.) flavoured with this preparation (or with curry-powder).”

Motivation - Is this a curry?

- If a curry can be defined by the spices a dish contains, then we ought to be able to predict whether a recipe is a curry from ingredients listed in recipes
- We will evaluate the probability that #TheStew is a curry by training a curry classifier on a set of recipes
- We will use data on 9384 recipes from the BBC recipe archive
- This data includes information on
    - Recipe names
    - Recipe ingredients
    - Recipe instructions

Motivation - Is this a curry?

Our data includes information on each recipe:

```
recipes$recipe_name[1]
```

```
## [1] "Mustard and thyme crusted rib-eye of beef "
```

```
recipes$ingredients[1]
```

```
## [1] "2.25kg/5lb rib-eye of beef, boned and rolled 450ml/Â¾ pint red wine 150ml/Â¼ pin
```

```
recipes$directions[1]
```

```
## [1] "Place the rib-eye of beef into a large non-metallic dish. In a jug, mix together
rare) or 1 hour 50 minutes (for well-done). Meanwhile, for the horseradish cream, mix th
30 minutes. To serve, carve the rib-eye of beef into slices and arrange on warmed plates
```

Defining a curry

We also have “hand-coded” information on whether each dish is really a curry:

```
table(recipes$curry)
```

```
##
## FALSE  TRUE
##  9094   290
```

Defining a curry

```
head(recipes$recipe_name[recipes$curry])
```

```
## [1] "Venison massaman curry"
## [2] "Almond and cauliflower korma curry"
## [3] "Aromatic beef curry"
## [4] "Aromatic blackeye bean curry"
## [5] "Aubergine curry"
## [6] "Bangladeshi venison curry"
```

A curry dictionary

Given that we have some idea of the concept we would like to measure, perhaps we can just use a dictionary:

```
## Convert to corpus
recipe_corpus <- corpus(recipes, text_field = "ingredients")

# Tokenize
recipe_tokens <- tokens(recipe_corpus, remove_punct = TRUE,
                        remove_numbers = TRUE, remove_symbols = TRUE) %>%
                 tokens_remove(c(stopwords("en"),
                    "ml","fl","x","mlâ","mlfl","g","kglb",
                    "tsp","tbsp","goz","oz", "glb", "gâ", "â"))

# Convert to DFM
recipe_dfm <- recipe_tokens %>%
    dfm() %>%
    dfm_trim(max_docfreq = .3,
             min_docfreq = .002,
             docfreq_type = "prop")
```

A curry dictionary

```
## Document-feature matrix of: 9,384 documents, 902 features (98.15% sparse) and 6 docvars.
##        features
## docs    beef boned rolled pint red wine vinegar sugar allspice bay
##   text1    1     1      1    2   2    2       1     1        1   1
##   text2    0     0      0    1   0    1       1     1        0   0
##   text3    0     0      0    0   0    1       0     1        0   0
##   text4    0     0      0    0   0    0       0     0        0   1
##   text5    0     0      0    0   0    0       0     1        0   0
##   text6    0     0      0    0   1    0       0     0        0   0
## [ reached max_ndoc ... 9,378 more documents, reached max_nfeat ... 892 more features ]
```

A curry dictionary

```
topfeatures(recipe_dfm, 20)
```

```
##    finely     sugar     flour    sliced    garlic    peeled       cut freerange
##      3707      3118      2486      2456      2362      2333      2299      2196
##    leaves     juice     white       red     large     extra    caster     seeds
##      1859      1757      1730      1673      1658      1626      1615      1541
##     small vegetable     onion     plain
##      1498      1493      1485      1450
```

A curry dictionary

```
curry_dict <- dictionary(list(curry = c("spices",
                                        "turmeric")))

curry_dfm <- dfm_lookup(recipe_dfm, dictionary = curry_dict)
```

A curry dictionary

```
curry_dfm$recipe_name[order(curry_dfm[,1], decreasing = T)[1:10]]
```

```
##  [1] "Indonesian stir-fried rice (Nasi goreng)"
##  [2] "Pineapple, prawn and scallop curry"
##  [3] "Almond and cauliflower korma curry"
##  [4] "Aloo panchporan (Stir-fried potatoes tempered with five spices)"
##  [5] "Aromatic beef curry"
##  [6] "Asian-spiced rice with coriander-crusted lamb and rosemary oil"
##  [7] "Beef chilli flash-fry with yoghurt rice"
##  [8] "Beef rendang with mango chutney and sticky rice"
##  [9] "Beef curry with jasmine rice"
## [10] "Beef Madras"
```

A curry dictionary

Let's classify a recipe as a "curry" if it includes *any* of our dictionary words

```
recipes$curry_dictionary <- as.numeric(curry_dfm[,1]) > 0

conf_mat_curry_dictionary <- table(recipes$curry_dictionary, recipes$curry)
```

A curry dictionary

```
##           True
## Predicted FALSE TRUE
##     FALSE  8915  195
##     TRUE    179   95
##                    Dictionary
## accuracy           0.96014493
## misclassification 0.03985507
## sensitivity        0.32758621
## specificity        0.98031669
```

$$\text{Sensitivity} = \frac{\#\text{True Positives}}{\#\text{True Positives} + \#\text{False Negatives}}$$

$$\text{Specificity} = \frac{\#\text{True Negatives}}{\#\text{True Negative} + \#\text{False Positives}}$$

Supervised Classification for Text

Supervised Learning vs Dictionaries

Supervised learning for text can be conceptualized as a generalization of dictionary methods.

- Dictionaries:
  - Words associated with each category are pre-specified by the researcher
  - Words typically have a weight of either zero or one
  - Documents are scored on the basis of words they contain
- Supervised learning:
  - Words are associated with categories on the basis of pre-labelled training data
  - Words have are weighted according to their relative prevalence in each each category
  - Documents are scored on the basis of words they contain

Key difference: in supervised learning the features associated with each category (and their relative weight) are **learned** from the data $\longrightarrow$ weights are corpus-specific.

Components of Supervised Learning

- Labelled dataset
  - Labelled (normally hand-coded) data which categorizes texts into different categories
  - Training set: used to train the classifier
  - Test set: used to validate the classifier
- Classification method
  - Statistical method to learn relationship between coded texts and words
  - E.g. Naive Bayes, Logistic Regression, SVM, tree-based methods, many others...
- Validation method
  - Predictive metrics (accuracy, sensitivity, specificity, etc)
  - Cross-validation
- Out-of-sample prediction
  - Use the model to predict categories for documents that have no labels

Creating a labelled datset

- **External sources of annotation**, e.g.
    - Party labels for election manifestos; Authors of Federalist papers
- **Expert annotation**, e.g.
    - Comparative Manifesto Project; undergraduate students ("expertise" comes from training)
- **Crowd-sourced coding**, e.g.
    - Ask random people on the internet to code texts into categories ("wisdom of crowds")

Here, we are cheating by assuming that any dish whose title contains the word "curry" is, in fact, a curry.

In a more serious application, we would hand-code individual curry recipes as "curry" or "not curry", but we are taking a short-cut here.

Language Models

- Probabilistic language models describe a story about how documents are generated using probability
- This *data-generating process* is based on a set of unknown parameters which we infer based on the data
- Once we have inferred values for the parameters, we can calculate the probability that any given document was generated by a particular language model
- The Naive Bayes text classification model is *one* example of a generative language model:
    a. Estimate separate language models for each category of interest
    b. Calculate probability that each text was generated by each model
    c. Assign the text to the category for which it has the highest probability

Language Models

- The basis of any language model is a probability distribution over words in a vocabulary.
- A probability distribution over a discrete variable must have three properties
    - Each element must be greater than or equal to zero
    - Each element must be less than or equal to one
    - The sum of the elements must be 1

Language Models

- Consider a 6 word vocabulary: “coriander”, “turmeric”, “garlic”, “sugar”, “flour”, “eggs”
- When writing a curry recipe, you will
  - frequently use the words “coriander”, “turmeric”, and “garlic”
  - infrequently use the words “sugar”, “flour”, and “eggs”
- When writing a cake recipe, you will
  - frequently use the words “sugar”, “flour”, and “eggs”
  - infrequently use the words “coriander”, “turmeric”, and “garlic”
- We can represent these different “models” for language using a probability distribution over the words in the vocabulary:

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

- Given these models, we can calculate the probability that a given set of word counts (i.e. a document) would be drawn from each distribution

$$P(W_i|\mu) = \frac{M_i!}{\prod_{j=1}^{J} W_{i,j}!} \prod_{j=1}^{J} \mu_j^{W_{ij}}$$

- This is the **multinomial** distribution
- μ_j is the probability of observing word j under a given model
- $W_{i,j}$ is the number of times word j appears in document i (i.e. it is an element of a dfm)
- M_i is the total number of words in document i
- ! is the factorial operator $(n! = n \times (n-1) \times (n-2) \times ... \times 1)$

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Which language model is most likely to have produced each document?

$$P(W_1|\mu_{\text{curry}}) = \frac{10!}{(6!)(2!)(1!)(1!)} \times (.4)^6 \times (.25)^2 \times (.2)^1 \times (.08)^1 = 0.01$$

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Which language model is most likely to have produced each document?

$$P(W_1|\mu_{\text{cake}}) = \frac{10!}{(6!)(2!)(1!)(1!)} \times (.02)^6 \times (.01)^2 \times (.01)^1 \times (.26)^1 = 0.000000000000042$$

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Implication: The probability of observing W_1 is higher under μ_{curry} than under μ_{cake}.

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

$$P(W_2|\mu_{\text{curry}}) = \frac{10!}{(1!)(4!)(2!)(3!)} \times (.4)^1 \times (.26)^4 \times (.4)^2 \times (.3)^3 = 0.0000000089$$

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

$$P(W_2|\mu_{\text{cake}}) = \frac{10!}{(1!)(4!)(2!)(3!)} \times (.02)^1 \times (.26)^4 \times (.4)^2 \times (.3)^3 = 0.005$$

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Implication: The probability of observing W_2 is higher under μ_{cake} than under μ_{curry}.

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

Conclusion: Given a set of probabilities, we can work out which model was *most likely* to have generated any given document.

Language Models

Model	coriander	turmeric	garlic	sugar	flour	eggs
μ_{curry}	0.4	0.25	0.20	0.08	0.04	0.03
μ_{cake}	0.02	0.01	0.01	0.26	0.4	0.3

Imagine we have two documents represented by the following DFM

Document	coriander	turmeric	garlic	sugar	flour	eggs
W_1	6	2	1	1	0	0
W_2	1	0	0	4	2	3

The likelihood of a document being generated by a given model will be

- ...**larger** when the model gives higher probabilities to the words that occur frequently in the document
- ...**smaller** when the model gives higher probabilities to the words that occur infrequently in the document

Naive Bayes

- Naive Bayes is a model that classifies documents into categories on the basis of the words they contain

$$P(y_i = C_k|W_i) = \frac{P(y_i = C_k)P(W_i|y_i = C_k)}{P(W_i)}$$

$P(y_i = C_k|W_i)$ is the **posterior distribution** – this tells us the probability that document i is in category k, given the words in the document and the prior probability of category k

Naive Bayes

$$P(y_i = C_k|W_i) = \frac{P(y_i = C_k)P(W_i|y_i = C_k)}{P(W_i)}$$

- Generally, we will want to make comparisons of the probabilities between different classes
    - e.g. Is $P(y_i = C_{\text{curry}}|W_i) > P(y_i = C_{\text{not curry}}|W_i)$
- This means that we can drop the $P(W_i)$ term and just focus on the likelihood and the prior probabilities

$$P(y_i = C_k|W_i) \propto P(y_i = C_k)P(W_i|y_i = C_k)$$

- where $\propto$ means "proportional to" (rather than "equal than" for $=$)

Naive Bayes

To work out the whether a document should be labelled as belonging to a particular class, we therefore need to work out:

- the **prior probability** ($P(Y = C_k)$) that the document is from category k
  - This is usually estimated by calculating the proportion of documents of category k in the training data
- the **conditional probability** or **likelihood** ($P(W|Y = C_k)$) of the words in the document occurring in category k

$$P(W_i|y_i = C_k) = \frac{M_i!}{\prod_{j=1}^{J} W_{i,j}!} \prod_{j=1}^{J} \mu_{j(k)}^{W_{ij}} \quad (1)$$

$$\propto \prod_{j=1}^{J} \mu_{j(k)}^{W_{ij}} \quad (2)$$

Question: How do we estimate μ?

