Day 10: Classification and Scaling with Texts

ME314: Introduction to Data Science and Machine Learning

Jack Blumenau

26th July 2022

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?

Alison Roman's famous stew recipe *is* problematic; it's essentially a watered down chana masala or Carribean chickpea curry. She needs to do more to acknowledge these cultural sources and ingredients. And perhaps we need to reflect on why a watered down ethnic recipe by a white woman is so much more accessible and popular than a traditional dish.

Despite using Asian ingredients such as those mentioned in the title and other spices, Roman insisted that her stew was not a curry.

But this isn't a stew, nor is it a soup, as I saw some other cooking blogs try to claim. This is a curry. These are the ingredients *of a curry*. Yes, "curry" is itself a broad description of a certain kind of dish that has different manifestations throughout Asia, that has been adapted countless times. I don't want to be too precious with language here, yet Roman's refusal to acknowledge *the reality of the dish she made* is immensely aggravating.

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?

```
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

```
## Set up "curry" vector

recipes$curry <- grepl("curry", recipes$recipe_name, ignore.case = TRUE)

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, tokens, and dfm

recipe_corpus <- corpus(recipes, text_field = "ingredients")

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"))

recipe_tokens_ngram <- tokens_ngrams(recipe_tokens, 1:2)

recipe_dfm <- recipe_tokens_ngram %>%
    dfm() %>%
    dfm_trim(max_docfreq = .3, min_docfreq = .002, docfreq_type = "prop") %>%
    dfm_remove(c(stopwords("en"),"ml","fl","x","mlâ","mlfl","g","kglb"))
```

A curry dictionary

```
## Document-feature matrix of: 9,384 documents, 2,324 features (98.81% 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 ... 2,314 more features ]
```

A curry dictionary

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

```
##         finely          sugar finely_chopped          flour         sliced
##          3707           3118           2675           2486           2456
##        garlic         peeled            cut      freerange         leaves
##          2362           2333           2299           2196           1859
##         juice          white            red          large          extra
##          1757           1730           1673           1658           1626
##        caster   caster_sugar          seeds          small      vegetable
##          1615           1605           1541           1498           1493
```

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
```

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

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

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
```

Implication:

- We can pick up some signal with the dictionary, but we are clearly not doing a great job of classifying curries!
- Our sensitivity is a very low

Supervised Classification for Text

Supervised Learning Recap

Goal: classify documents into pre existing categories. e.g. authors of documents, sentiment of tweets, ideological position of parties based on manifestos, tone of movie reviews, etc.

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, K-nearest neighbors, BART, ensemble methods...
- Approach to validate classifier: cross-validation
- Performance metric to choose best classifier and avoid overfitting: confusion matrix, accuracy, precision, recall...

Creating a labelled datset

How do we obtain a labeled set?

- External sources of annotation
    - Disputed authorship of Federalist papers estimated based on known authors of other documents
    - Party labels for election manifestos
    - Legislative proposals by think tanks
- Expert annotation
    - “Canonical” dataset in Comparative Manifesto Project
    - In many projects, undergraduate students (expertise comes from training)
- Crowd-sourced coding
    - Wisdom of crowds: aggregated judgments of non-experts converge to judgments of experts at much lower cost (Benoit et al, 2016)
    - Easy to implement with CrowdFlower or MTurk

Supervised vs unsupervised methods for text

- The goal (in text analysis) is to differentiate documents from one another, normally treating them as “bags of words”
- Different approaches:
    - Supervised methods require a training set that exemplify contrasting classes, identified by the researcher
    - Unsupervised methods scale documents based on patterns of similarity from the term-document matrix, without requiring a training step
- Relative advantage of supervised methods:
    - You already know the dimension being scaled, because you set it in the training stage
- Relative disadvantage of supervised methods:
    - You *must* already know the dimension being scaled, because you have to feed it good sample documents in the training stage

Supervised learning vs dictionaries

- Dictionary methods:
    - Advantage: not corpus-specific, cost to apply to a new corpus is trivial
    - Disadvantage: not corpus-specific, so performance on a new corpus is unknown (domain shift)
- Supervised learning can be conceptualized as a generalization of dictionary methods, where features associated with each categories (and their relative weight) are learned from the data
- They will often outperform dictionary methods in classification tasks, particularly when the training sample is large enough

Multinomial Bayes model of Class given a Word

Consider J word types distributed across N documents, each assigned one of K classes.

At the word level, Bayes Theorem tells us that:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

Multinomial Bayes model of Class given a Word

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

- The word likelihood within class
- The maximum likelihood estimate is simply the proportion of times that word j occurs in class k,
- Common to use "Laplace" smoothing by adding 1 to each observed count within class

Multinomial Bayes model of Class given a Word

Word probabilities:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

- This represents the word probability from the training corpus
- Usually uninteresting, since it is constant for the training data, but needed to compute posteriors on a probability scale

Multinomial Bayes model of Class given a Word

Class prior probabilities:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

- This represents the class prior probability
- Machine learning typically takes this as the document frequency in the training set

Multinomial Bayes model of Class given a Word

Class posterior probabilities:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

- This represents the posterior probability of membership in class k for word j
- Key for the classifier: in new documents, we only observe word distributions and want to predict class

Moving to the document level

Recall the Naive Bayes classification model:

- Assume that features (words) are independent
- The probability of a document, Y_i, being assigned to a class, k:

$$P(Y_i = k|W_i) \propto P(k) \prod_{j=1}^{J} P(w_j|k)$$

- We then assign the document to kth class for which it has the highest posterior probability:

$$\hat{Y}_i = \underset{k \in \{1,\ldots,k\}}{\operatorname{argmax}} P(k) \prod_{j=1}^{J} P(w_j|k)$$

Moving to the document level

Note that this is certainly not a correct model of text!

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)
- *Positional independence* of word counts (the position of a word within a document doesn't give us any information about the class probability of that document)

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 w_j given class C_k:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

What are these word likelihoods for our curry data?

We can examine the `nb_train$param` object to see the probability of each word given each class.

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
##    0.017350891    0.012161115    0.011386522    0.010379551
##         garlic      vegetable finely_chopped         ginger
##    0.009295120    0.008520527    0.008443067    0.008288149
##         cloves         leaves  vegetable_oil          cumin
##    0.008133230    0.007823393    0.007591015    0.007513555
##          green            cut         powder         chilli
##    0.007513555    0.007358637    0.006971340    0.006816421
##            red       turmeric          onion          piece
##    0.006041828    0.006041828    0.005809450    0.005654531
```

