Multinomial NB

CS5154/6054

Yizong Cheng

10/20/2022

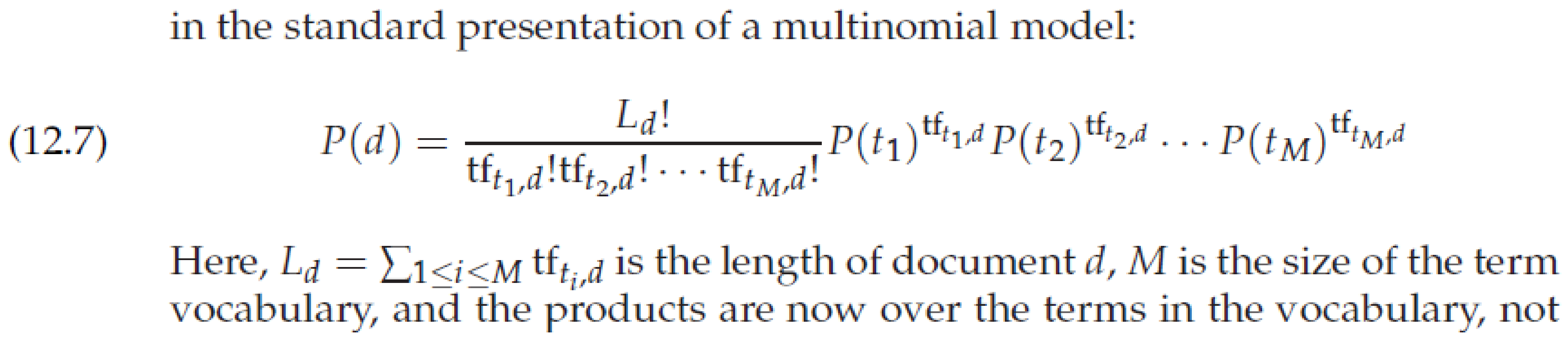
13 *Text classification and Naive Bayes*

13.2 Naive Bayes text classification

14 *Vector space classification*

14.4 Linear versus nonlinear classifiers

Multinomial Distribution on Bags of Words

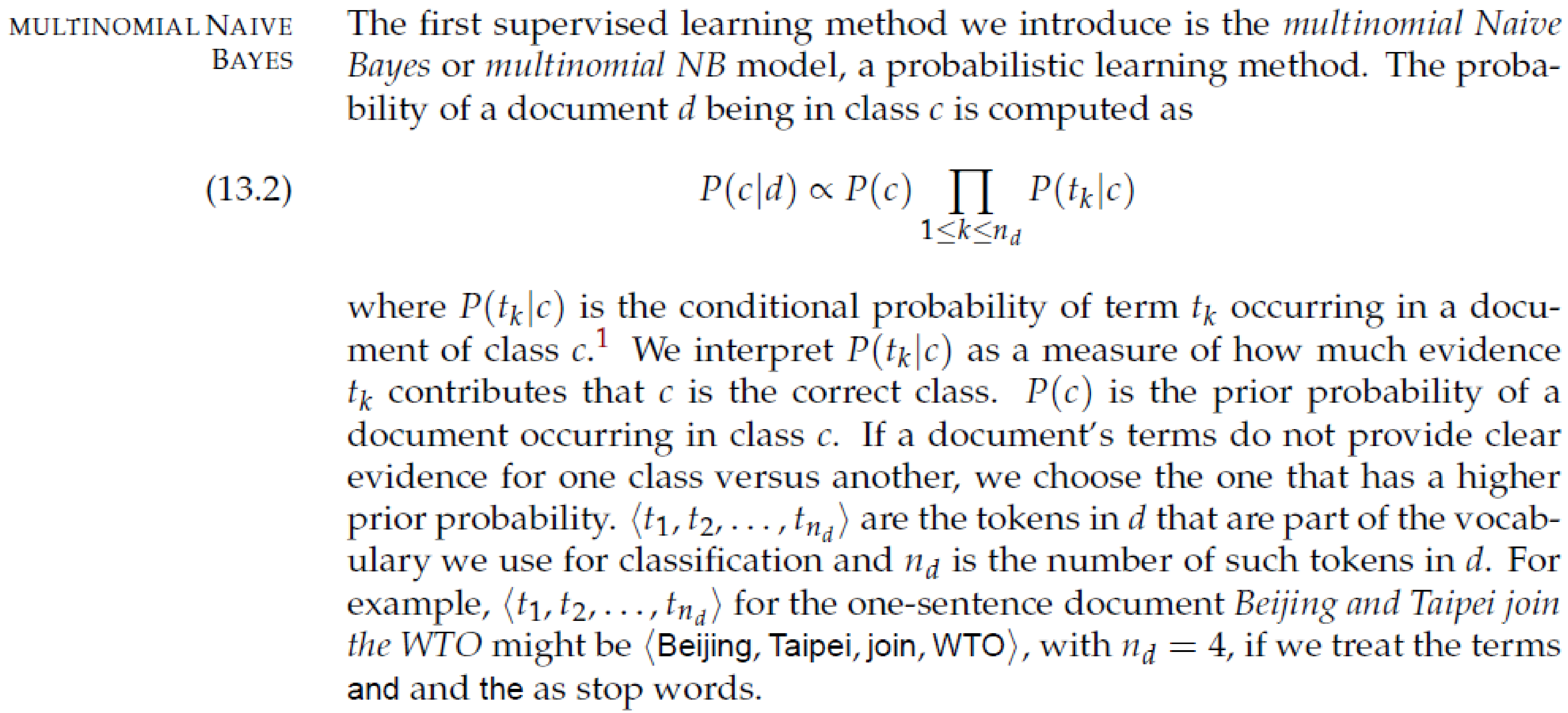


in the standard presentation of a multinomial model:

(12.7) $$P(d) = \frac{L_d!}{\mathrm{tf}_{t_1,d}!\mathrm{tf}_{t_2,d}! \cdots \mathrm{tf}_{t_M,d}!} P(t_1)^{\mathrm{tf}_{t_1,d}} P(t_2)^{\mathrm{tf}_{t_2,d}} \ldots P(t_M)^{\mathrm{tf}_{t_M,d}}$$

Here, $L_d = \sum_{1 \leq i \leq M} \mathrm{tf}_{t_i,d}$ is the length of document d, M is the size of the term vocabulary, and the products are now over the terms in the vocabulary, not

Multinomial Naïve Bayes (NB)