Naive Bayes Estimation

- $\mu_{j(k)}$ is the probability that word j will occur in documents of category k.
- We can **estimate** these probabilities from our training data:

$$\hat{\mu}_{j(k)} = \frac{W_{j(k)}}{\sum_{j \in V} W_{j(k)}} = \frac{\text{number of times j appears in category k}}{\text{total number of words in category k}}$$

- In the **curry recipes** our training data, we observe...
  - ...77 instances of the word "turmeric" ($W_{\text{turmeric(curry)}} = 77$)
  - ...10586 total words ($\sum_{j \in V} W_{j(\text{curry})} = 10586$)
  - ...and so $\frac{W_{\text{turmeric(curry)}}}{\sum_{j \in V} W_{j(\text{curry})}} = \frac{77}{10586} = 0.007$
- In the **not-curry recipes** our training data, we observe...
  - ...148 instances of the word "turmeric" ($W_{\text{turmeric(not curry)}} = 148$)
  - ...210805 total words ($\sum_{j \in V} W_{j(\text{not curry})} = 210805$)
  - ...and so $\frac{W_{\text{turmeric(not curry)}}}{\sum_{j \in V} W_{j(\text{not curry})}} = \frac{148}{210805} = 0.0007$
- The word "turmeric" is about 10 times more common in curry recipes

Naive Bayes Estimation – Laplace Smoothing

- What happens when a given word doesn't appear at all for one of the classes in our training data?
- Imagine that we never observe the word "duck" in the curry recipes in our training data $\frac{W_{\text{duck(curry)}}}{\sum_{j \in V} W_{j\text{(curry)}}} = \frac{0}{10586} = 0$
- Then, in our test data, we observe "For this curry you will need to coat the duck legs with 1 tsp ground turmeric"
- As we multiply together the individual word probabilities when calculating the probability of a sentence, we will get a probability of zero!
- **Solution**: Add one to the counts for each word in each category ("Laplace" smoothing)

$$\frac{W_{\text{duck(curry)}} + 1}{\sum_{j \in V}(W_{j\text{(curry)}} + 1)} = \frac{1}{10587} = 0.00009$$

Why is Naive Bayes “Naive”?

By treating documents as bags of words we are assuming:

- *Conditional independence* of word counts
    - Knowing a document contains one word doesn’t tell us anything about the probability of observing other words in that document
    - e.g. The fact that a recipe includes the word “turmeric” doesn’t make it any more or less likely that it will also include the word “coriander”
- *Positional independence* of word counts
    - The position of a word within a document doesn’t give us any information about the category of that document
    - e.g. Word order is unimportant

While this is a very simple model of language which is “wrong”, it is nevertheless useful for classification.

Naive Bayes Classification

The classification decision made by the Naive Bayes model is simple: we assign document i to the category, k, for which it has the highest posterior probability:

$$\hat{Y}_i = \underset{k \in \{1,\ldots,k\}}{\operatorname{argmax}} P(y_i = C_k) \times P(W_i | y_i = C_k)$$

where $\underset{k \in \{1,\ldots,k\}}{\operatorname{argmax}}$ means "which category, k, has the *maximum* posterior probability".

Intuition:

- Assign documents to categories when the probability of observing the words in that document are high given the probability distribution for that category (i.e. when $P(W_i | y_i = C_k)$ is large)
- Assign more documents to categories that contain more documents in the training data (i.e. when $P(y_i = C_k)$ is large)

Naive Bayes Application

```
## Training and test set

train <- sample(c(TRUE, FALSE), nrow(recipes), replace = TRUE, prob = c(.8, .2))

test <- !train

## Naive Bayes

recipe_dfm_train <- dfm_subset(recipe_dfm, train)
recipe_dfm_test <- dfm_subset(recipe_dfm, test)
recipe_dfm_test_matched <- dfm_match(recipe_dfm_test,
                                     features = featnames(recipe_dfm_train))

nb_train <- textmodel_nb(x = recipe_dfm_train,
                         y = recipe_dfm_train$curry,
                         prior = "docfreq")
```

Naive Bayes Application

```
## Training and test set

table(recipe_dfm_train$curry)
```

```
##
## FALSE  TRUE
##  7270   237
```

```
table(recipe_dfm_test$curry)
```

```
##
## FALSE  TRUE
##  1824    53
```

Naive Bayes Application

Recall that we are interested in the probability of observing word j given class k, i.e.

$$\mu_{j(k)} = \frac{W_{j(k)}}{\sum_{j \in V} W_{j(k)}}$$

What are these word probabilities for our curry data?

We can examine the probability of each word given each class using the `coef()` function on the `nb_train` object.

Naive Bayes Application

Words with highest probability in the "**curry**" class (i.e. $P(w_j|c_k = \text{"curry"})$):

```
head(sort(nb_train$param[2,], decreasing = TRUE), 20)
```

```
##       seeds      finely   coriander      peeled      garlic
## 0.030701754 0.021518640 0.020148026 0.018366228 0.016447368
##   vegetable      ginger      cloves      leaves       cumin
## 0.015076754 0.014665570 0.014391447 0.013843202 0.013294956
##       green         cut      powder      chilli         red
## 0.013294956 0.013020833 0.012335526 0.012061404 0.010690789
##    turmeric       onion       piece      sliced       large
## 0.010690789 0.010279605 0.010005482 0.009594298 0.009594298
```

Naive Bayes Application

Words with highest probability in the "**not curry**" class (i.e. $P(w_j|c_k = \text{"not curry"})$):

```
head(sort(nb_train$param[1,], decreasing = TRUE), 20)
```

```
##      finely       sugar       flour      sliced      garlic
## 0.021711836 0.018519369 0.014668504 0.014653192 0.013788088
##      peeled         cut   freerange      leaves       white
## 0.013412954 0.013344051 0.013152657 0.010832951 0.010350635
##       juice       extra       large      caster         red
## 0.010281733 0.009692237 0.009646302 0.009608023 0.009516154
##         egg       small       onion       plain   vegetable
## 0.008674016 0.008605114 0.008589802 0.008482621 0.008482621
```

Naive Bayes example prediction

What are the class-conditional word probabilities for “Aromatic blackeye bean curry”?

```
##           p(w|not curry) p(w|curry)
## seeds             0.008      0.031
## finely            0.022      0.022
## coriander         0.005      0.020
## peeled            0.013      0.018
## garlic            0.014      0.016
## ginger            0.005      0.015
## cloves            0.008      0.014
## leaves            0.011      0.014
## cumin             0.002      0.013
## chilli            0.006      0.012
## onion             0.009      0.010
## piece             0.002      0.010
```

Naive Bayes example prediction

Choose board Save

Saved from **bigoven.com**

Aromatic blackeye bean curry

Aromatic blackeye bean curry recipe: This delicious vegan curry recipe is spiced with flavours from the west coast of India. Each serving provides 345kcal ,12g protein, 22g carbohydrate (of which...

Saved by **BigOven**

Beans Curry **3 Ingredient Recipes** **Curry Recipes** **3 Ingredients**

More information...

Naive Bayes example prediction

What are the class-conditional word probabilities for “Schichttorte”?

```
##           p(w|not curry) p(w|curry)
## large              0.010      0.010
## sugar              0.019      0.006
## paste              0.001      0.006
## flour              0.015      0.006
## plain              0.008      0.005
## lemon              0.008      0.004
## freerange          0.013      0.003
## eggs               0.008      0.002
## unsalted           0.005      0.002
## zest               0.005      0.001
## caster             0.010      0.001
## cornflour          0.001      0.001
```

Naive Bayes example prediction

Schichttorte

★ ★ ★ ★ ☆ 2 ratings

Rate this recipe

Naive Bayes example

Which recipes are predicted to have a high curry probability?

```
##  [1] "Chickpea curry with green mango and pomegranate"
##  [2] "Green coconut fish curry"
##  [3] "Thai green prawn curry"
##  [4] "Rogan josh"
##  [5] "Bengal coconut dal"
##  [6] "Lamb madras with bombay potatoes"
##  [7] "Thai-style duck red curry"
##  [8] "Pineapple, prawn and scallop curry"
##  [9] "Aromatic blackeye bean curry"
## [10] "Mussels with dry coconut"
```

Naive Bayes Application

```
##           True
## Predicted FALSE TRUE
##     FALSE  1696   12
##     TRUE    128   41
##                    Naive Bayes
## accuracy            0.92541289
## misclassification   0.07458711
## sensitivity         0.77358491
## specificity         0.92982456
```

Implication:

- We are doing a better job on predicting true positives now (our sensitivity is much higher)
- We are now predicting too many curries that are actually something else (our specificity is a little lower)
- Why? Recall that we are a) treating words as independent, b) possibly over-fitting our training data

Disadvantages of Naive Bayes

- Independence assumption
  - Independence means NB is unable to account for interactions between words
    - e.g. When the word “eggs” appears with the word “sugar” that should indicate something different from when “eggs” appears without the word “sugar”
  - Independence also means that NB is often overconfident
    - Each additional word counts as a new piece of information
  - In some contexts, the independence assumption can decrease predictive accuracy
- Linear classifier
  - Other methods (e.g. SVM) allow the classification probabilities to change *non-linearly* in the word counts
  - e.g. Perhaps seeing the word “eggs” once should have a smaller effect on the probability that the recipe is a curry than seeing the word “eggs” five times

Regularized regression

Assume we have:

- $i = 1, 2, \ldots, N$ documents
- Each document i is in class $y_i = 0$ or $y_i = 1$
- $j = 1, 2, \ldots, J$ unique features
- And x_{ij} as the count of feature j in document i

We could build a linear regression model as a classifier, using the values of β_0, β_1, ..., β_J that minimize:

$$RSS = \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{J} \beta_j x_{ij} \right)^2$$

But can we?

- If $J > N$, OLS does not have a unique solution
- Even with $N > J$, OLS has low bias/high variance (overfitting)

Regularized regression

```
dim(recipe_dfm_train)
```

```
## [1] 7507  902
```

- We have approximately one predictor for every three training observations!
- OLS is not appropriate here

Regularized regression

What can we do? Use the regularized regression tools we learned before! We add a penalty for model complexity, such that we now minimize:

$$\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{J}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{J}\beta_j^2 \rightarrow \text{ridge regression}$$

or

$$\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{J}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{J}|\beta_j| \rightarrow \text{lasso regression}$$

where λ is the **penalty parameter** (to be estimated)

Regularized regression

Reminder: Why do we add a penalty (shrinkage) paramter?

- Reduces the variance
- Identifies the model if J $>$ N
- Some coefficients become zero (feature selection)

The penalty can take different forms:

- Ridge regression: $\lambda \sum_{j=1}^{J} \beta_j^2$ with $\lambda > 0$; and when $\lambda = 0$ becomes OLS
- Lasso $\lambda \sum_{j=1}^{J} |\beta_j|$ where some coefficients become zero

How to find best value of λ? Cross-validation.