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 finely_chopped          flour
##    0.013306495    0.011349934    0.009623280    0.008989861
##         sliced         garlic         peeled            cut
##    0.008980477    0.008450281    0.008220374    0.008178146
##      freerange         leaves          white          juice
##    0.008060846    0.006639172    0.006343576    0.006301348
##          extra          large         caster   caster_sugar
##    0.005940064    0.005911913    0.005888453    0.005846225
##            red            egg          small          onion
##    0.005832149    0.005316029    0.005273801    0.005264417
```

Naive Bayes example prediction

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

```
##                p(w|not curry) p(w|curry)
## seeds                  0.005      0.017
## finely                 0.013      0.012
## coriander              0.003      0.011
## peeled                 0.008      0.010
## garlic                 0.008      0.009
## finely_chopped         0.010      0.008
## ginger                 0.003      0.008
## cloves                 0.005      0.008
## leaves                 0.007      0.008
## cumin                  0.001      0.008
## chilli                 0.003      0.007
## onion                  0.005      0.006
```

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                  0.006      0.005
## sugar                  0.011      0.003
## paste                  0.001      0.003
## flour                  0.009      0.003
## plain                  0.005      0.003
## lemon                  0.005      0.002
## freerange              0.008      0.002
## plain_flour            0.005      0.002
## eggs                   0.005      0.001
## unsalted               0.003      0.001
## freerange_eggs         0.004      0.001
## zest                   0.003      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] "Aromatic beef curry"
##  [2] "Aromatic blackeye bean curry"
##  [3] "Beef Madras"
##  [4] "Beef rendang with lemongrass and ginger, Thai herb salad and sticky rice"
##  [5] "Beef with onion and green pepper"
##  [6] "Banana leaf-baked sea bass"
##  [7] "Bunny chow"
##  [8] "Butterflied Madagascan prawns with a Goan curry sauce and a spinach salad"
##  [9] "Butternut squash and prawn curry with noodles"
## [10] "Chicken curry with ginger and red pepper served with pilau rice "
```

Naive Bayes Application

```
##           True
## Predicted FALSE TRUE
##     FALSE  1674   12
##     TRUE    150   41
##                   Naive Bayes
## accuracy           0.91369206
## misclassification  0.08630794
## sensitivity        0.77358491
## specificity        0.91776316
```

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

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 2324
```

- 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:

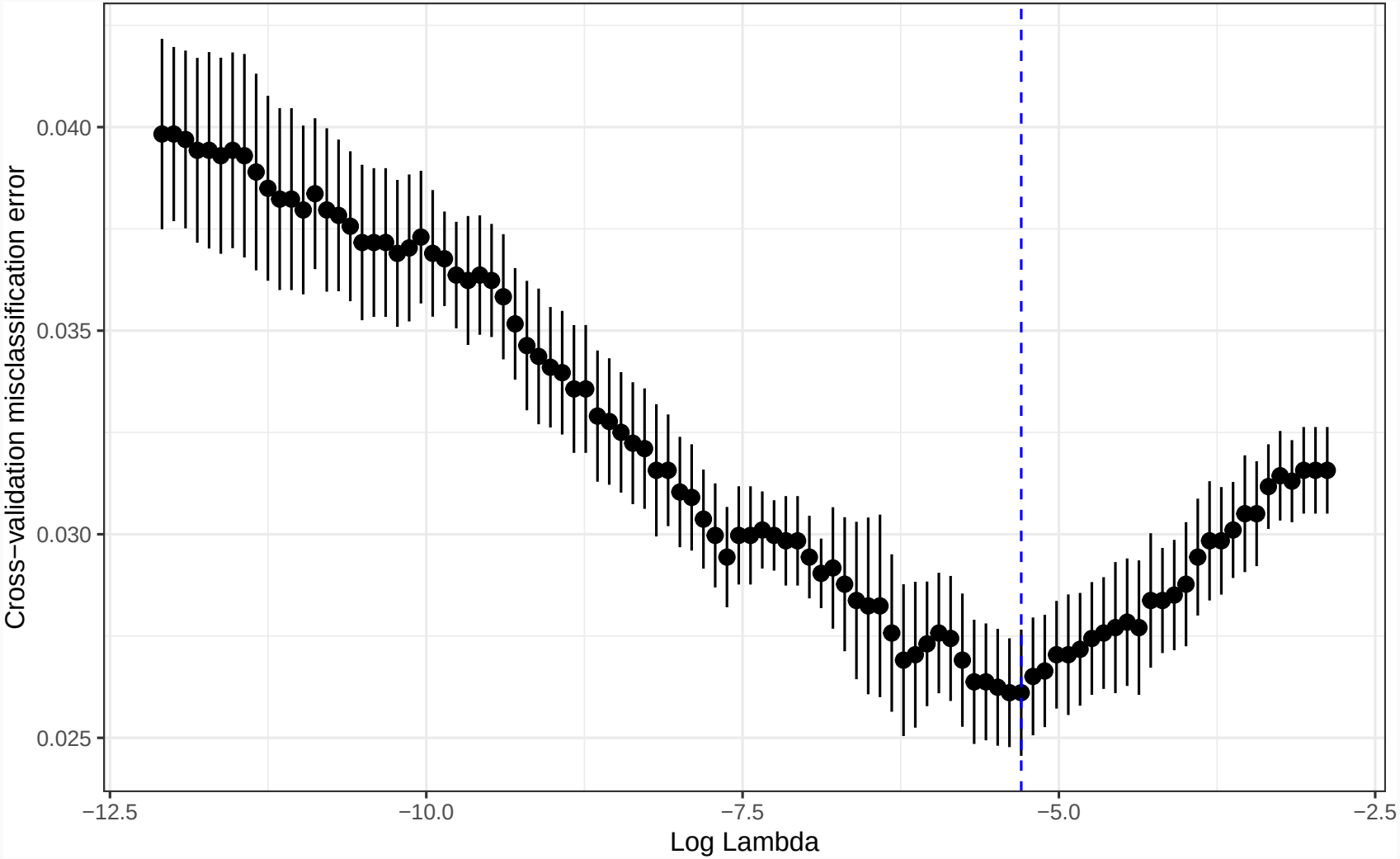
$$log\left(\frac{p(Y=1)}{1-p(Y=1)}\right) = \beta_0 + \sum_{j=1}^{J} \beta_j X_{i,j}$$

- 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