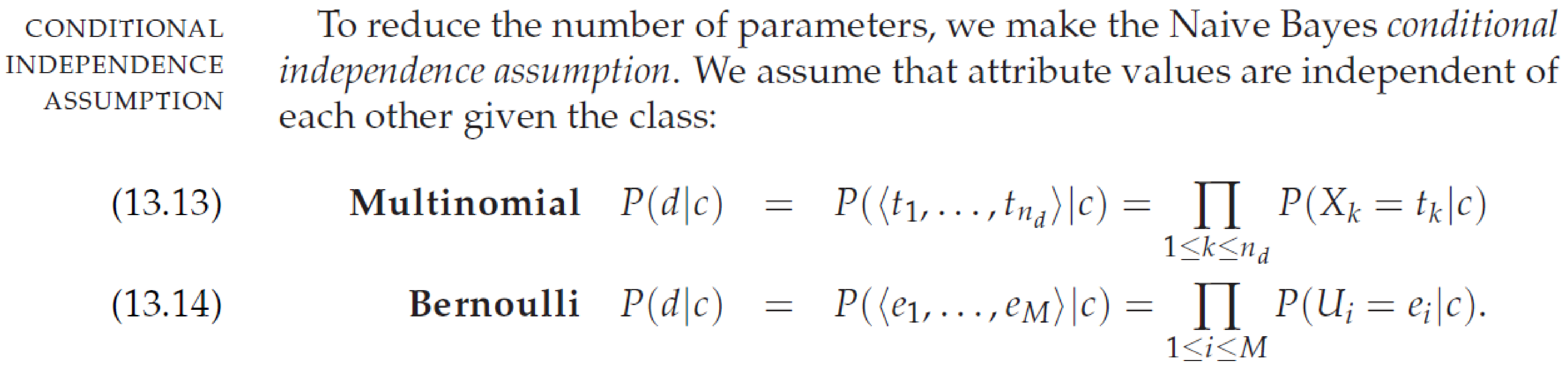
MULTINOMIAL NAIVE BAYES

The first supervised learning method we introduce is the *multinomial Naive Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document d being in class c is computed as

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \qquad (13.2)$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c.[1] We interpret $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class. $P(c)$ is the prior probability of a document occurring in class c. If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability. $\langle t_1, t_2, \ldots, t_{n_d} \rangle$ are the tokens in d that are part of the vocabulary we use for classification and n_d is the number of such tokens in d. For example, $\langle t_1, t_2, \ldots, t_{n_d} \rangle$ for the one-sentence document *Beijing and Taipei join the WTO* might be $\langle$Beijing, Taipei, join, WTO$\rangle$, with $n_d = 4$, if we treat the terms and and the as stop words.

Random Variables X and U



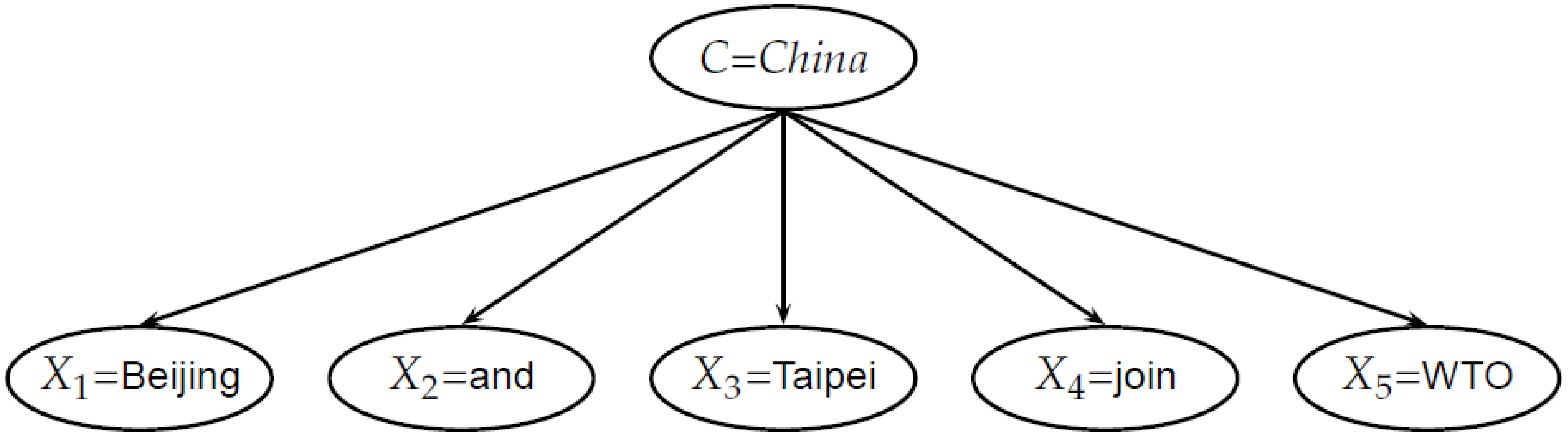
CONDITIONAL INDEPENDENCE ASSUMPTION

To reduce the number of parameters, we make the Naive Bayes *conditional independence assumption*. We assume that attribute values are independent of each other given the class:

$$\textbf{Multinomial} \quad P(d|c) \quad = \quad P(\langle t_1, \ldots, t_{n_d} \rangle | c) = \prod_{1 \le k \le n_d} P(X_k = t_k | c) \tag{13.13}$$

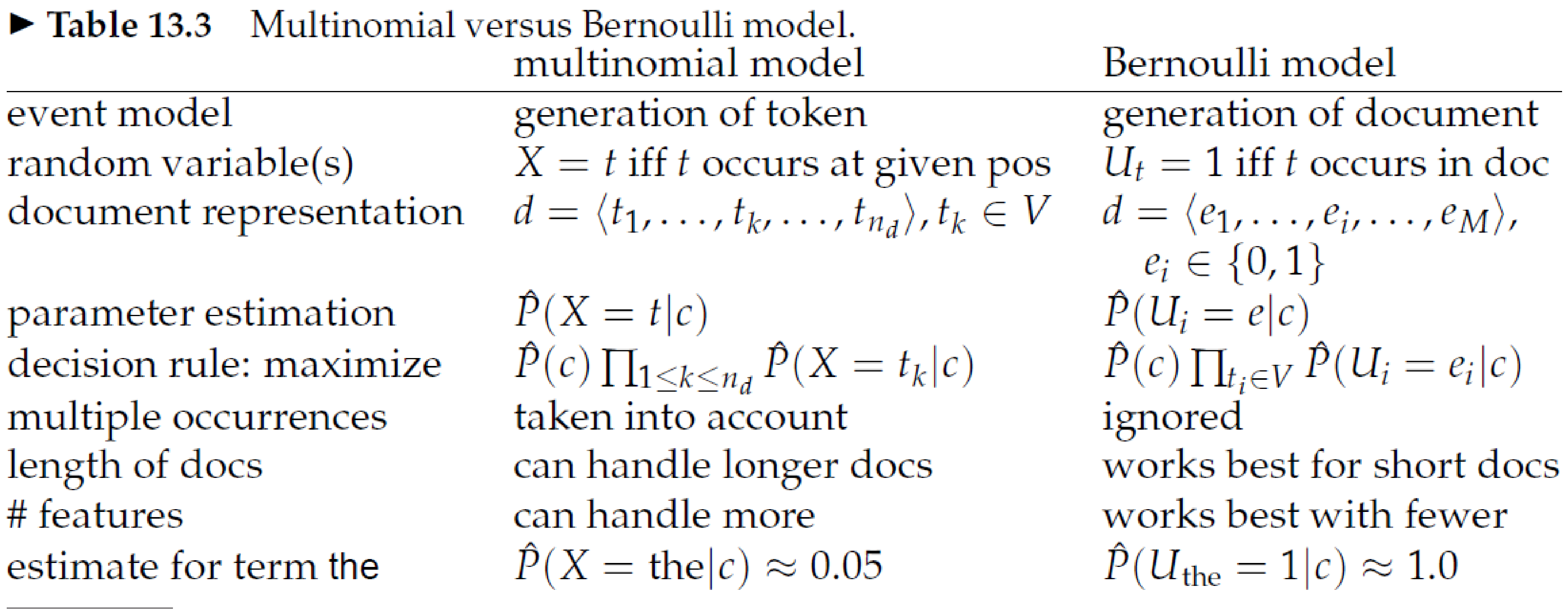
$$\textbf{Bernoulli} \quad P(d|c) \quad = \quad P(\langle e_1, \ldots, e_M \rangle | c) = \prod_{1 \le i \le M} P(U_i = e_i | c). \tag{13.14}$$