Regularized *logistic* regression

On day 5 we covered logistic regression as a classification method:

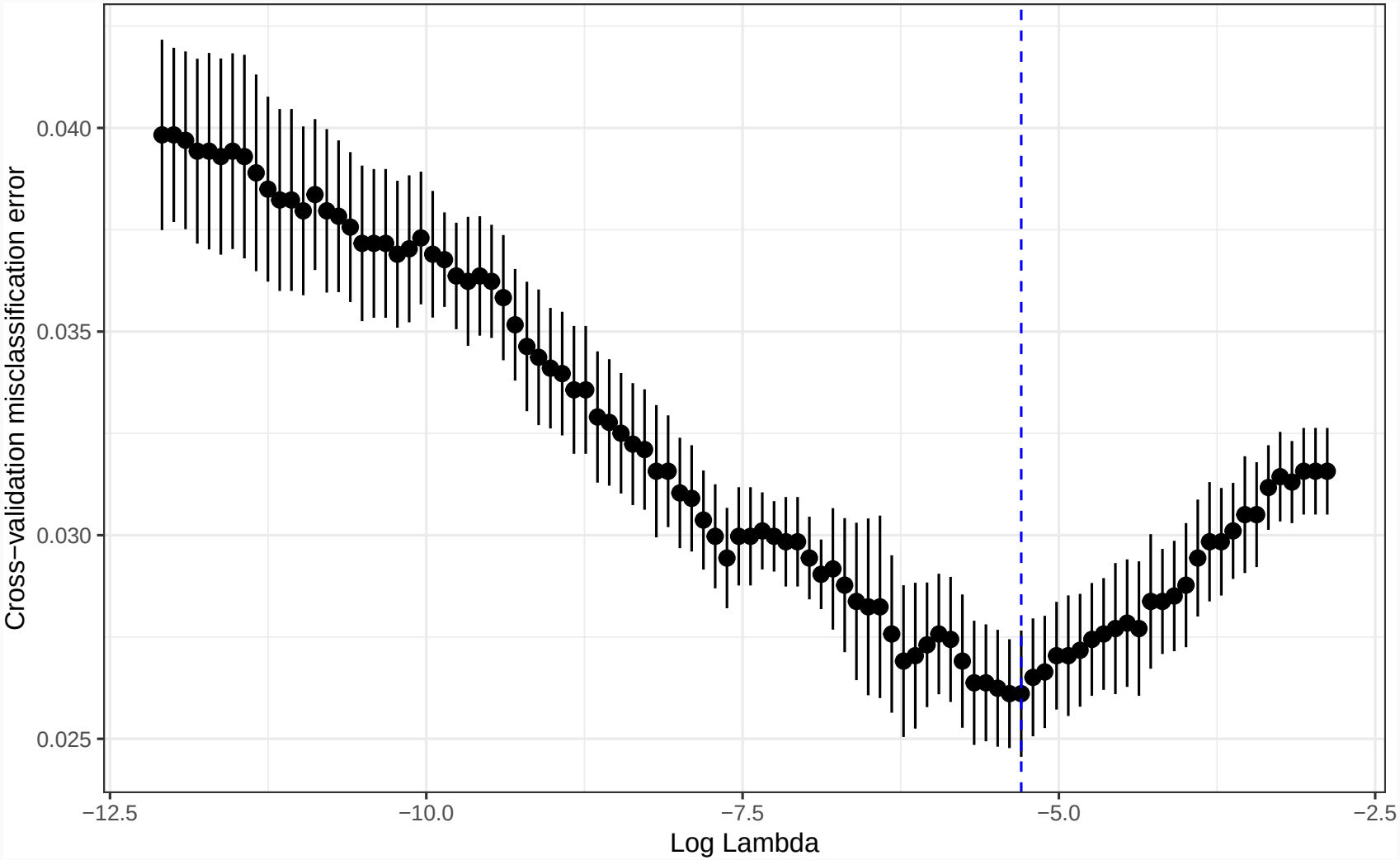
$$log\left(\frac{p(Y=1)}{1-p(Y=1)}\right) = \beta_0 + \sum_{j=1}^{J} \beta_j X_{i,j}$$

- We can easily apply a lasso L_1 penalty (i.e. $\lambda \sum_{j=1}^{J} |\beta_j|$) when estimating the parameters of the logistic model.
- As with OLS, this is achieved by finding the values of β_j that provide good predictions for our training data, subject to the constraint that those estimates are not "too large"
- As with OLS, this will shrink many of the predictors to 0.

Lasso regression example

```
cv_glm_out <- cv.glmnet(x = train_mat,
                        y = recipe_dfm_train$curry,
                        type.measure = "class",
                        family = "binomial", # Logistic regression
                        alpha = 1, # Lasso penalty
                        nfolds = 5) # 5-fold cross-validation
```

Lasso regression example



Lasso regression example

In the cross-validation preferred model, almost all the coefficients are set to zero in the Lasso. Which features have non-zero coefficients?

```
##              feature  coef
## 1              curry 2.111
## 2        lime_leaves 1.548
## 3         salt_taste 1.325
## 4     piece_cinnamon 1.162
## 5     green_chillies 0.964
## 6           turmeric 0.952
## 7        powder_milk 0.927
## 8        curry_paste 0.800
## 9        cumin_seeds 0.770
## 10             garam 0.737
## 11         cardamom 0.737
## 12           masala 0.669
## 13             ghee 0.600
## 14          yoghurt 0.504
## 15         galangal 0.480
## 16 coriander_ground 0.412
```

Lasso regression example

Which recipes are predicted to have a high curry probability?

```
##  [1] "Pineapple, prawn and scallop curry"
##  [2] "Green coconut fish curry"
##  [3] "Rogan josh"
##  [4] "Thai green prawn curry"
##  [5] "Bunny chow"
##  [6] "Ginger and coconut lobster curry with ginger chutney and coconut sticky rice"
##  [7] "Scallop and lobster curry with tamarind sauce and steamed rice"
##  [8] "Bengal coconut dal"
##  [9] "Chickpea curry with green mango and pomegranate"
## [10] "Southern Indian mixed vegetable curry (Avial)"
```

Lasso regression example

```
##           True
## Predicted FALSE TRUE
##     FALSE  1732   15
##     TRUE     92   38
##                         Lasso
## accuracy          0.94299414
## misclassification 0.05700586
## sensitivity       0.71698113
## specificity       0.94956140
```

Implication:

- Our sensitivity is marginally lower than the Naive Bayes model (we may be *under*fitting a little now!)
- Our specificity is higher (we are predicting fewer false positives)

Comparing classification performance

```
##                    Dictionary Naive Bayes Lasso
## accuracy                0.960       0.925 0.943
## misclassification       0.040       0.075 0.057
## sensitivity             0.328       0.774 0.717
## specificity             0.980       0.930 0.950
```

Was #TheStew really #TheCurry?

- The purpose of training a classification model is to make **out-of-sample** predictions
- Generally, we have a small hand-coded training dataset and then we predict for lots of other documents
- Here, we are only predicting for one out-of-sample observation

Was #TheStew really #TheCurry?

- The purpose of training a classification model is to make **out-of-sample** predictions
- Generally, we have a small hand-coded training dataset and then we predict for lots of other documents
- Here, we are only predicting for one out-of-sample observation

```
ingredients <- c("cup olive oil, plus more for serving garlic cloves,
                 chopped large yellow onion, chopped (2-inch) piece ginger,
                 finely chopped Kosher salt and black pepper teaspoons ground
                 turmeric, plus more for serving teaspoon red-pepper flakes,
                 plus more for serving (15-ounce) cans chickpeas,
                 drained and rinsed (15-ounce) cans full-fat coconut milk
                 cups vegetable or chicken stock bunch Swiss chard, kale or
                 collard greens, stems removed, torn into bite-size pieces cup
                 leaves, mint for serving Yogurt, for serving (optional)
                 Toasted pita, lavash or other flatbread, for serving (optional)")
```

Was #TheStew really #TheCurry?

- The purpose of training a classification model is to make **out-of-sample** predictions
- Generally, we have a small hand-coded training dataset and then we predict for lots of other documents
- Here, we are only predicting for one out-of-sample observation

```
dfm_stew <- tokens(ingredients) %>%
            dfm() %>%
            dfm_match(features = featnames(recipe_dfm))

predict(nb_train, newdata = dfm_stew, type = "probability")
```

```
##             FALSE      TRUE
## text1 0.01809137 0.9819086
```

Was #TheStew really #TheCurry?

- The purpose of training a classification model is to make **out-of-sample** predictions
- Generally, we have a small hand-coded training dataset and then we predict for lots of other documents
- Here, we are only predicting for one out-of-sample observation

```
dfm_stew <- tokens(ingredients) %>%
            dfm() %>%
            dfm_match(features = featnames(recipe_dfm))

predict(nb_train, newdata = dfm_stew, type = "probability")
```

```
##             FALSE      TRUE
## text1 0.01809137 0.9819086
```

Yes, #TheStew really was #TheCurry!

Break

(Are you hungry yet?)

Supervised Scaling of Texts

From Classification to Scaling

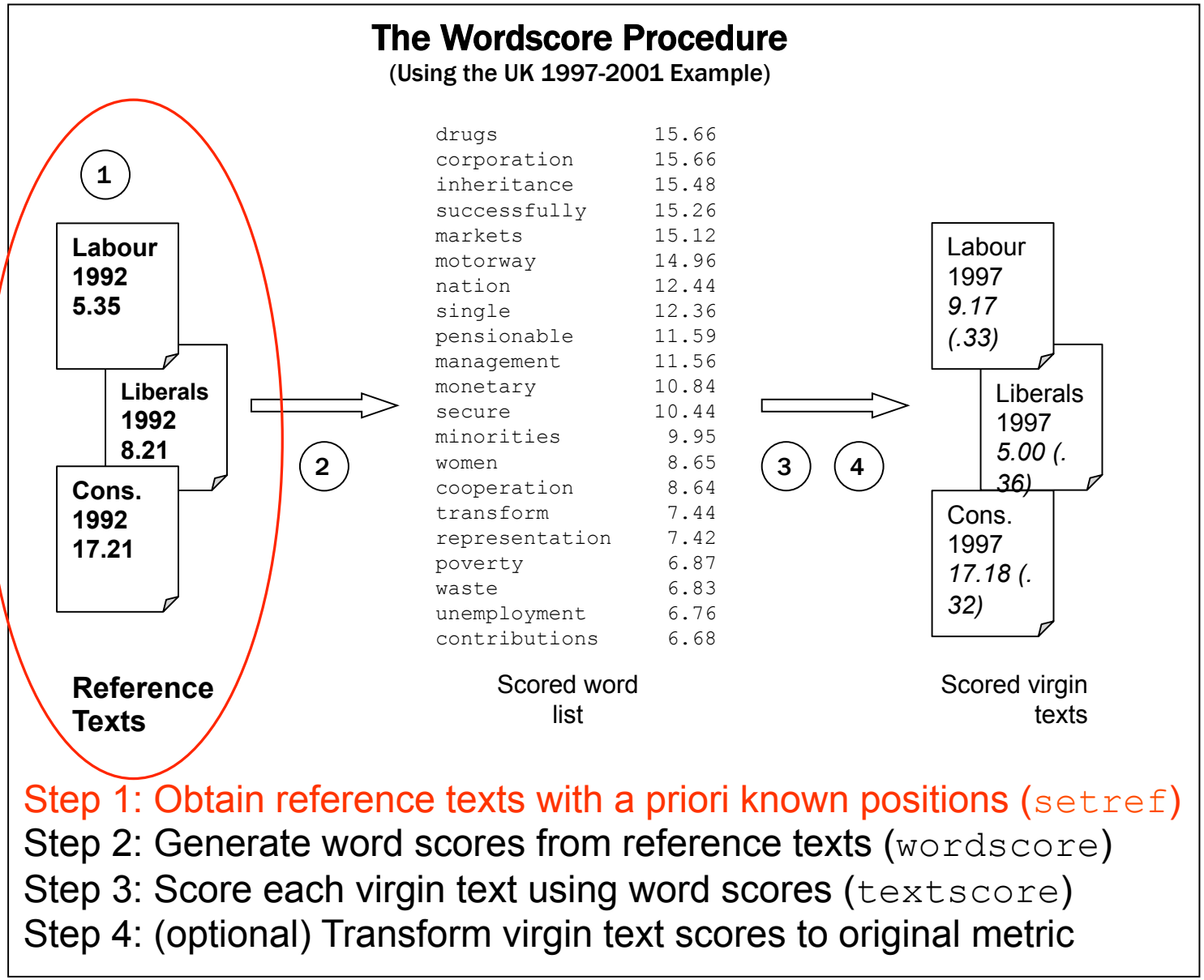
- Machine learning focuses on identifying classes (classification), while social science is often more interested in locating things on latent traits (scaling).
- Spatial models of politics typically describe preferences in terms of ideological “ideal points” which interact with institutions to generate outcomes
- Locating the positions of legislators/parties/media outlets, etc, on different dimensions of politics has become a major area for text-as-data research
- Examples
    - Measuring legislator ideal points from parliamentary speech
    - Measuring party positions from manifestos
    - Measuring the influence of different actors from legislative texts
- Although conceptually distinct, these approaches share similarities with classification methods such as Naive Bayes