```
##         feature coef
## 2          beef    0
## 3         boned    0
## 4        rolled    0
## 5          pint    0
## 6           red    0
## 7          wine    0
## 8       vinegar    0
## 9         sugar    0
## 10    allspice    0
## 11         bay    0
## 12      leaves    0
## 13       thyme    0
## 14 peppercorns    0
## 15     crushed    0
## 16     english    0
## 17       dijon    0
```

Lasso regression example

In the cross-validation preferred model, almost all the coefficients are set to zero in the Lasso:

```
table(as.matrix(coef(cv_glm_out, s = cv_glm_out$lambda.1se))[,1] == 0)
```

```
##
## FALSE  TRUE
##    32  2293
```

Question: Which features have non-zero coefficients?

Lasso regression example

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

Lasso regression example

```
##           True
## Predicted FALSE TRUE
##     FALSE  1735   13
##     TRUE     89   40
##                         Lasso
## accuracy          0.94565796
## misclassification 0.05434204
## sensitivity       0.75471698
## specificity       0.95120614
```

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       0.914 0.946
## misclassification       0.040       0.086 0.054
## sensitivity             0.328       0.774 0.755
## specificity             0.980       0.918 0.951
```

Was #TheStew really #TheCurry?

Break

(Are you hungry yet?)

Supervised Scaling of Texts

From Classification to Scaling

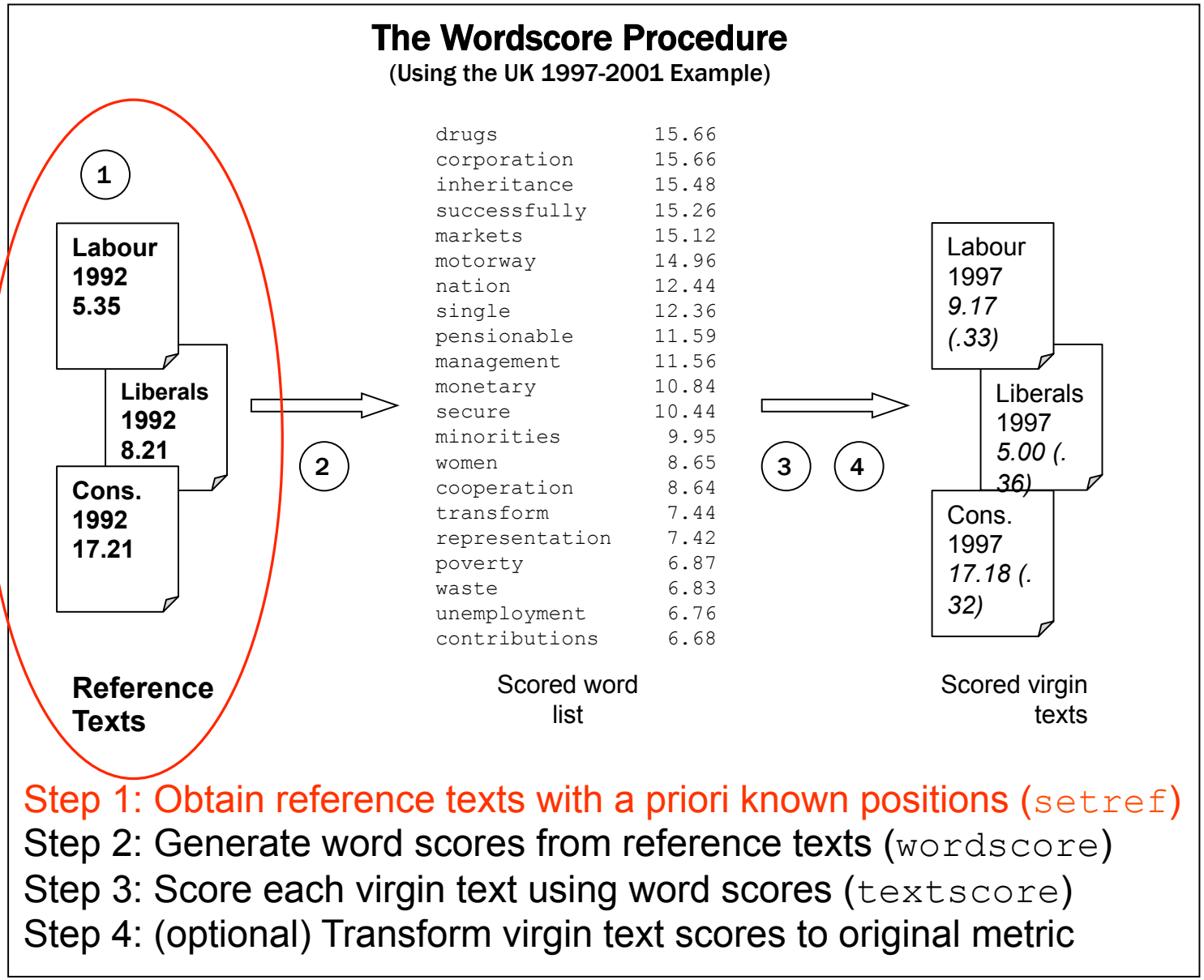
- Machine learning focuses on identifying classes (classification), while social science is typically interested in locating things on latent traits (scaling), e.g.:
    - Policy positions on economic vs social dimension
    - Inter- and intra-party differences
    - Soft news vs hard news
    - Petitioner vs respondent in legal briefs
    - ...and any other continuous scale
- But the two methods overlap! The class predictions for a collection of words from NB can be adapted to scaling