Random Variable X as a Vector of Terms



► **Figure 13.4** The multinomial NB model.

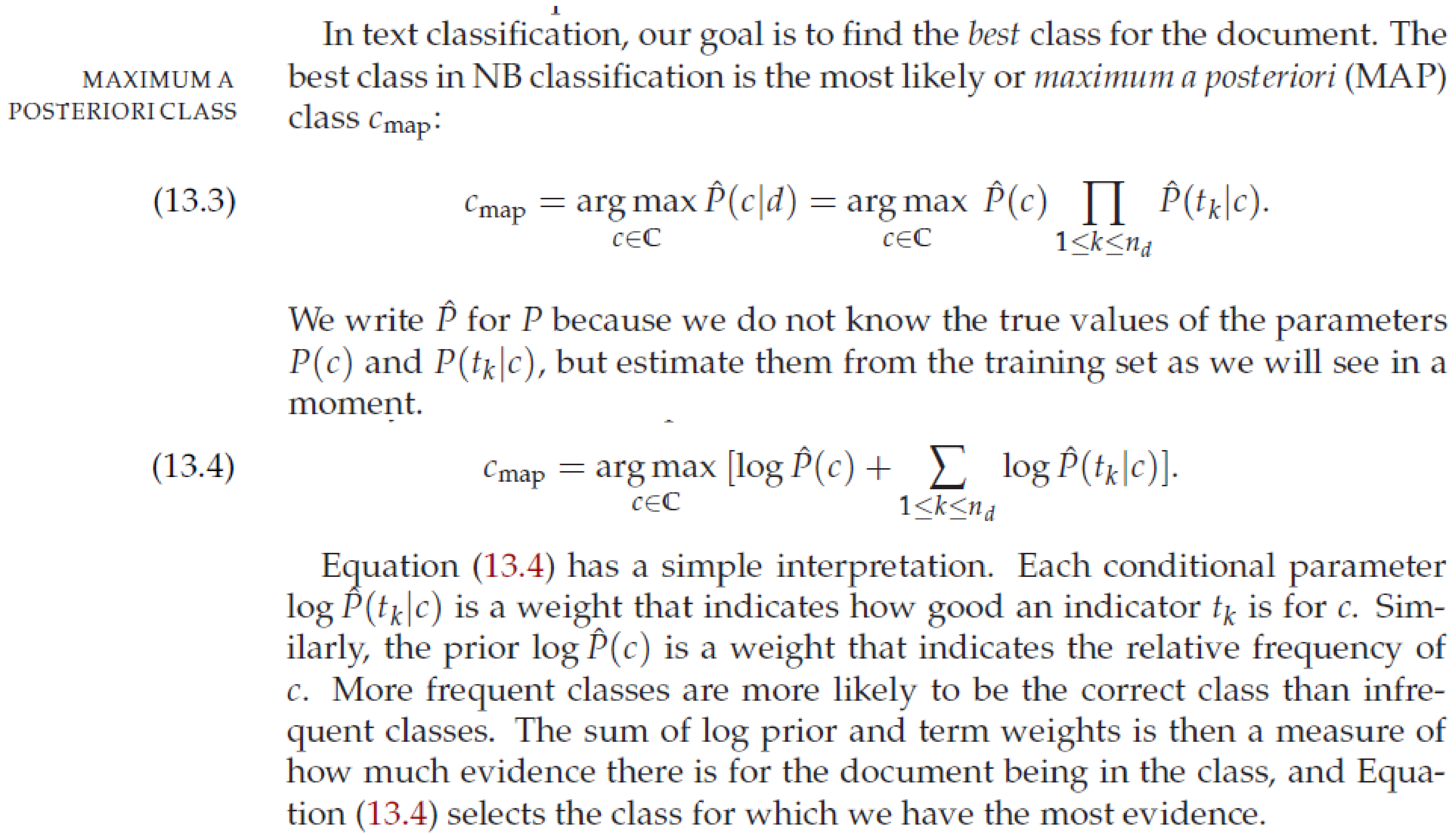
Comparing MNB with BernoulliNB



▶ **Table 13.3** Multinomial versus Bernoulli model.

	multinomial model	Bernoulli model
event model	generation of token	generation of document
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
document representation	$d = \langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \ldots, e_i, \ldots, e_M \rangle$, $e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t \mid c)$	$\hat{P}(U_i = e \mid c)$
decision rule: maximize	$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k \mid c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i \mid c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for term the	$\hat{P}(X = \text{the} \mid c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1 \mid c) \approx 1.0$

MAP Class



MAXIMUM A POSTERIORI CLASS

In text classification, our goal is to find the *best* class for the document. The best class in NB classification is the most likely or *maximum a posteriori* (MAP) class c_{map}:

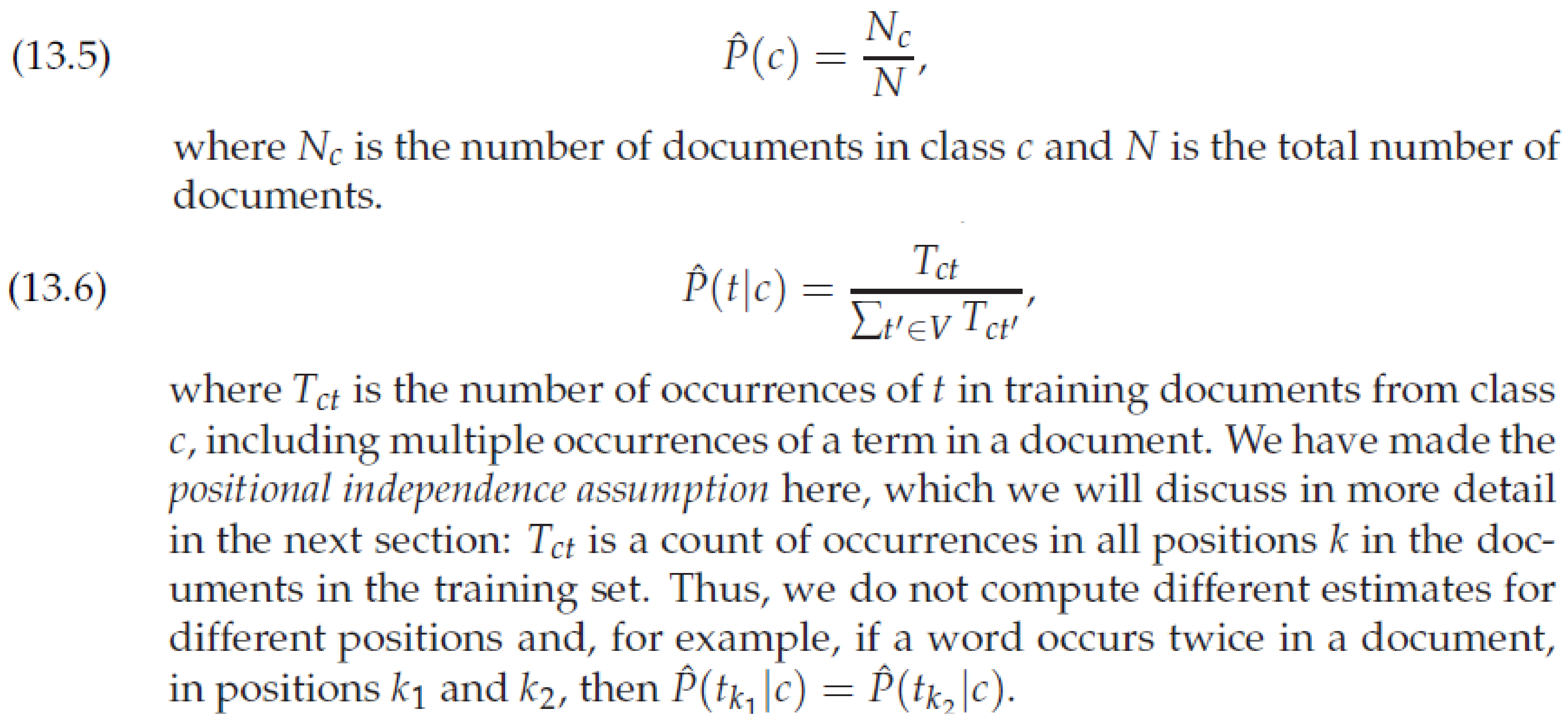
$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c). \tag{13.3}$$

We write $\hat{P}$ for P because we do not know the true values of the parameters $P(c)$ and $P(t_k|c)$, but estimate them from the training set as we will see in a moment.

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]. \tag{13.4}$$

Equation (13.4) has a simple interpretation. Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator t_k is for c. Similarly, the prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c. More frequent classes are more likely to be the correct class than infrequent classes. The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class, and Equation (13.4) selects the class for which we have the most evidence.

MLE Estimates of Parameters



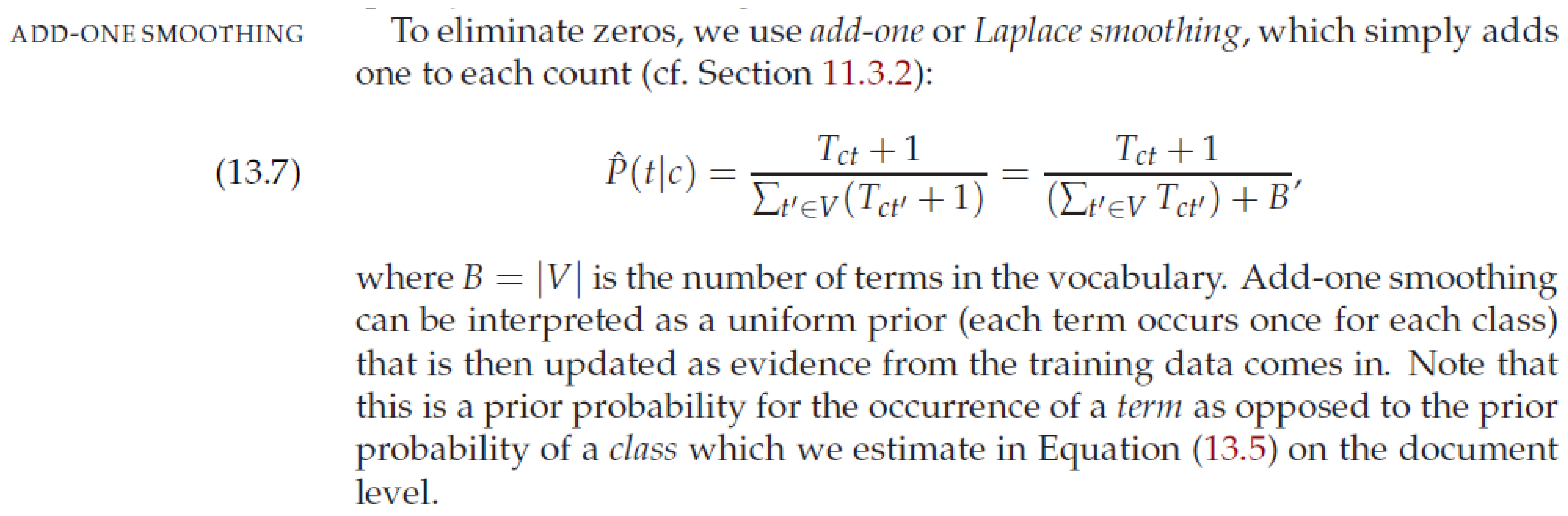
$$\hat{P}(c) = \frac{N_c}{N}, \tag{13.5}$$

where N_c is the number of documents in class c and N is the total number of documents.

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}, \tag{13.6}$$

where T_{ct} is the number of occurrences of t in training documents from class c, including multiple occurrences of a term in a document. We have made the *positional independence assumption* here, which we will discuss in more detail in the next section: T_{ct} is a count of occurrences in all positions k in the documents in the training set. Thus, we do not compute different estimates for different positions and, for example, if a word occurs twice in a document, in positions k_1 and k_2, then $\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c)$.

Laplace Smoothing to Avoid Zeros



ADD-ONE SMOOTHING

To eliminate zeros, we use *add-one* or *Laplace smoothing*, which simply adds one to each count (cf. Section 11.3.2):

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'}, \quad (13.7)$$

where $B = |V|$ is the number of terms in the vocabulary. Add-one smoothing can be interpreted as a uniform prior (each term occurs once for each class) that is then updated as evidence from the training data comes in. Note that this is a prior probability for the occurrence of a *term* as opposed to the prior probability of a *class* which we estimate in Equation (13.5) on the document level.

```
TRAINMULTINOMIALNB($\mathbb{C}, \mathbb{D}$)
 1  $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
 2  $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
 3  for each $c \in \mathbb{C}$
 4  do $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}, c$)
 5      $prior[c] \leftarrow N_c / N$
 6      $text_c \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS($\mathbb{D}, c$)
 7      for each $t \in V$
 8      do $T_{ct} \leftarrow$ COUNTTOKENSOFTERM($text_c, t$)
 9      for each $t \in V$
10      do $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11  return $V, prior, condprob$
```

```
APPLYMULTINOMIALNB($\mathbb{C}, V, prior, condprob, d$)
1  $W \leftarrow$ EXTRACTTOKENSFROMDOC($V, d$)
2  for each $c \in \mathbb{C}$
3  do $score[c] \leftarrow \log prior[c]$
4      for each $t \in W$
5      do $score[c]$ += $\log condprob[t][c]$
6  return $\arg\max_{c \in \mathbb{C}} score[c]$
```

► **Figure 13.2** Naive Bayes algorithm (multinomial model): Training and testing.