Supervised scaling methods

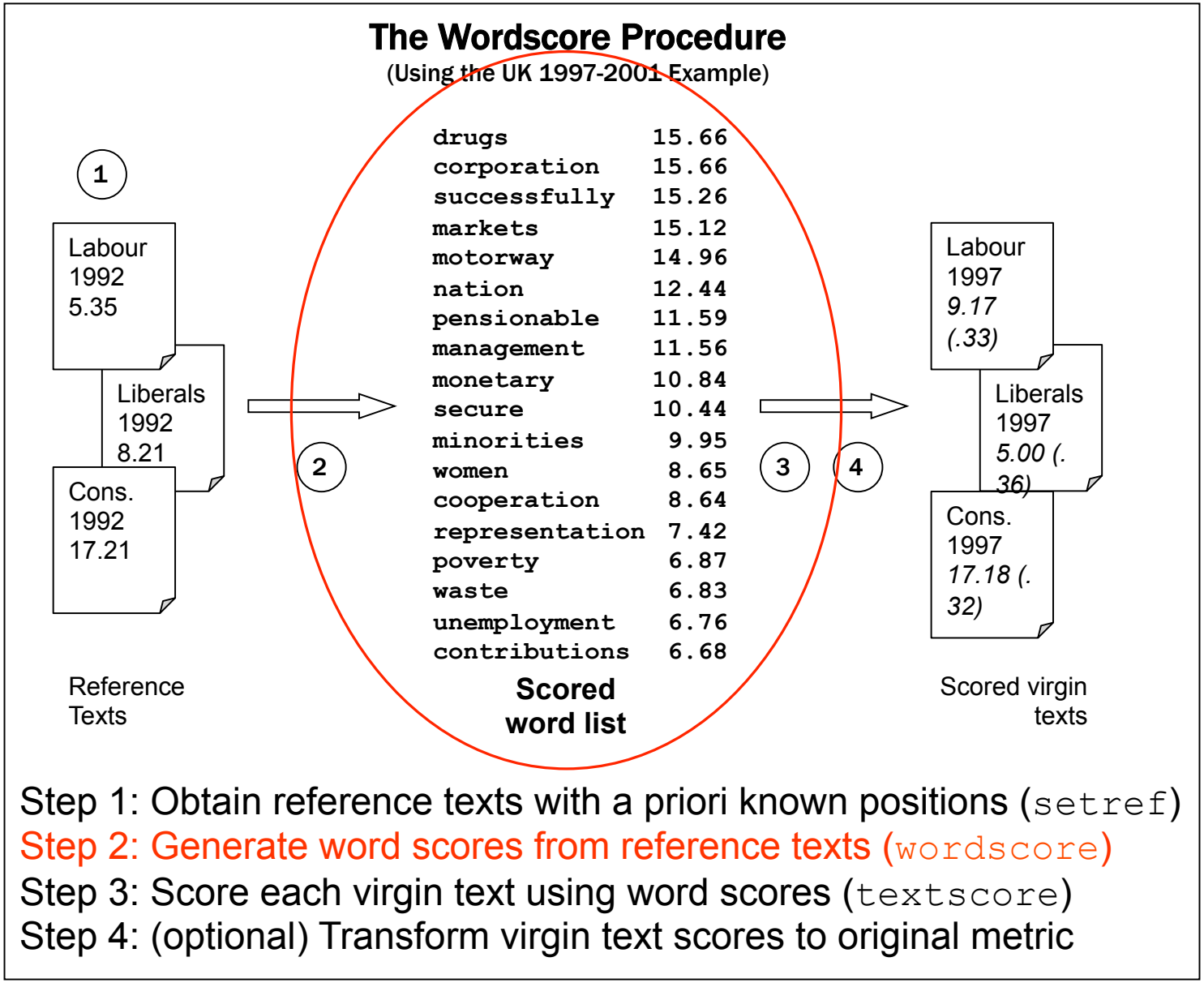
- We start with a set of "reference" texts that define an ideological space
  - These are texts for which we already know their position
  - Equivalent to the training set in a supervised classification task
- We learn the relationship between each word/feature and the texts at different positions in the space, resulting in a score for each word
  - An algorithm that assigns scores to words on the basis of the relative rate that each word is used in each group in the reference texts
  - Equivalent to the model used to predict categories in a classification task
- We use the word scores to assign scores to the remaining ("virigin") texts
  - Equivalent to out-of-sample prediction in supervised classification

A commonly used supervised scaling method is **wordscores** (Laver, Benoit & Garry, 2003).

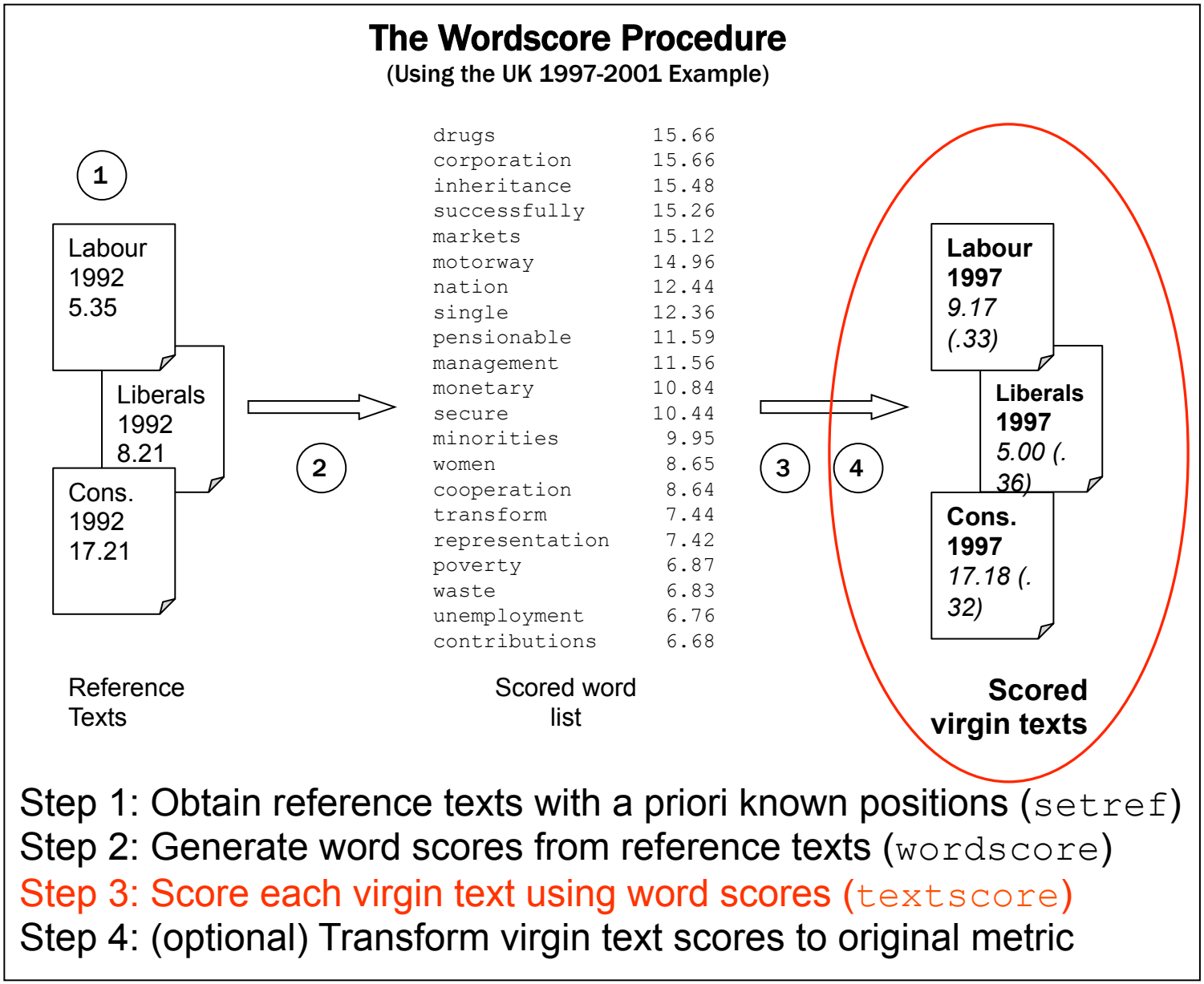
Wordscores (Laver, Benoit & Garry, 2003)



Wordscores (Laver, Benoit & Garry, 2003)

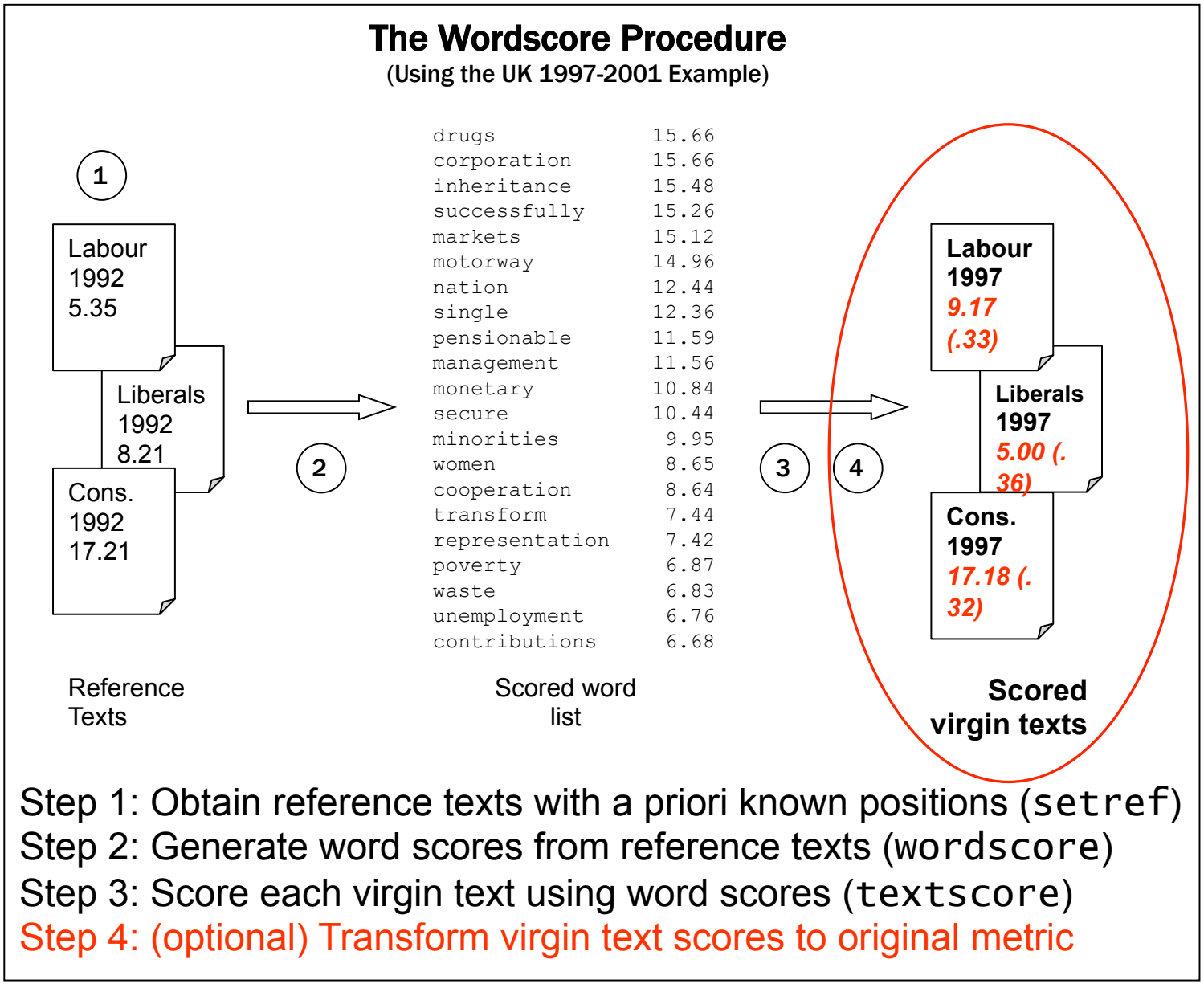


Wordscores (Laver, Benoit & Garry, 2003)



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 (Laver, Benoit & Garry, 2003)



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: Reference texts

- Start with a set of I *reference* texts, represented by an $I \times J$ document-feature matrix W_{ij}, where i indexes the document and j indexes the J total word types
- Each text will have an associated "score", a_i, which is a single number locating this text on a single dimension of difference
  - This can be on a scale metric, such as $1 - 20$
  - Can use arbitrary endpoints, such as $-1, 1$
- We *normalize* the document-feature matrix within each document by converting W_{ij} into a *relative* document-feature matrix (within document), by dividing W_{ij} by its word total marginals:

$$F_{ij} = \frac{W_{ij}}{W_i}$$

- F_{ij} represents the probability of observing word j in document i

Wordscores: Word scores

- Compute an $I \times J$ matrix of relative document probabilities P_{ij} for each word in each reference text, as

$$P_{ij} = \frac{F_{ij}}{\sum_{i=1}^{I} F_{ij}}$$

- This tells us the probability that given the observation of a specific word j, that we are reading a text of a certain reference document i

Wordscores: Word scores

- Assume we have two reference texts, A and B
- The word “sovereignty” is used 10 times per 1,000 words in Text A and 30 times per 1,000 words in Text B
- So $F_{\text{"sovereignty"}} = \{.010, .030\}$
- If we know only that we are reading the word “sovereignty” in one of the two reference texts, then probability is 0.25 that we are reading Text A, and 0.75 that we are reading Text B

$$P_{A\text{”sovereignty”}} = \frac{.010}{(.010 + .030)} = 0.25$$

$$P_{B\text{”sovereignty”}} = \frac{.030}{(.010 + .030)} = 0.75$$

- $P_{i,j}$ therefore represents the probability of observing document i given the presence of word w