Supervised scaling methods

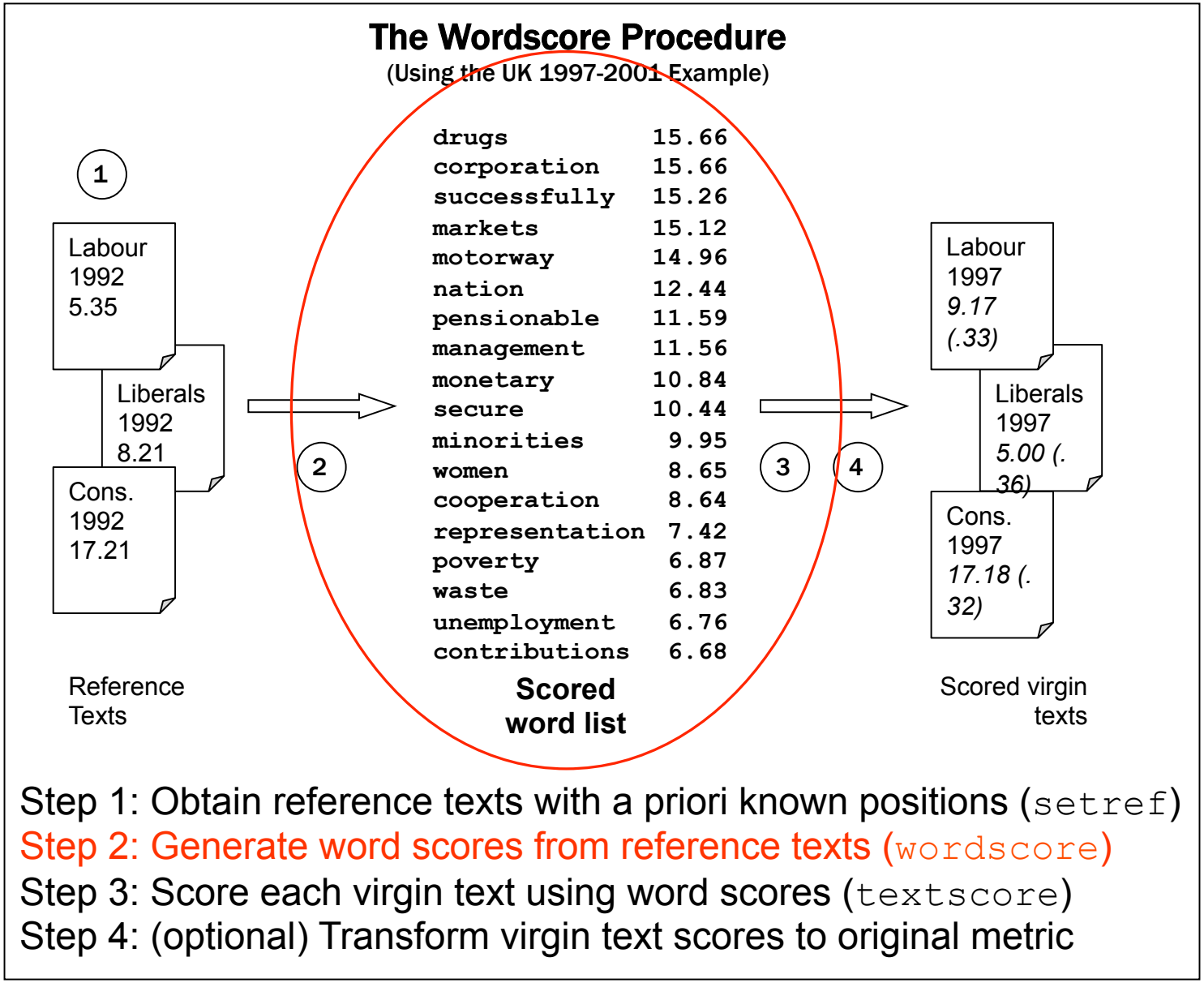
Wordscores method (Laver, Benoit & Garry, 2003):

- 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)
- These are analogous to a "training set" and a "test set" in classification
- Basic procedure:
    1. Analyze reference texts to obtain word scores
    2. Use word scores to score virgin texts

Wordscores Procedure



Wordscores Procedure



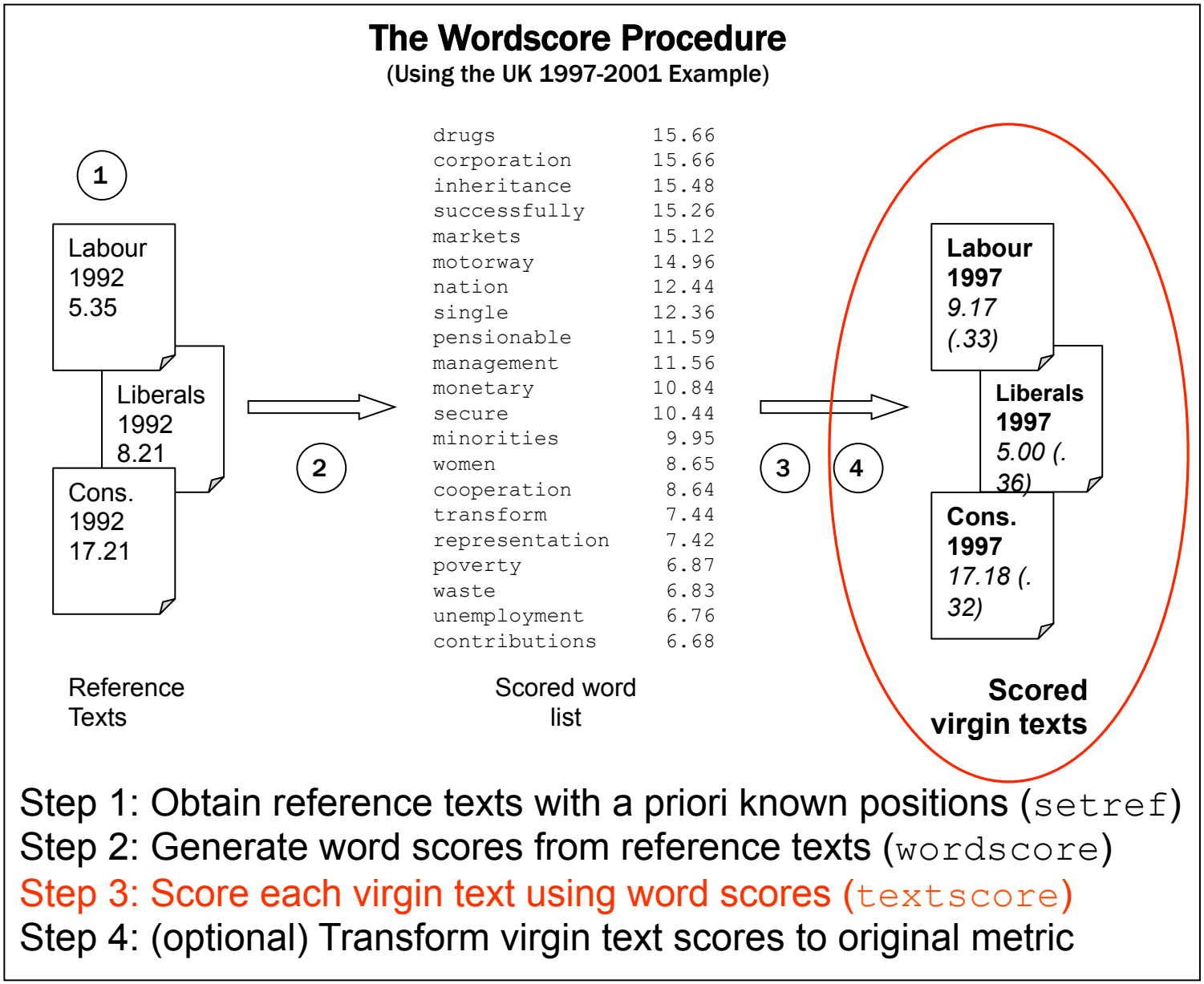
Step 1: Obtain reference texts with a priori known positions (`setref`)