▶ **Table 13.1** Data for parameter estimation examples.

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

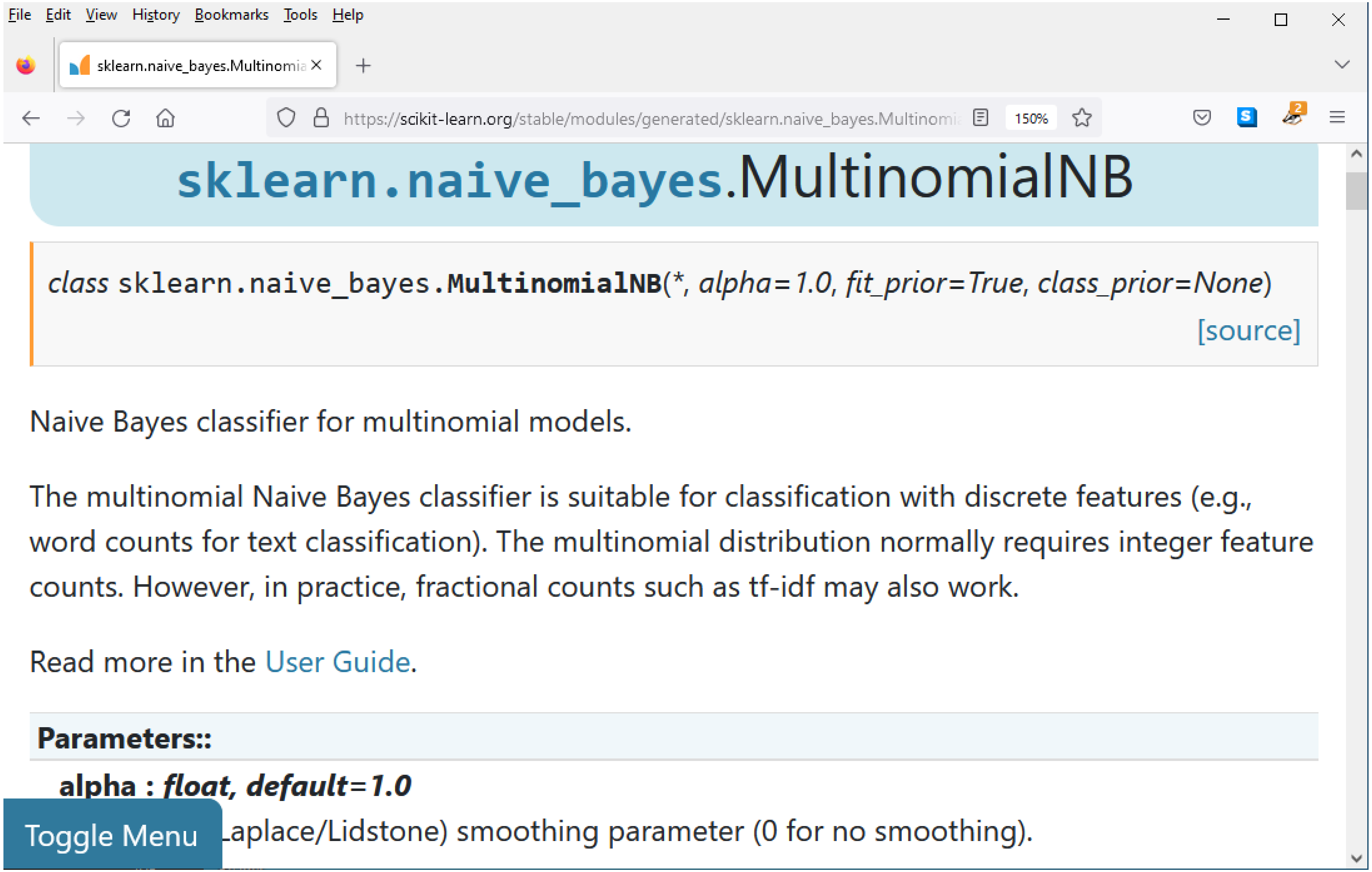
Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}
\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms.

We then get:

$$\begin{aligned}
\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003. \\
\hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.
\end{aligned}$$


File Edit View History Bookmarks Tools Help
sklearn.naive_bayes.Multinomia
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.Multinomi
150%
Toggle Menu

sklearn.naive_bayes.MultinomialNB

class `sklearn.naive_bayes.`**`MultinomialNB`**(**, alpha=1.0, fit_prior=True, class_prior=None*)

[source]

Naive Bayes classifier for multinomial models.