Wordscores: Word scores

- To estimate the score for each word, we take the average of each document i's scores a_i, weighted by each word's P_{ij}:

$$S_j = \sum_{i=1}^{I} a_i P_{ij}$$

- This procedure will yield a single "score" for every word that reflects the balance of the scores of the reference documents, weighted by the relative document frequency of its normalized term frequency
- Intuition:
  - When P_{ij} is large (when word j is relatively more common in document i than other documents), the score for document i (a_i) will contribute more to the score for word j
  - When P_{ij} is small (when word j is relatively less common in document i than other documents), the score for document i (a_i) will contribute less to the score for word j

Wordscores: Word scores

For example:

- If we "know" (from independent sources) that reference Text A has a position of -1.0, and Reference Text B has a position of +1.0
- The score of the word "sovereignty" is then
  $0.25(-1.0) + 0.75(1.0) = -0.25 + 0.75 = +0.50$
- Intuition: because sovereignty is more commonly used in text B, its score is closer to B's score than it is to A's

Wordscores: Scoring “virgin” texts

Once we have scores for each word, we can then generate scores for any new text:

- Calculate the mean of the scores of its words, weighted by their term frequency
- So the score S of a virgin document k consisting of the j word types is:

$$S_k = \sum_j (F_{kj} \cdot s_j)$$

where $F_{kj} = \frac{W_{kj}}{W_{k\cdot}}$ as in the reference document relative word frequencies

- Assumes that the relative frequencies of word use in the virgin texts are linked to positions in the same way as the frequencies in the reference texts
- New words (outside of the set J) that appear only in the virgin documents are simply ignored

Wordscores: Rescaling raw text scores

- The scores for each virgin text, S_k, are weighted averages of the word scores
- Most words will have a score that is between the extreme document scores because most words will be used in documents of different types
- As a consequence, the text scores for the virgin documents will be dragged to the interior of the reference scores
- A common approach is to **rescale** the virgin text scores to make them comparable with the virgin text scores
- The consequence of this rescaling is that the virgin scores will have the same variance as the reference scores

What assumptions are we making here?

- The fundamental difficulty in trying to estimate political positions from variation in word use is that there are several more predictive sources of variation in word use:

    1. Language
    2. Style
    3. Topic
    4. ...and only then position, preference, or sentiment

- Any approach that seeks to locate actors ideologically on the basis of words they express – assumes that ideology is the dominant source of variation in word use

- The validity of this **ideological dominance assumption** rests entirely on whether this is plausible for a given set of texts

- Implication: we need to validate our word-scores output by examining the documents at different points on the underlying dimension, and the words associated with different positions

Suggestions for choosing reference texts

- Texts need to contain information representing a clearly dimensional position
- Dimension must be known a priori. Sources might include:
    - Survey scores or manifesto scores
    - Arbitrarily defined scales (e.g. -1.0 and 1.0)
- Should be as discriminating as possible: extreme texts on the dimension of interest, to provide reference anchors
- Need to be from the same lexical universe as virgin texts
- Should contain lots of words

Suggestions for choosing reference values

- Must be “known” through some trusted external source
- For any pair of reference values, all scores are simply linear rescalings, so might as well use (-1, 1)
- The “middle point” will not be the midpoint, however, since this will depend on the relative word frequency of the reference documents
- Reference texts if scored as virgin texts will have document scores more extreme than other virgin texts

Similarity to Naive Bayes

- It turns out that the wordscores approach is *very* similar to a Naive Bayes model
- In particular, under certain conditions the *word*-scores are a linear combination of word-level class posterior probabilities from a NB model
    - $s_j = P(y_i = C_k|w_j) - P(y_i = C'_k|w_j)$
- In practice, the *document*-scores correlate very highly with the document probabilities from NB
- Why not just use NB, then?
    - The class-probabilities in NB are *highly separated* (most observations are close to 1 or 0) because the independence assumption in NB leads to overconfidence.
    - Since we want to *scale* actors, rather than *classify* them, it is better to use the wordscores approach.

Who supports Brexit?

In 2016, the UK government held a referendum on whether the UK should **remain** a member of the European Union or whether the UK should **leave** the EU. The leave-remain divide represented a new dimension of political contestation in UK politics, one which cut across party lines.

Question: Can we use parliamentary speeches to estimate the leave-remain preferences of Members of Parliament on the basis of the speeches they delivered in the House of Commons?

Wordscores Application

We will use data on speeches on debates about the European Union delivered by MPs between 2016 and 2017:

```
glimpse(mps_eu_speeches[,c("name","party", "constituency", "body")])
```

```
## Rows: 448
## Columns: 4
## $ name         <chr> "Diane Abbott", "David Amess", "Kevin~
## $ party        <chr> "Labour", "Conservative", "Labour", "~
## $ constituency <chr> "Hackney North and Stoke Newington", ~
## $ body         <chr> "If the Vote Leave campaigners brough~
```

Note that our data here as been *collapsed* to the MP-level.

Wordscores Application

Let's start by defining a set of reference texts on the basis of MPs with known positions:

```
leavers <- c("Jacob Rees-Mogg", "Bill Cash", "David Davis",
             "John Redwood", "Steven Baker", "Dominic Raab",
             "Peter Bone", "Kate Hoey", "Douglas Carswell",
             "Boris Johnson", "Iain Duncan Smith", "Priti Patel",
             "John Redwood", "Philip Hollobone", "Christopher Chope",
             "Philip Davies", "Edward Leigh", "Gisela Stuart")
```

Wordscores Application

Let's start by defining a set of reference texts on the basis of MPs with known positions:

```
remainers <- c("Anna Soubry", "Dominic Grieve", "Caroline Lucas",
               "Ian Blackford", "Yvette Cooper", "Kenneth Clarke",
               "Chris Bryant", "David Guake", "Rory Stewart", "Ed Vaisey",
               "Chuka Umunna", "Tim Farron", "Jo Swinson", "Edward Davey",
               "Chris Leslie", "Oliver Letwin", "David Lidington",
               "Hilary Benn", "Andy Burnham", "Clive Lewis")
```

Wordscores Application

For our reference texts:

- Score $\alpha_i = -1$ if the MP is a Remainer
- Score $\alpha_i = 0$ if the MP is a Leaver

```
mps_eu_speeches$leave_remain <- NA
mps_eu_speeches$leave_remain[mps_eu_speeches$name %in% remainers] <- -1
mps_eu_speeches$leave_remain[mps_eu_speeches$name %in% leavers] <- 1

table(mps_eu_speeches$leave_remain, useNA = "always")
```

```
##
##   -1    1 <NA>
##   15   16  417
```

NA here are our “virgin” texts

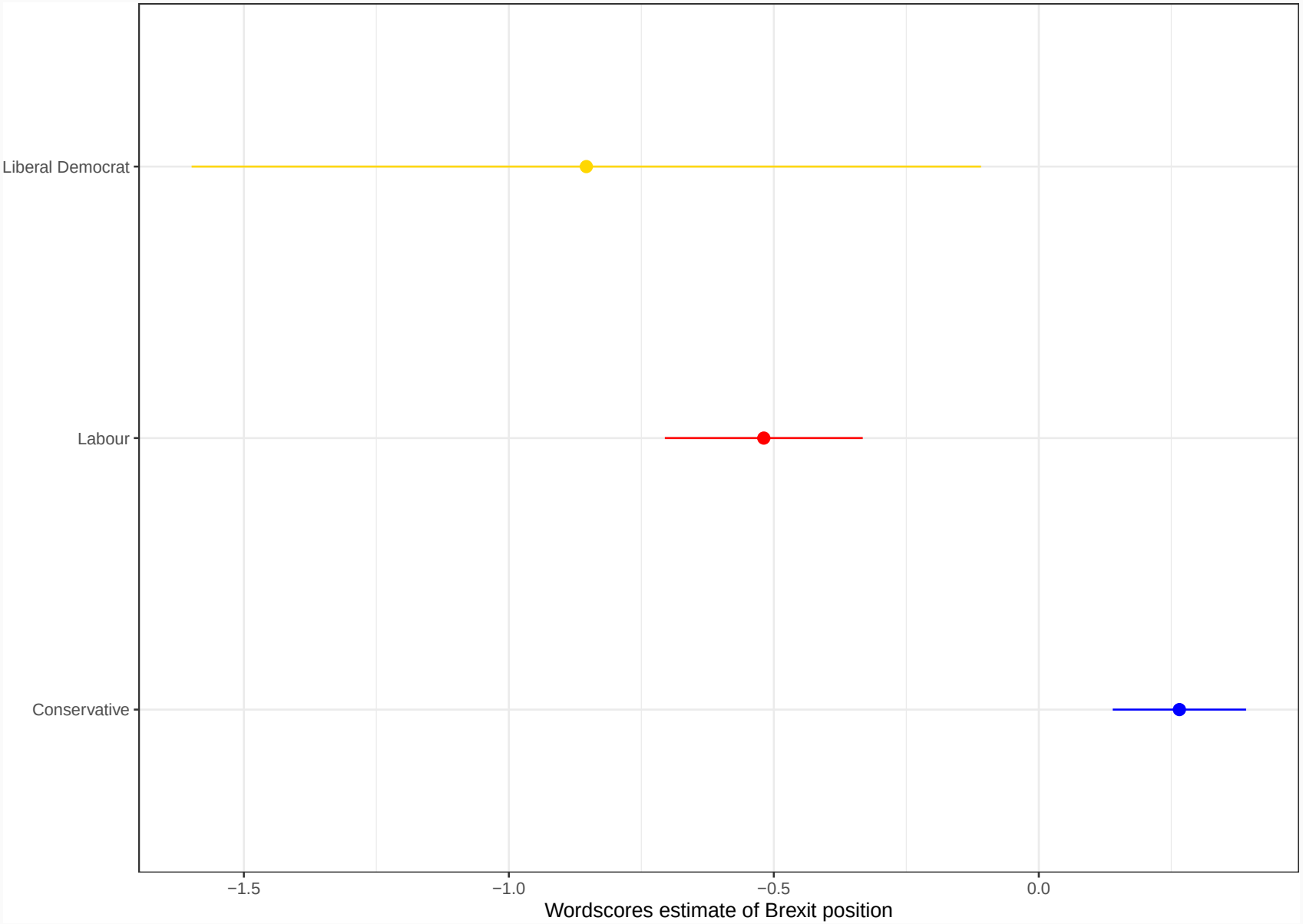
Wordscores Application

```
# Convert to corpus, tokens, and dfm
eu_dfm <- mps_eu_speeches %>%
  corpus(text_field = "body") %>%
  tokens(remove_punct = TRUE) %>%
  tokens_remove(stopwords("en")) %>%
  tokens_ngrams(1:2) %>%
  dfm()

# Trim common and rare words
eu_dfm <- eu_dfm %>%
  dfm_trim(min_docfreq = 0.01,
           max_docfreq = .9,
           docfreq_type = "prop")
```