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

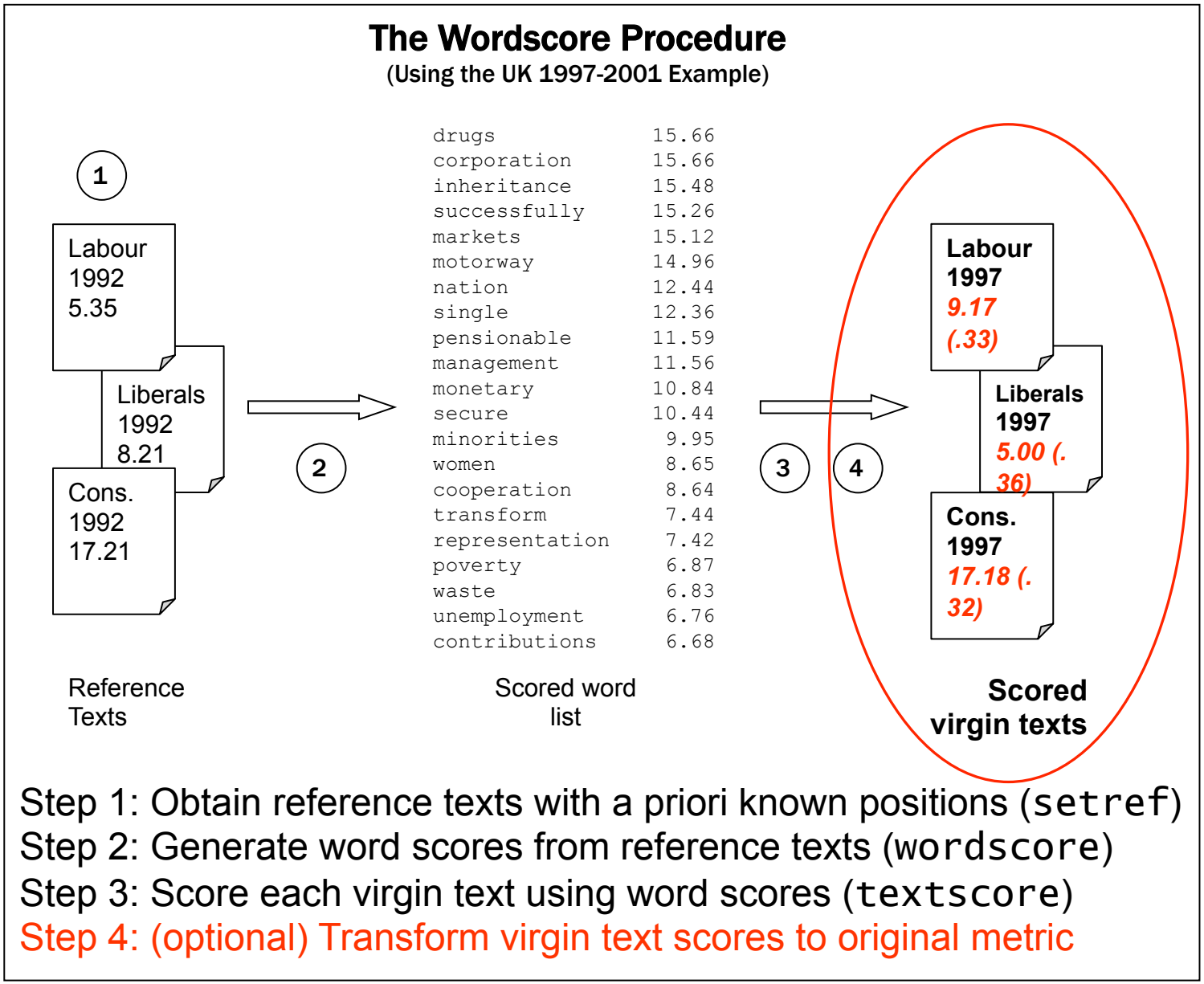
Step 3: Score each virgin text using word scores (`textscore`)

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

Wordscores Procedure



Wordscores Procedure



Wordscores mathematically: Reference texts

- Start with a set of I *reference* texts, represented by an $I \times J$ document-feature matrix C_{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 C_{ij} into a *relative* document-feature matrix (within document), by dividing C_{ij} by its word total marginals:

$$F_{ij} = \frac{C_{ij}}{C_{i\cdot}} \tag{1}$$

Wordscores mathematically: Word scores

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

$$P_{ij} = \frac{F_{ij}}{\sum_{i=1}^{I} F_{ij}} \tag{2}$$

- 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 mathematically: Word scores (example)

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

$$P_{A\ \text{"choice"}} = \frac{.010}{(.010 + .030)} = 0.25 \tag{3}$$

$$P_{B\ \text{"choice"}} = \frac{.030}{(.010 + .030)} = 0.75 \tag{4}$$

Wordscores mathematically: Word scores

- 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}:

$$S_j = \sum_{i=1}^{I} a_i P_{ij} \tag{5}$$

- 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

Wordscores mathematically: Word scores

- Continuing with our example:
  - 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 "choice" is then $0.25(-1.0) + 0.75(1.0) = -0.25 + 0.75 = +0.50$

Wordscores mathematically: Scoring "virgin" texts

- Here the objective 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
- So the score v_k of a virgin document k consisting of the j word types is:

$$v_k = \sum_j (F_{kj} \cdot s_j) \tag{6}$$

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

- 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

Wordscores mathematically: Rescaling raw text scores

- Because of overlapping or non-discriminating words, the raw text scores will be dragged to the interior of the reference scores
- Some procedures can be applied to rescale them, either to a unit normal metric or to a more "natural" metric

Pros and Cons of the Wordscores approach

Pros:

- Fully automated technique with minimal human intervention or judgment calls – only with regard to reference text selection
- Language-blind: all we need to know are reference scores
- Estimates unknown positions on a priori scales – hence no inductive scaling with a posteriori interpretation of unknown policy space