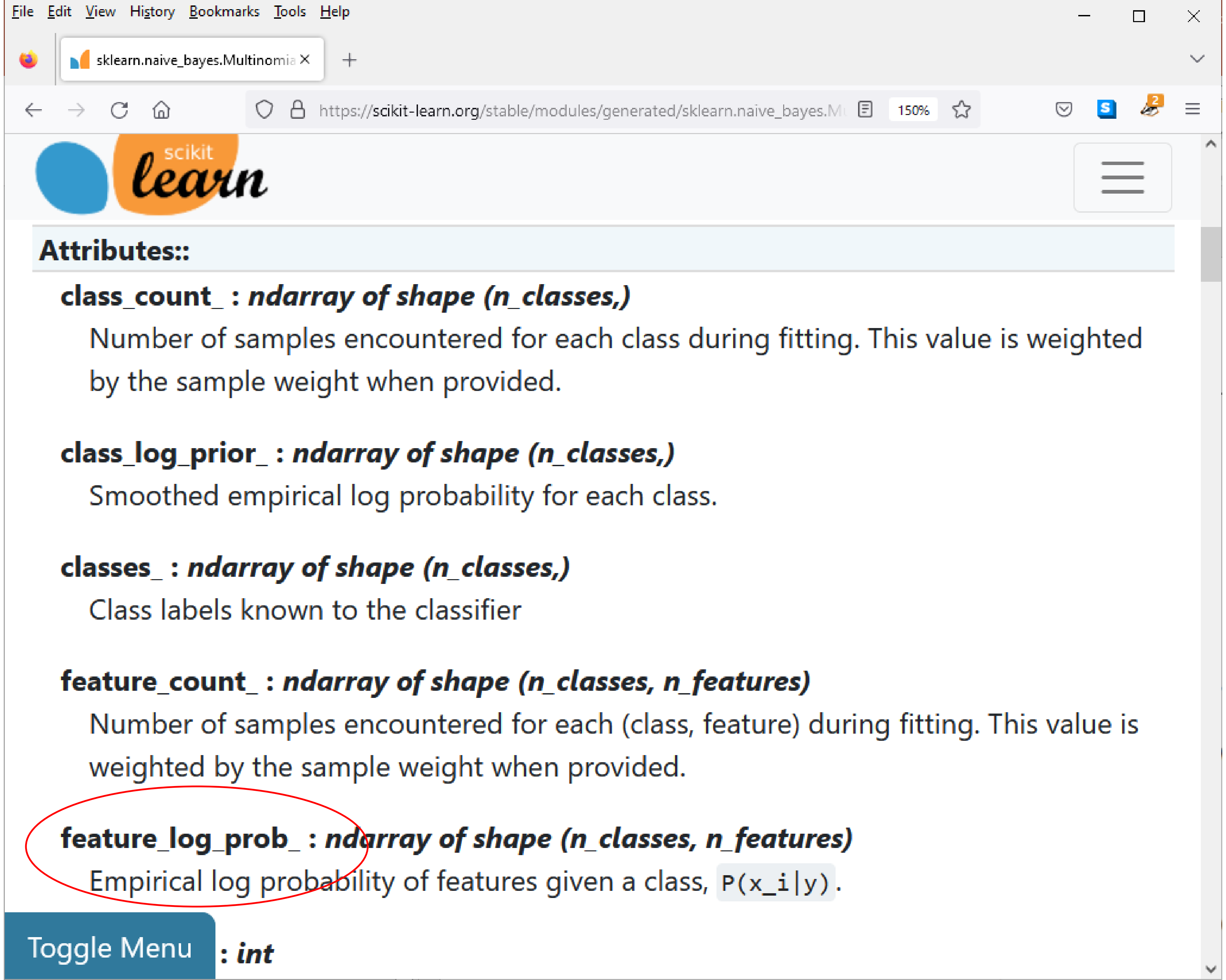
The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Read more in the User Guide.

Parameters::

alpha : *float, default=1.0*

aplace/Lidstone) smoothing parameter (0 for no smoothing).


File Edit View History Bookmarks Tools Help
sklearn.naive_bayes.Multinomia
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.M
150%
scikit learn

Attributes::

class_count_ : ***ndarray of shape (n_classes,)***
Number of samples encountered for each class during fitting. This value is weighted by the sample weight when provided.

class_log_prior_ : ***ndarray of shape (n_classes,)***
Smoothed empirical log probability for each class.

classes_ : ***ndarray of shape (n_classes,)***
Class labels known to the classifier

feature_count_ : ***ndarray of shape (n_classes, n_features)***
Number of samples encountered for each (class, feature) during fitting. This value is weighted by the sample weight when provided.

feature_log_prob_ : ***ndarray of shape (n_classes, n_features)***
Empirical log probability of features given a class, `P(x_i|y)`.

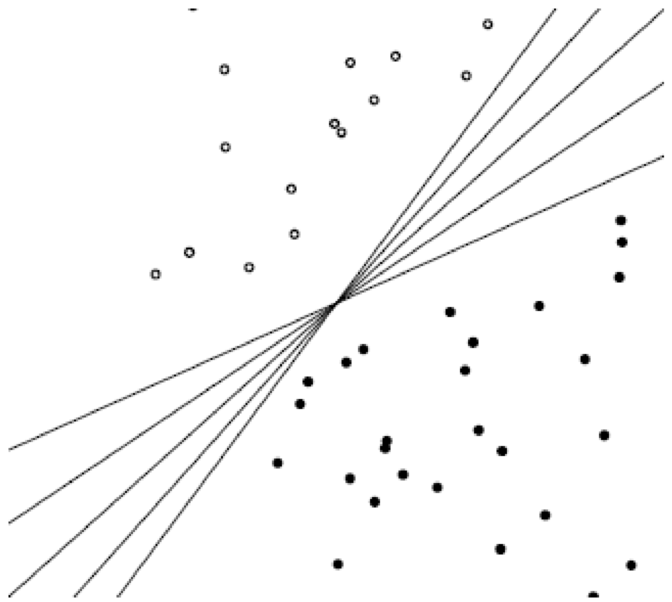
: ***int***

14.4 Linear versus nonlinear classifiers

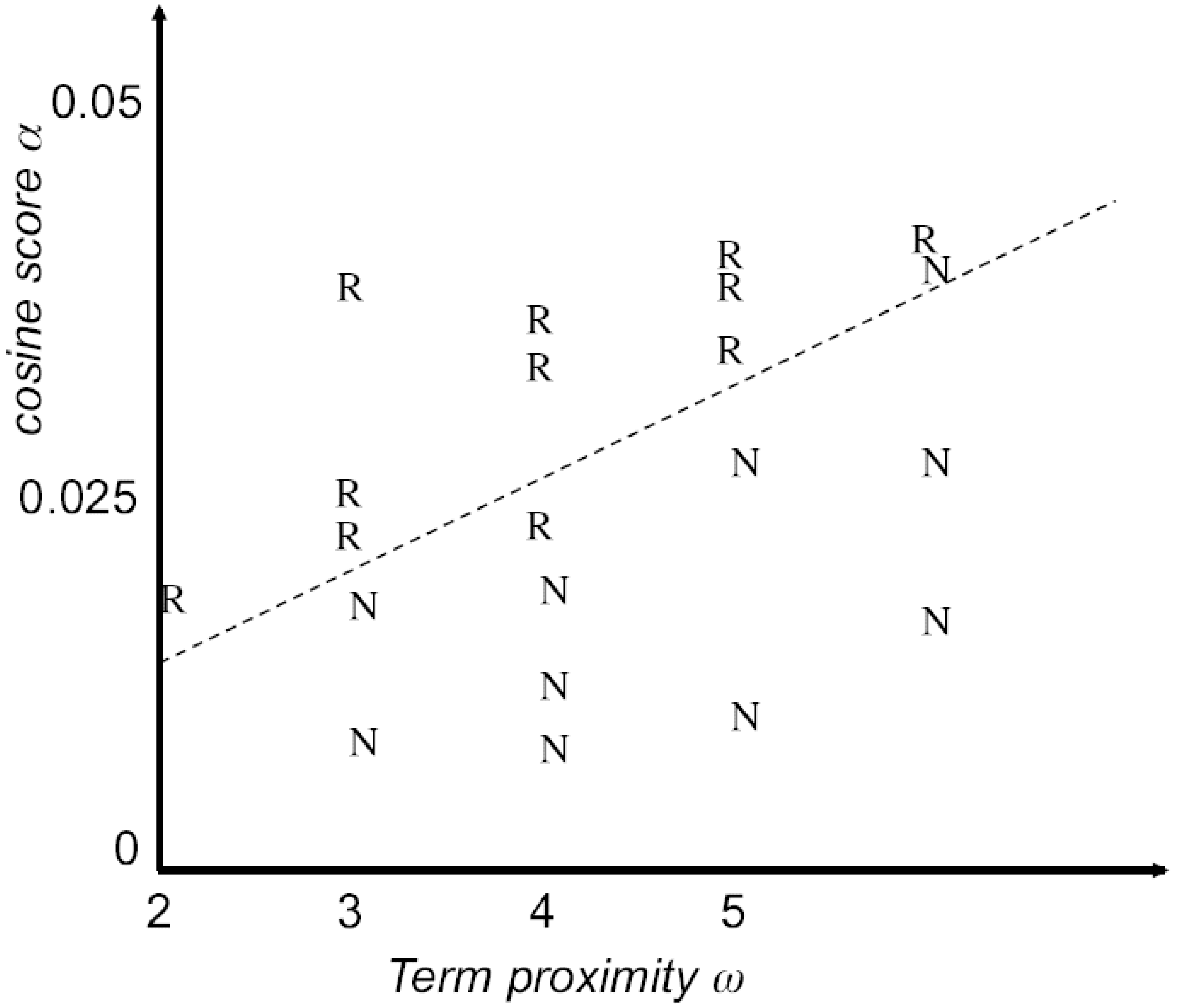
In this section, we show that the two learning methods Naive Bayes and Rocchio are instances of linear classifiers, the perhaps most important group of text classifiers, and contrast them with nonlinear classifiers. To simplify the discussion, we will only consider two-class classifiers in this section and define a *linear classifier* as a two-class classifier that decides class membership by comparing a linear combination of the features to a threshold.