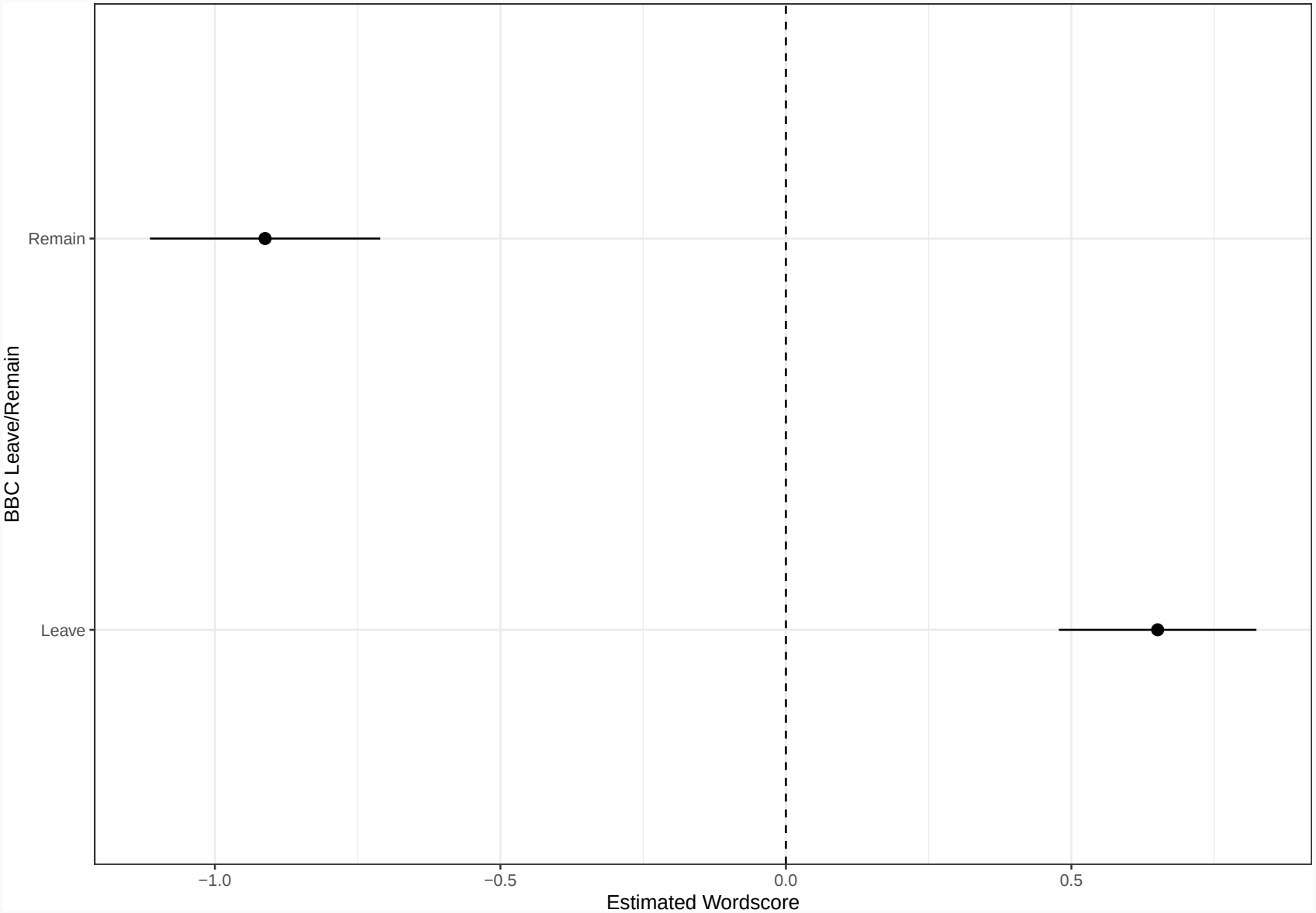
Wordscores Application

```
# Estimate wordscores model on reference texts
wordscores_mod <- textmodel_wordscores(eu_dfm, eu_dfm$leave_remain)

# Predict for virgin texts
mps_eu_speeches$wordscore <- predict(wordscores_mod,
                                     eu_dfm,
                                     rescaling = "lbg")
```

Wordscores Application



Wordscores Application



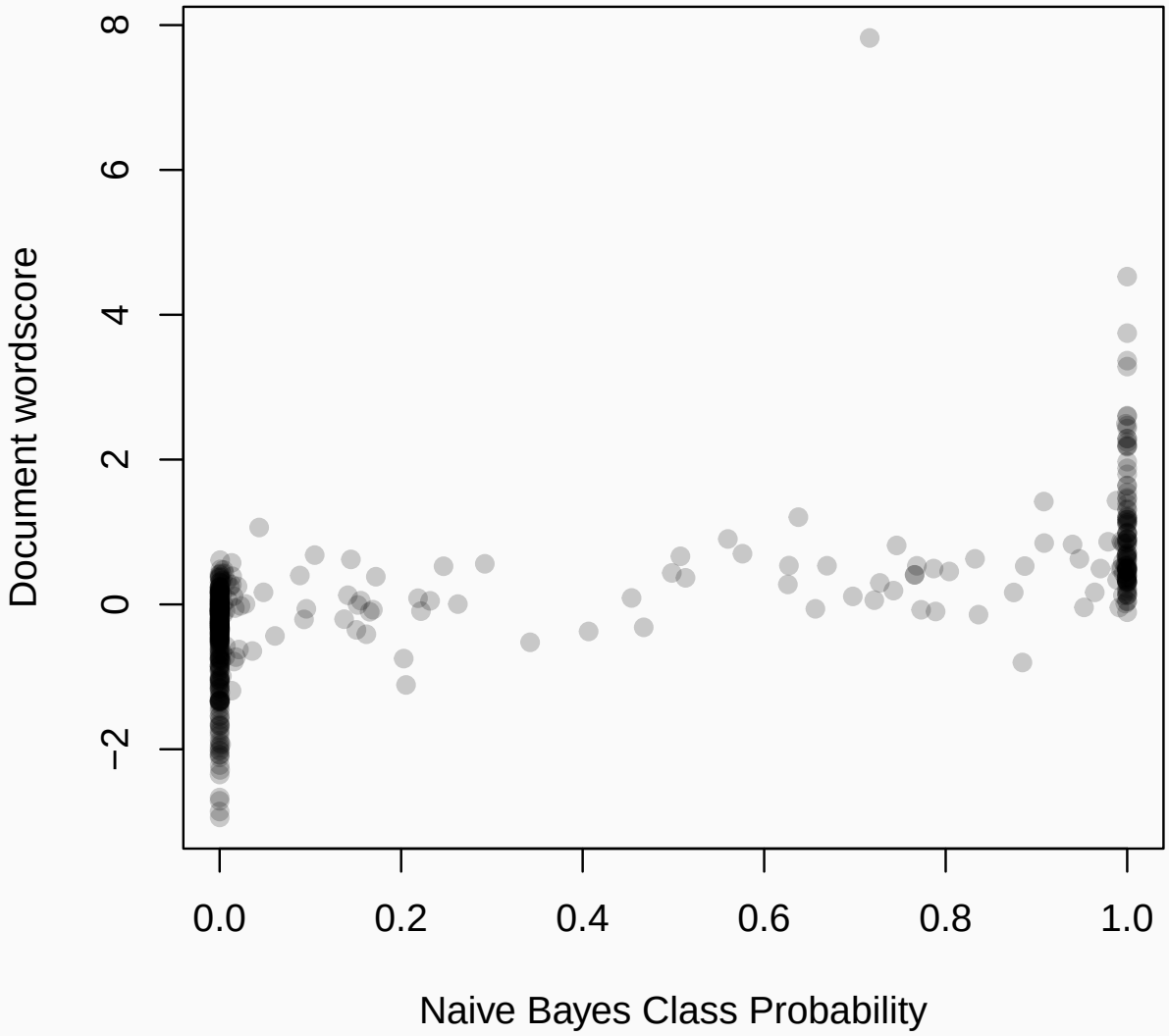
Wordscores Application

Remain words	Leave words
figures	friend_making
commons_library	acceptable
america	even_though
odds	climate_change
left_eu	home_office
fifth_largest	colleague
largest_economy	profound
leave_hon	us_government
net_migration	us_work
wording	differently
hospitals	making_powerful
group	best_served
heath	constraints
backdrop	bus
economists	campaigner

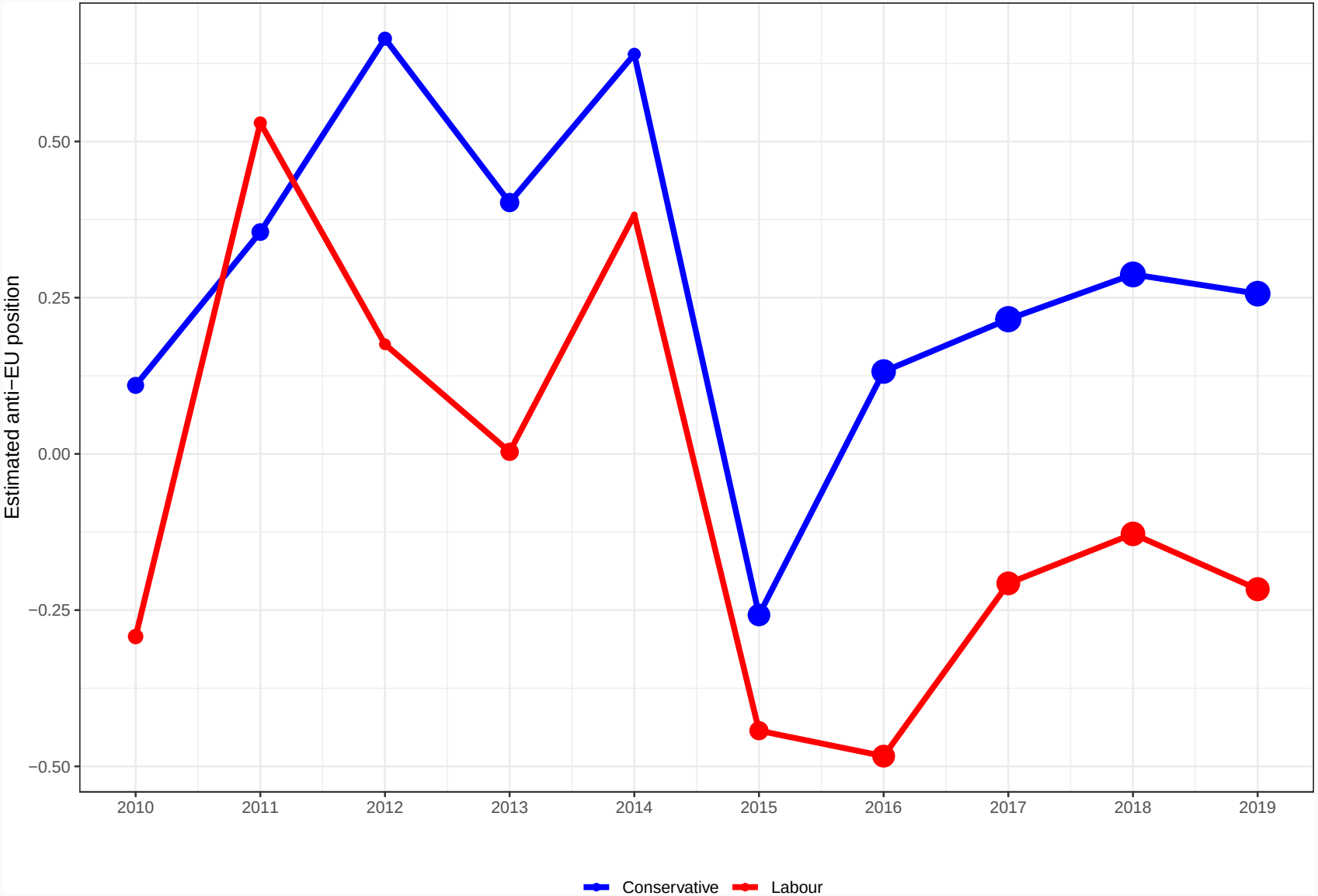
Wordscores Application: Comparison with Naive Bayes

```
nb_mod <- textmodel_nb(eu_dfm, eu_dfm$leave_remain)

mps_eu_speeches$nb <- predict(nb_mod, eu_dfm, type = "probability")[,2]

cor(mps_eu_speeches$nb, mps_eu_speeches$wordscore)

## [1] 0.6248488
```

Wordscores Application: Comparison with Naive Bayes



Wordscores Application



Unsupervised Scaling of Texts

Unsupervised scaling methods

- We start texts we believe contain information an ideological space of interest
    - We do not have any information about the position of each document
    - There is no training set!
- We estimate an ideological space, positioning documents on the underlying dimension on the basis of differences in word use
    - A document's position is assumed to affect the rate at which words are used
- We *interpret* the resulting ideological space to work out what it "means"

Unsupervised methods scale distance between documents

- At a high level, unsupervised models scale documents on the basis of the similarity or differences in word use
    - Documents using similar words will be located closer together, documents using different words will be located further apart
- In order to interpret the output of unsupervised models as *ideological positions*, we need to again make the **ideological dominance assumption**
    - We have to believe that the primary source of linguistic variation between political actors is ideology
- Unsupervised scaling models will capture the main source of variation, whatever that is
    - Careful validation is therefore essential

Unsupervised scaling methods

Parametric methods model feature occurrence according to some stochastic distribution, typically in the form of a measurement model

- For instance, model words as a multi-level Bernoulli distribution, or a Poisson distribution
- Word effects and “positional” effects are unobserved parameters to be estimated
- e.g. Wordfish (Slapin and Proksch 2008) and Wordshoal (Lauderdale and Herzog 2016)

Wordfish (Slapin and Proksch 2008)

- Goal: unsupervised scaling of ideological positions
- The frequency with which politician i uses word k is drawn from a Poisson distribution:

$$w_{ik} \sim Poisson(\lambda_{ik})$$

$$\lambda_{ik} = exp(\alpha_i + \psi_k + \beta_k * \theta_i)$$

- with latent parameters:
  - α_i is the "loquaciousness" of politician i (i.e. a fixed-effect for politician i)
  - ψ_k is frequency of word k
  - β_k is the "discrimination" parameter of word k
  - θ_i is the politician's "ideological" position
- Key intuition: controlling for document length and word frequency, words with negative β_k will tend to be used more often by politicians with negative θ_i (and vice versa)

Wordfish (Slapin and Proksch 2008)

Why Poisson?