Cons:

- Very dependent on correct identification of:
  - appropriate reference texts
  - appropriate reference scores

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 estimates of Document-Class Probabilities

- It turns out that the wordscores approach is *very* similar to a Naive Bayes model
- In particular, the wordscores are a linear combination of word-level class posterior probabilities from a NB model
- In practice, the document probabilities from NB correlate very highly with the wordscores

Applying Wordscores to the BBC food corpus

- Let's use wordscores to scale the BBC recipes on a main meal vs dessert/pudding dimension
- Reference texts:
  - Score $\alpha_i = -1$ if recipe title includes "beef", "chicken", "lamb", "duck", "pork", or "ham"
  - Score $\alpha_i = 1$ if recipe title includes "sugar", "cake", "biscuit", "ice cream", "icing", "toffee", or "jam"
  - Note: this is possibly not a good way to select reference texts!

```
table(recipes$main_or_pudding, useNA = "always")
```

```
##
##   -1    1 <NA>
## 1401 2272 5711
```

N.b. **NA** here are our "virgin" texts

Applying Wordscores to the BBC food corpus

```
wordscores_mod <- textmodel_wordscores(recipe_dfm, recipe_dfm$main_or_pudding)

wordscores <- predict(wordscores_mod)
```

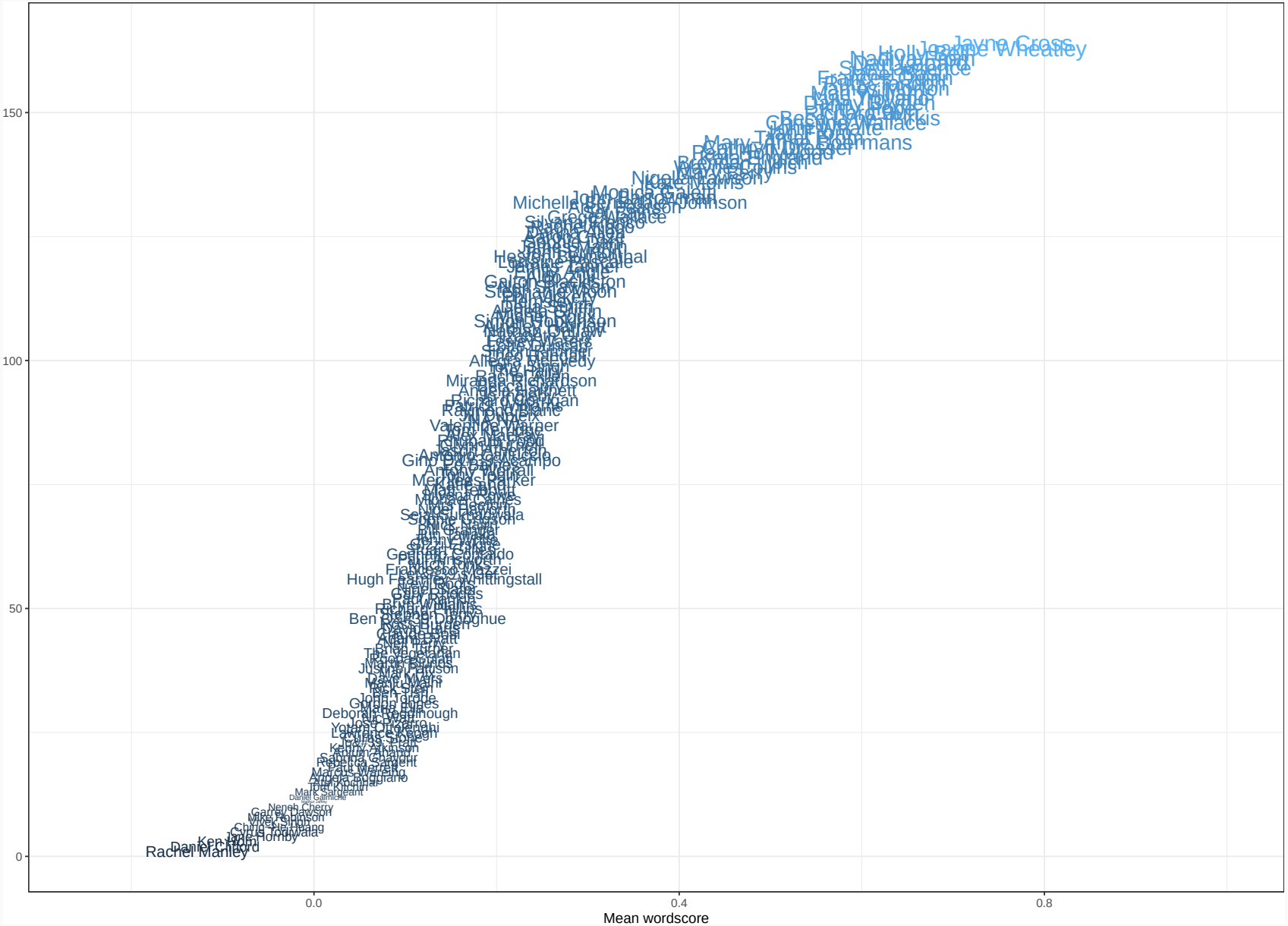
Top “main meal” recipes

```
##                                                                                recipe_name main_or_pudding
## 1                                                                          Lamb with herb crust              NA
## 2                                                                 Pork tenderloin with creamed leeks              -1
## 3     Tea-smoked lamb chops, with miso grilled aubergine (nasu dengaku) and pickled daikon              -1
## 4                                                                   Honey-roasted spicy leg of lamb              -1
## 5                                                                  Pan-fried pork with pepper sauce              -1
## 6                                                 Balsamic figs with roasted duck breast and crispy leeks              -1
## 7                                               Chargrilled duck breast with broccoli and blackberry sauce              -1
## 8     Pan-roasted goose breast with roast swede and chestnuts and leek and cranberry sauce              NA
## 9                                                    Grilled lamb cutlets filled with parma ham and herbs              -1
## 10 Roasted poussin with roast potatoes, cashew-crusted tomatoes and tarragon vinegar cream              NA
```

Top “pudding” recipes

```
##                                                  recipe_name main_or_pudding
## 1                                                Key lime pie              NA
## 2                                             Strawberry fool               1
## 3                                  Strawberry and almond crumble             NA
## 4                    Honeycomb ice cream with pistachio shortbread             1
## 5                                 Espresso and ice cream (affogato)            1
## 6  Elderflower and lemon tart, with strawberry sorbet and meringues             NA
## 7                    Honeyed yoghurt with rosewater scented peaches             NA
## 8                Roe deer with blackcurrant sauce and cabbage slaw             NA
## 9                     Caramelised blueberries with lemon syllabub              NA
## 10                                      Sweet blueberry omelette              NA
```