LINEAR CLASSIFIER

In two dimensions, a linear classifier is a line. Five examples are shown in Figure 14.8. These lines have the functional form $w_1x_1 + w_2x_2 = b$. The classification rule of a linear classifier is to assign a document to c if $w_1x_1 + w_2x_2 > b$ and to $\overline{c}$ if $w_1x_1 + w_2x_2 \leq b$. Here, $(x_1, x_2)^T$ is the two-dimensional vector representation of the document and $(w_1, w_2)^T$ is the parameter vector that defines (together with b) the decision boundary. An alternative geometric interpretation of a linear classifier is provided in Figure 15.7 (page 343).



▶ **Figure 14.8** There are an infinite number of hyperplanes that separate two linearly separable classes.



▶ **Figure 15.7** A collection of training examples. Each R denotes a training example labeled *relevant*, while each N is a training example labeled *nonrelevant*.

```
ApplyLinearClassifier($\vec{w}, b, \vec{x}$)
1  $score \leftarrow \sum_{i=1}^{M} w_i x_i$
2  if $score > b$
3     then return 1
4     else return 0
```

▶ **Figure 14.9** Linear classification algorithm.

We can generalize this 2D linear classifier to higher dimensions by defining a hyperplane as we did in Equation (14.2), repeated here as Equation (14.3):

$$\vec{w}^T \vec{x} = b \tag{14.3}$$

The assignment criterion then is: assign to c if $\vec{w}^T\vec{x} > b$ and to $\overline{c}$ if $\vec{w}^T\vec{x} \leq b$. We call a hyperplane that we use as a linear classifier a *decision hyperplane*.

DECISION HYPERPLANE

The corresponding algorithm for linear classification in M dimensions is shown in Figure 14.9. Linear classification at first seems trivial given the simplicity of this algorithm. However, the difficulty is in training the linear classifier, that is, in determining the parameters $\vec{w}$ and b based on the training set. In general, some learning methods compute much better parameters than others where our criterion for evaluating the quality of a learning method is the effectiveness of the learned linear classifier on new data.

We can derive the linearity of Naive Bayes from its decision rule, which chooses the category c with the largest $\hat{P}(c|d)$ (Figure 13.2, page 260) where:

$$\hat{P}(c|d) \propto \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

and n_d is the number of tokens in the document that are part of the vocabulary. Denoting the complement category as $\bar{c}$, we obtain for the log odds:

$$\log \frac{\hat{P}(c|d)}{\hat{P}(\bar{c}|d)} = \log \frac{\hat{P}(c)}{\hat{P}(\bar{c})} + \sum_{1 \leq k \leq n_d} \log \frac{\hat{P}(t_k|c)}{\hat{P}(t_k|\bar{c})} \quad (14.5)$$

We choose class c if the odds are greater than 1 or, equivalently, if the log odds are greater than 0. It is easy to see that Equation (14.5) is an instance of Equation (14.3) for $w_i = \log[\hat{P}(t_i|c)/\hat{P}(t_i|\bar{c})]$, $x_i =$ number of occurrences of t_i in d, and $b = -\log[\hat{P}(c)/\hat{P}(\bar{c})]$. Here, the index i, $1 \leq i \leq M$, refers to terms of the vocabulary (not to positions in d as k does; cf. Section 13.4.1, page 270) and $\vec{x}$ and $\vec{w}$ are M-dimensional vectors. So in log space, Naive Bayes is a linear classifier.

t_i	w_i	d_{1i}	d_{2i}	t_i	w_i	d_{1i}	d_{2i}
prime	0.70	0	1	dlrs	-0.71	1	1
rate	0.67	1	0	world	-0.35	1	0
interest	0.63	0	0	sees	-0.33	0	0
rates	0.60	0	0	year	-0.25	0	0
discount	0.46	1	0	group	-0.24	0	0
bundesbank	0.43	0	0	dlr	-0.24	0	0

▶ **Table 14.4** A linear classifier. The dimensions t_i and parameters w_i of a linear classifier for the class *interest* (as in interest rate) in Reuters-21578. The threshold is $b = 0$. Terms like dlr and world have negative weights because they are indicators for the competing class *currency*.

Example 14.3: Table 14.4 defines a linear classifier for the category *interest* in Reuters-21578 (see Section 13.6, page 279). We assign document $\vec{d}_1$ "rate discount dlrs world" to *interest* since $\vec{w}^T\vec{d}_1 = 0.67 \cdot 1 + 0.46 \cdot 1 + (-0.71) \cdot 1 + (-0.35) \cdot 1 = 0.07 > 0 = b$. We assign $\vec{d}_2$ "prime dlrs" to the complement class (not in *interest*) since $\vec{w}^T\vec{d}_2 = -0.01 \leq b$. For simplicity, we assume a simple binary vector representation in this example: 1 for occurring terms, 0 for non-occurring terms.