- Poisson-distributed variables are bounded between (0, ∞) and take on only discrete values 0,1,2,...,∞
- Exponential transformation: word counts are function of log document length and word frequency

$$\lambda_{ik} = exp(\alpha_i + \psi_k + \beta_k * \theta_i)$$

$$log(\lambda_{ik}) = \alpha_i + \psi_k + \beta_k * \theta_i$$

How to estimate this model

Conditional maximum likelihood estimation:

- If we knew ψ and β (the word parameters) then we have a Poisson regression model
- If we knew α and θ (the party / politician / document parameters) then we have a Poisson regression model too!
- So we alternate them and hope to converge to reasonable estimates for both
- Implemented in the quanteda package as `textmodel_wordfish()`

An alternative is MCMC with a Bayesian formulation or variational inference using an Expectation-Maximization algorithm (Imai et al 2016)

Conditional maximum likelihood for wordfish

Start by guessing the parameters (some guesses are better than others, e.g. SVD)

Algorithm:

1. Assume the current legislator parameters are correct and fit as a Poisson regression model
2. Assume the current word parameters are correct and fit as a Poisson regression model
3. Normalize θ_i to mean 0 and variance 1

Iterate until convergence (change in values is below a certain threshold)

Identification

The scale and direction of θ is undetermined — like most models with latent variable models.

To identify the model in Wordfish we fix the relative position of two θ_is to specify the "direction" of the dimension

```
wordfish_model <- textmodel_wordfish(my_dfm, dir = c(1,2))
```

Where `dir = c(1,2)` is used to specify that $\hat{\theta}_1 < \hat{\theta}_2$ and thus identifies the model.

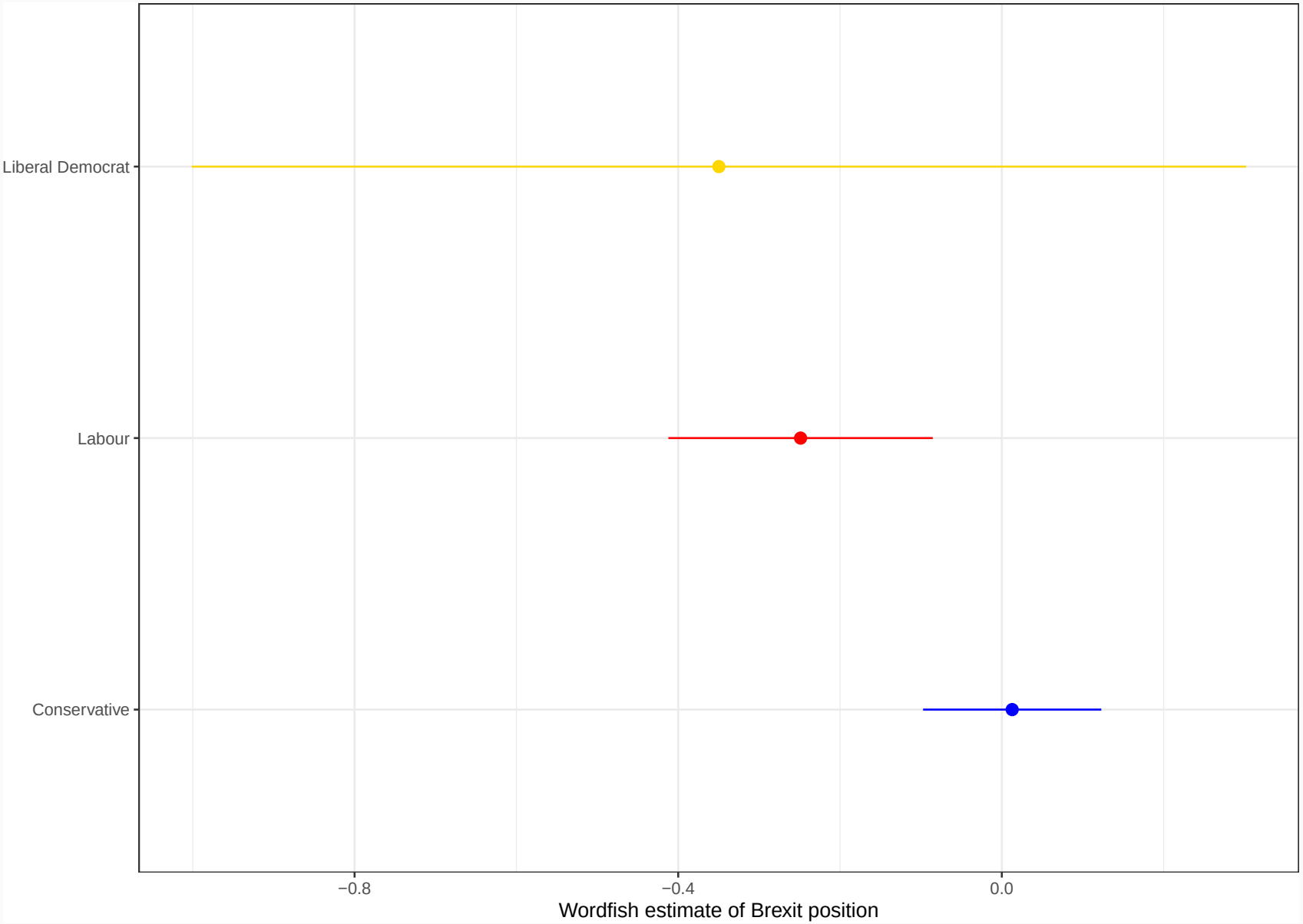
Wordfish Application

```
wordfish_model <- textmodel_wordfish(
    eu_dfm,
    dir = c(which(mps_eu_speeches$name == "Anna Soubry"),
            which(mps_eu_speeches$name == "Jacob Rees-Mogg"))
                                    )
```

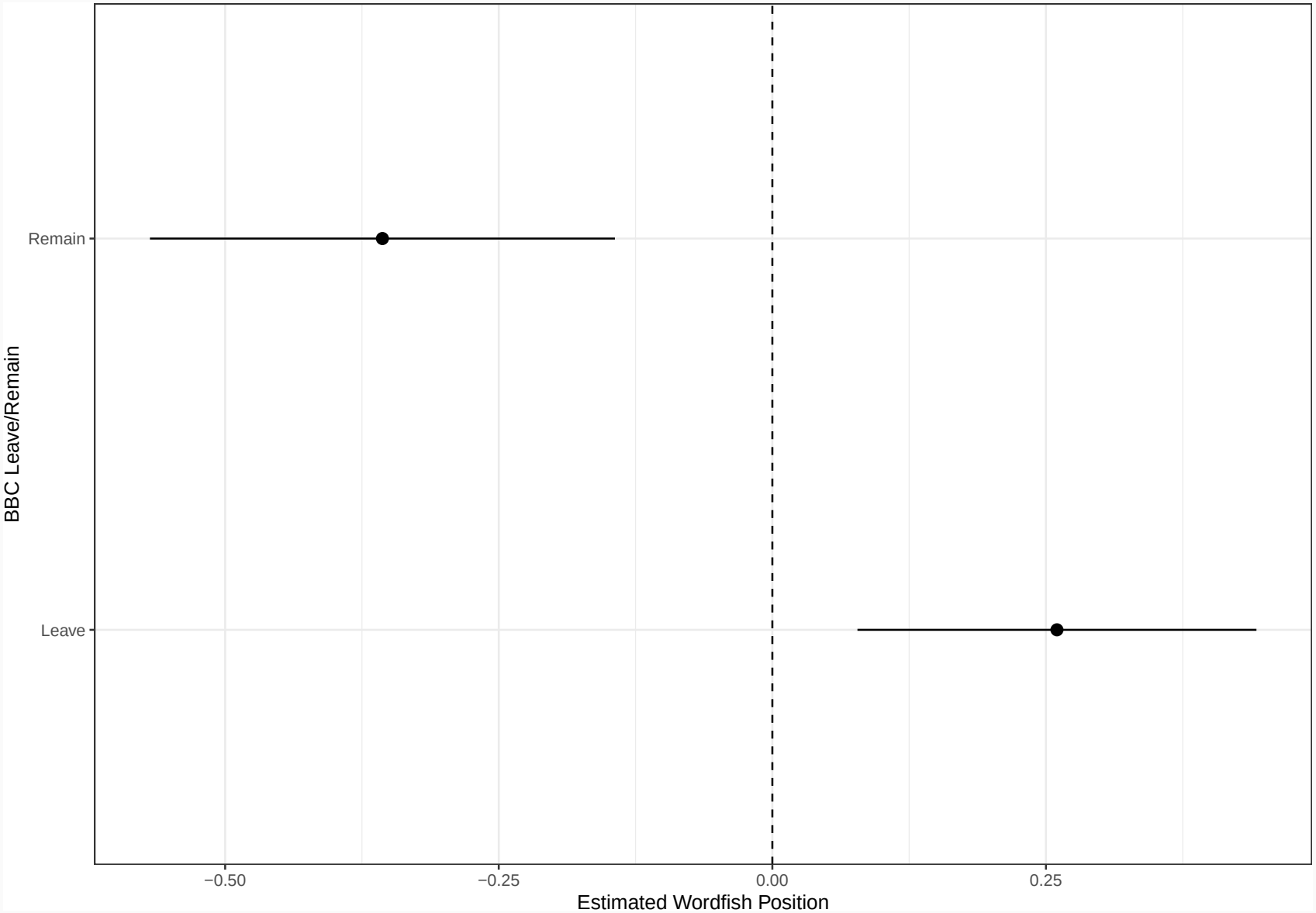
```
mps_eu_speeches$wordfish <- wordfish_model$theta
cor(mps_eu_speeches$wordfish, mps_eu_speeches$wordscore)
```

```
## [1] 0.04967368
```