Chefs, savoury and sweet



Chefs, savoury and sweet

Joanne Wheatley

British baker

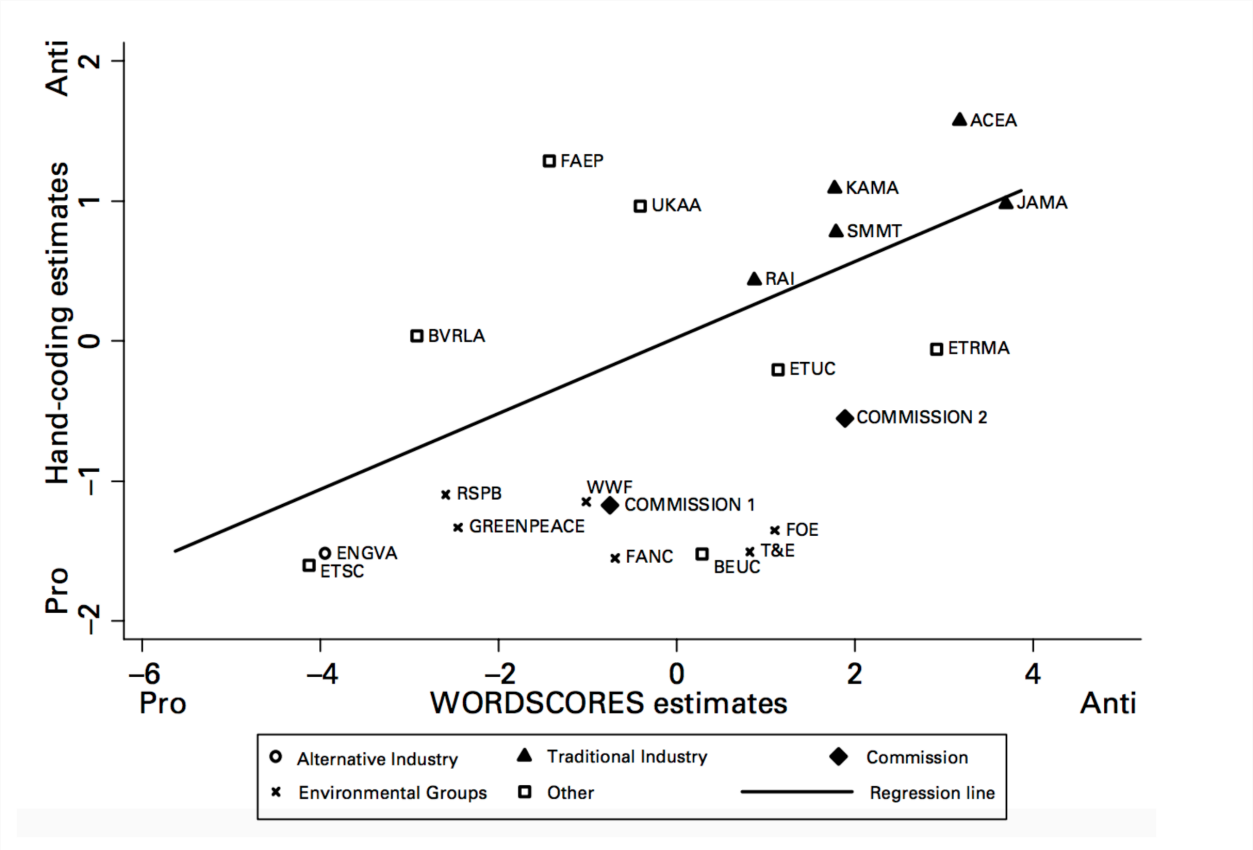
JO WHEATLEY has loved baking since she was a child. Last year she watched the BBC's The Great British Bake Off and decided to enter. She feels incredibly proud that not only did she make it onto the programme, but she also went on to win against some amazing bakers. ... Google Books

Mean wordscore = 0.75

Chefs, savoury and sweet

Mean wordscore = -0.09

Serious Social Science Application: Scaling environmental interest groups (Klüver 2009)



- Dataset: text of online consultation on EU environmental regulations
- Reference texts: most extreme pro- and anti-regulation groups

Unsupervised Scaling of Texts

Unsupervised methods scale distance between documents

- Text gets converted into a quantitative matrix of features words, typically
    - could be dictionary entries, or parts of speech
    - Documents are scaled based on *similarity* or *distance* in feature use
- Fundamental problem: distance on which scale?
- Ideally, something we care about, e.g. policy positions, ideology, preferences, sentiment
- But often other dimensions (language, rhetoric style, authorship) are more predictive
- First dimension in unsupervised scaling will capture main source of variation, whatever that is
- Unlike supervised models, validation comes after estimating the model

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 (think of PCA).

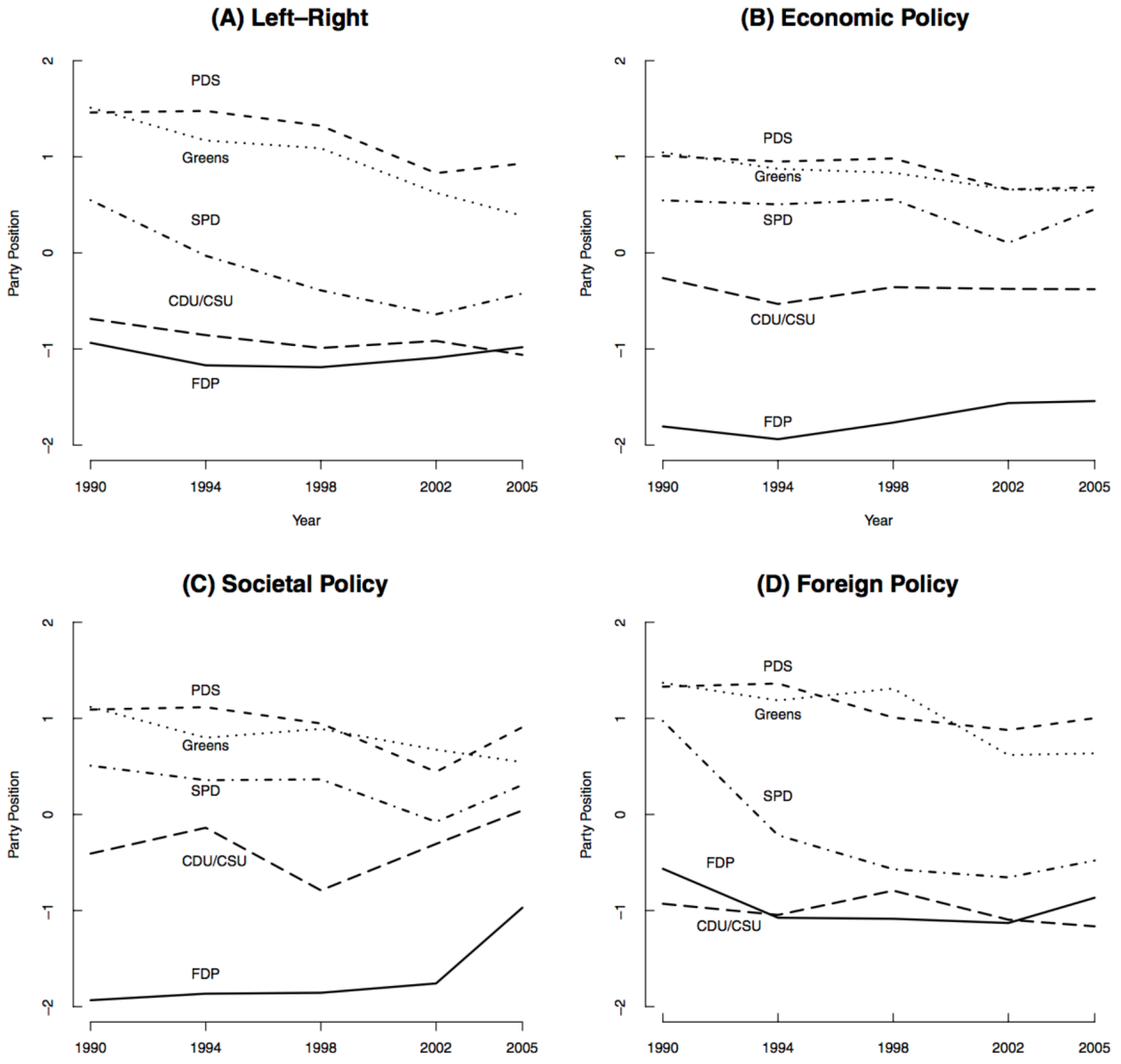
To identify the model in Wordfish we fix the relative position of two θ_is to specify the "direction" of the dimension

```
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.

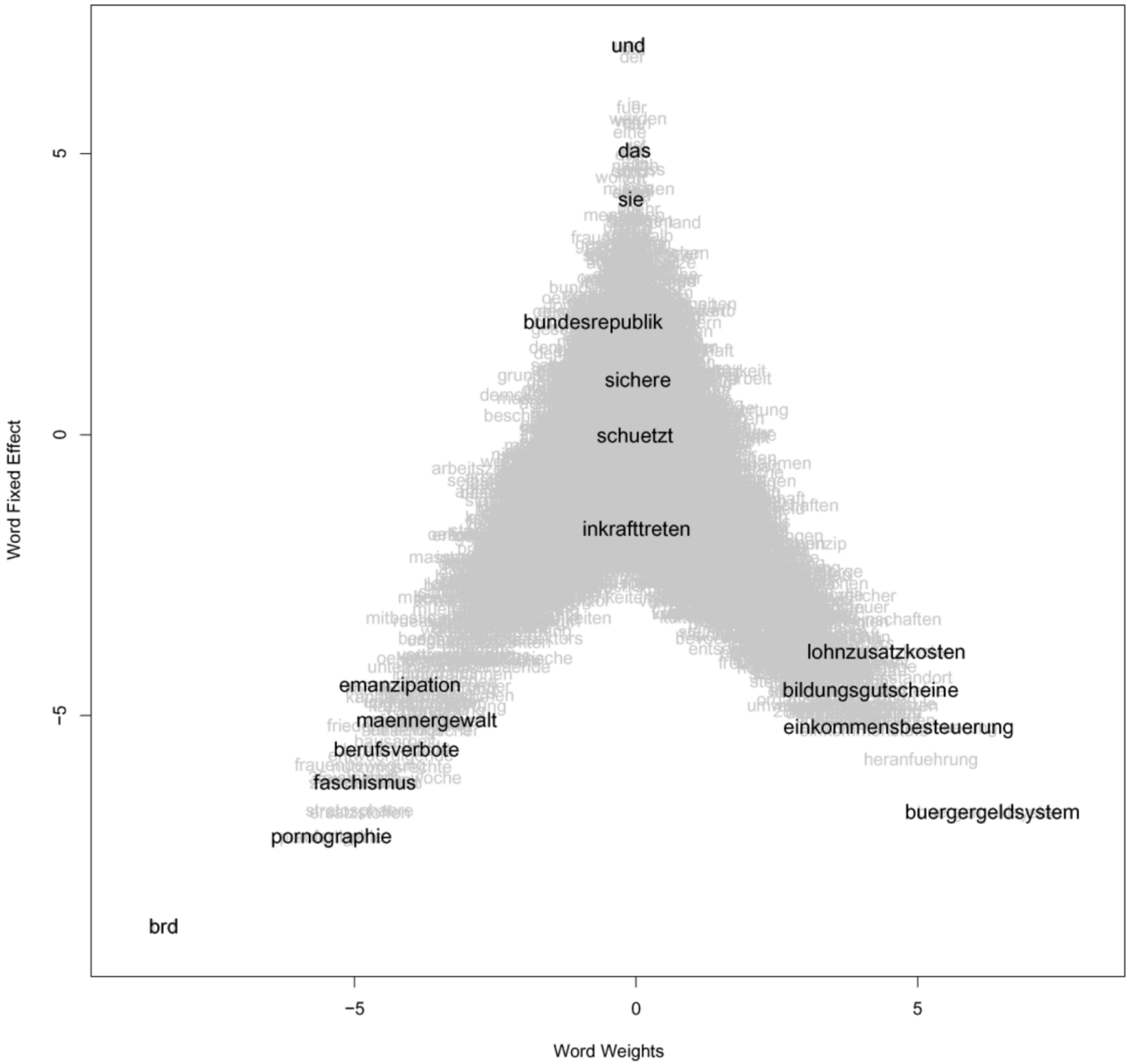
Application to German Political Manifestos

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