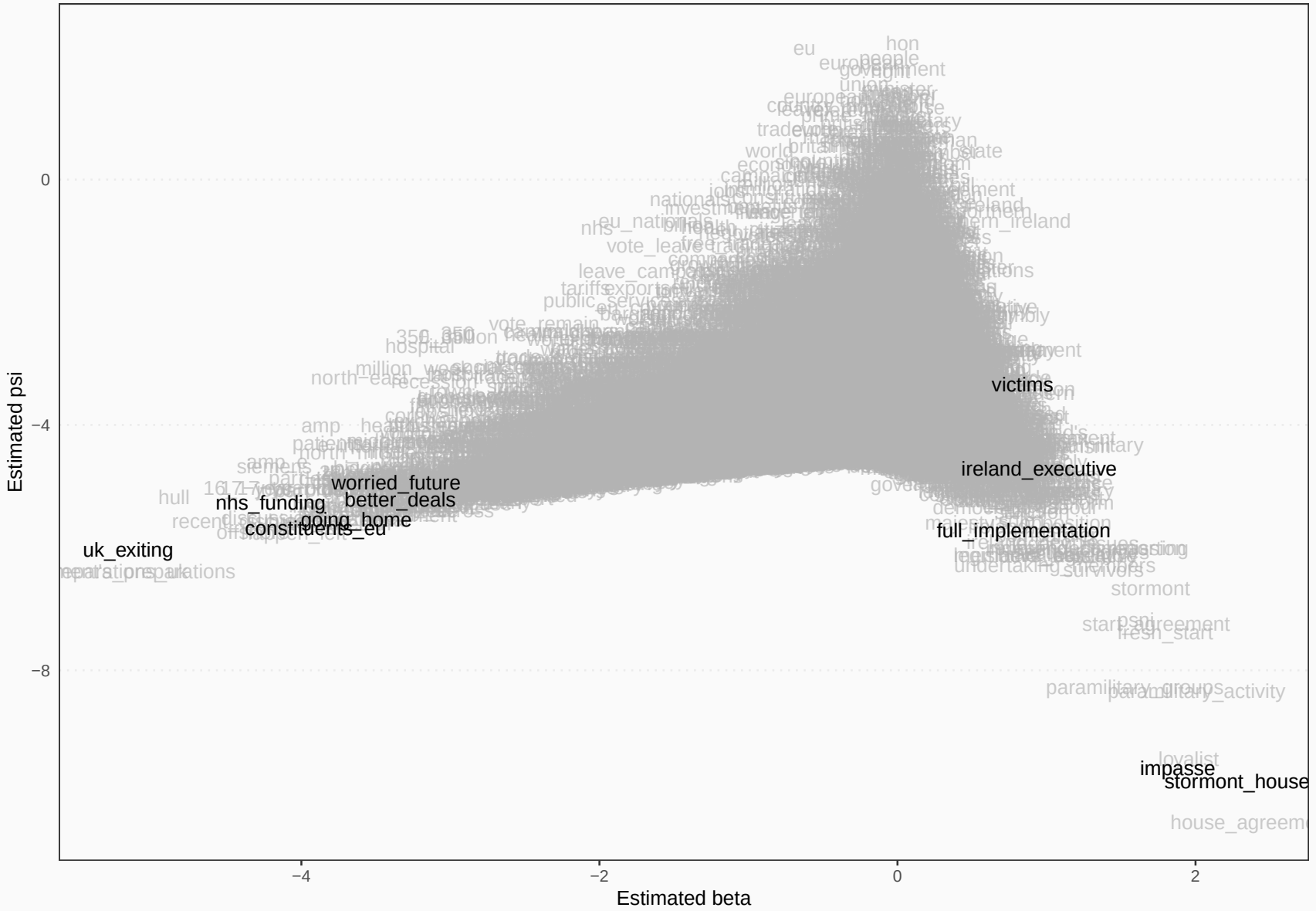
Wordfish Application



Wordfish Application

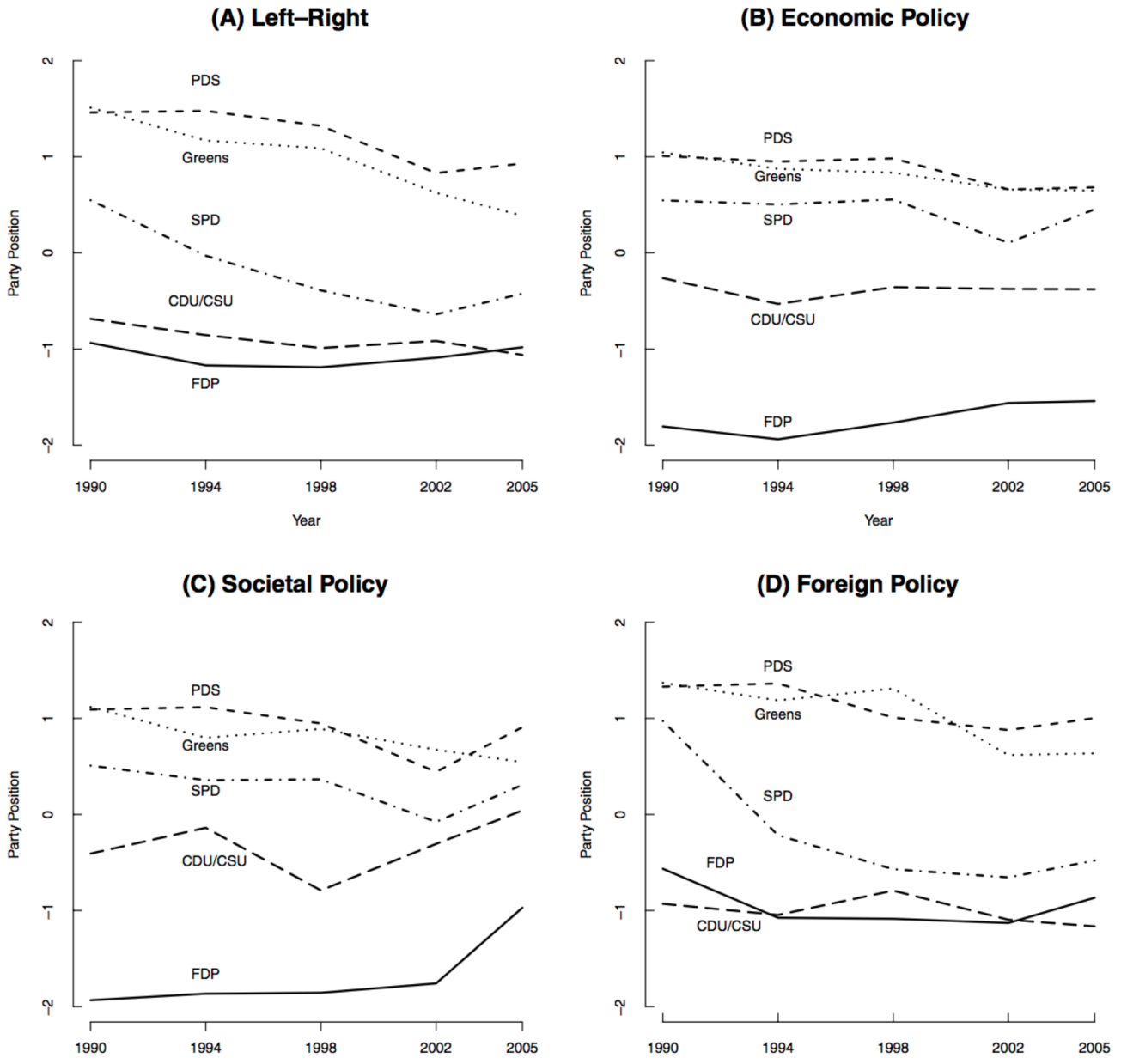


Wordscores Application



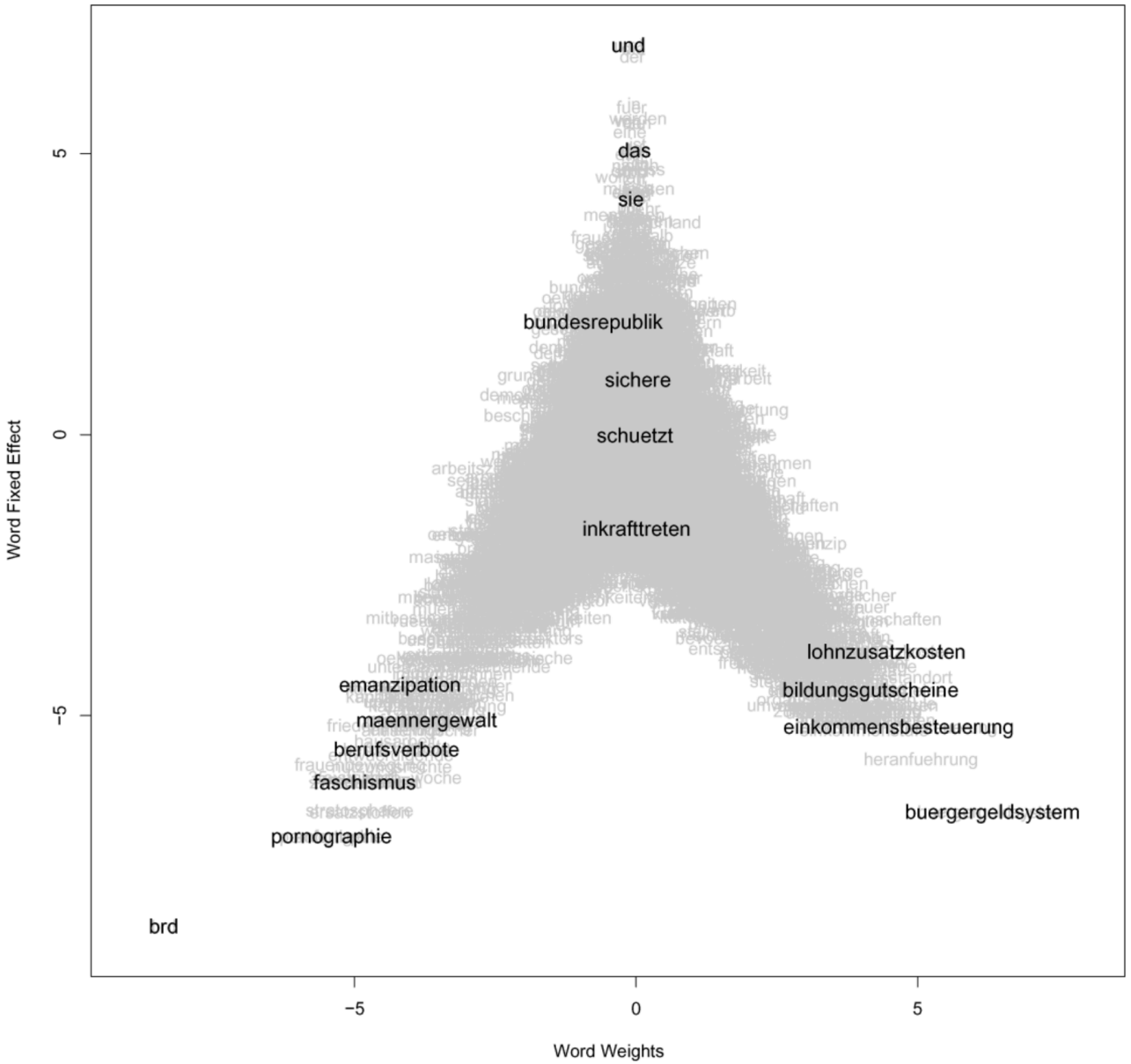
Application to German Political Manifestos

FIGURE 1 Estimated Party Positions in Germany, 1990–2005



Application to German Political Manifestos

FIGURE 2 **Word Weights vs. Word Fixed Effects. Left-Right Dimension, Germany 1990–2005 (Translations given in text)**



Application to German Political Manifestos

TABLE 1 Top 10 Words Placing Parties on the Left and Right

	Top 10 Words Placing Parties on the. . .	
Dimension	**Left**	**Right**
Left-Right	Federal Republic of Germany (BRD)	general welfare payments (Bürgergeldsystem)
	immediate (sofortiger)	introduction (Heranführung)
	pornography (Pornographie)	income taxation (Einkommensbesteuerung)
	sexuality (Sexualität)	non-wage labor costs (Lohnzusatzkosten)
	substitute materials (Ersatzstoffen)	business location (Wirtschaftsstandort)
	stratosphere (Stratosphäre)	university of applied sciences (Fachhochschule)
	women's movement (Frauenbewegung)	education vouchers (Bildungsgutscheine)
	fascism (Faschismus)	mobility (Beweglichkeit)
	Two thirds world (Zweidrittelwelt)	peace tasks (Friedensaufgaben)
	established (etablierten)	protection (Protektion)
Economic	Federal Republic of Germany (BRD)	to seek (anzustreben)
	democratization (Demokratisierung)	general welfare payments (Bürgergeldsystem)
	to prohibit (verbieten)	inventors (Erfinder)
	destruction (Zerstörung)	mobility (Beweglichkeit)
	mothers (Mütter)	location (Standorts)
	debasing (entwürdigende)	negotiated wages (Tarif-Löhne)
	weeks (Wochen)	child-raising allowance (Erziehungsgeld)
	quota (Quotierung)	utilization (Verwertung)
	unprotected (ungeschützter)	savings (Ersparnis)
	workers' participation (Mitbestimmungs-möglichkeiten)	reliable (verlässlich)

Application to German Political Manifestos

TABLE 2 Cross-Validation: Correlations between German Party Position Estimates

	Poisson Scaling Model			
	Left-Right	**Economic**	**Societal**	**Foreign**
Hand-coding manifestos				
CMP: Left-Right (n = 15, 1990–1998)	−0.82			
CMP: Markeco (n = 15, 1990–1998)		0.81		
CMP: Welfare (n = 15, 1990–1998)			0.58	
CMP: Intpeace (n = 15, 1990–1998)				0.81
Expert Survey				
Benoit/Laver 2006: Left-Right (n = 5, 2002)	−0.91			
Benoit/Laver 2006: Taxes-Spending (n = 5, 2002)		0.86		
Wordscores				
Laver et al. 2003: Economic (n = 10, 1990–1994)		0.93		
Laver et al. 2003: Social (n = 10, 1990–1994)			−0.47	
Proksch/Slapin 2006: Economic (n = 5, 2005)		0.98		
Proksch/Slapin 2006: Social (n = 5, 2005)			−0.47	

Conclusion

- Many of the supervised learning methods that we have covered can be easily applied to text data
- Text data is *very* high dimensional, so regularisation methods can be helpful in improving predictive accuracy
- When our interest is in *scaling* rather than *classifying* documents, we can use a different set of tools, many of which share similarities with classification methods
- Particularly with unsupervised scaling, we need to work hard to *interpret* the resulting dimensions
- Tomorrow we cover topic models, where interpretation is also very important!
- Curry is delicious, but so are some things that are not curries