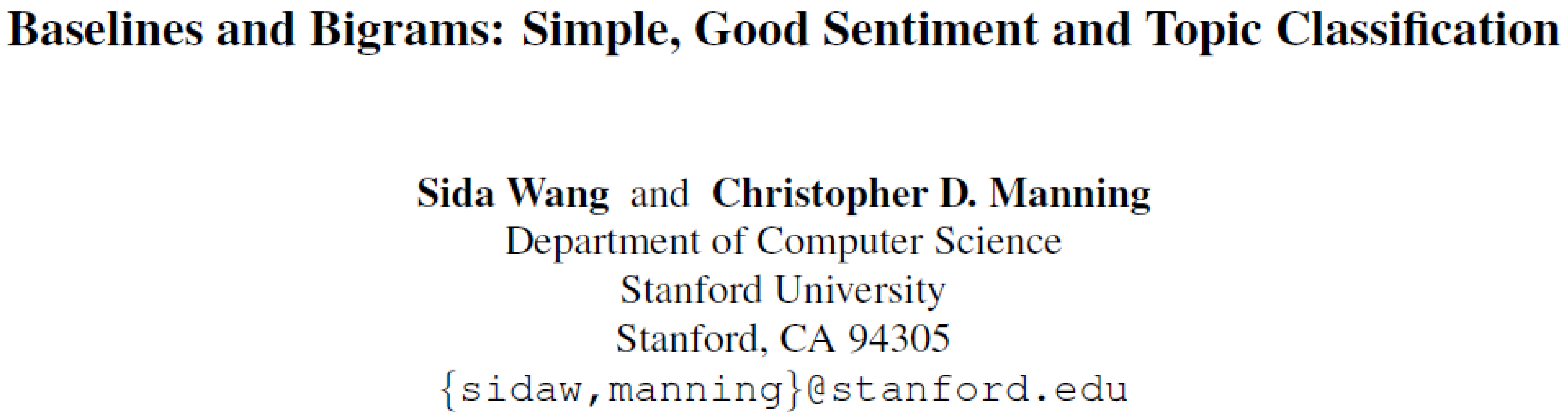
```
# IR14C.py CS5154/6054 cheng 2022
# NB top features by parameter weights
# Usage: python IR14C.py
f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N =len(docs)
trainX = np.concatenate([docs[0:1000], docs[N-1000:N]])
y = np.concatenate([np.zeros(1000, dtype=np.int16), np.ones(1000, dtype=np.int16)])

cv = CountVectorizer(binary=True, max_df=0.4, min_df=4)
X = cv.fit_transform(trainX).toarray()
print(X.shape)
voc = cv.get_feature_names()

model = BernoulliNB()
model.fit(X, y)
logprob = model.feature_log_prob_
logodds = logprob[0] - logprob[1]
vocpos = logodds.argsort()
for i in vocpos[:10]:
   print(voc[i], logodds[i])
for i in vocpos[-10:]:
   print(voc[i], logodds[i])
```

```
christ -4.1588830833596715
jesus -4.07753744390572
faith -4.007333185232471
world -3.6888794541139363
works -3.5263605246161616
throne -3.4965075614664802
holy -3.4965075614664802
written -3.258096538021482
write -3.258096538021482
sins -3.1780538303479458
abimelech 3.1780538303479458
jacob 3.295836866004329
leah 3.295836866004329
rebekah 3.332204510175204
field 3.4011973816621555
sarah 3.4339872044851463
rachel 3.4965075614664802
cattle 3.5263605246161616
laban 3.8066624897703196
abram 3.8918202981106265
```

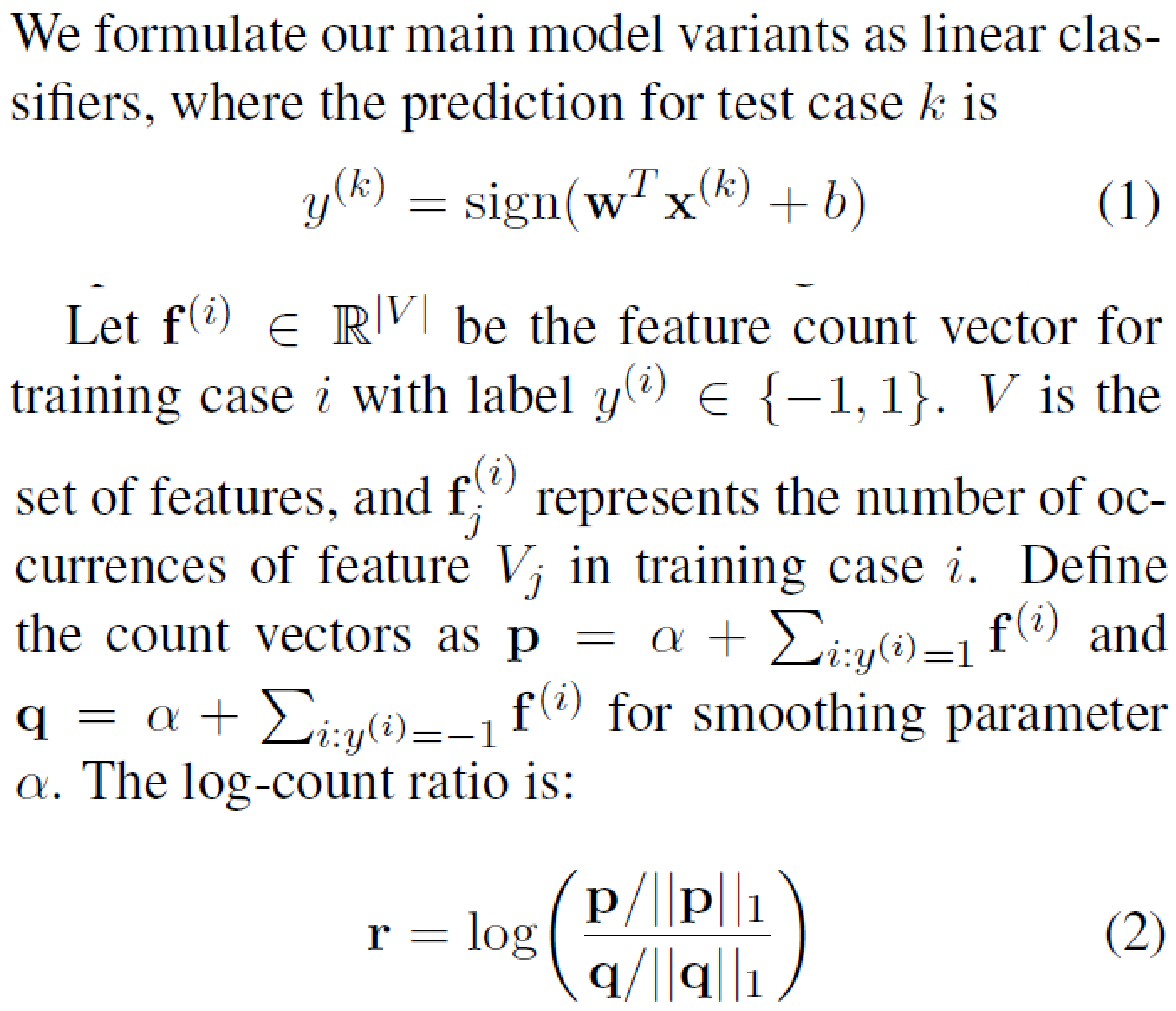
ACL'12: MNB and NBSVM



Baselines and Bigrams: Simple, Good Sentiment and Topic Classification

Sida Wang and **Christopher D. Manning**
Department of Computer Science
Stanford University
Stanford, CA 94305
{sidaw,manning}@stanford.edu

Our results	RT-2k	IMDB	Subj.
MNB-uni	83.45	83.55	**92.58**
MNB-bi	85.85	86.59	**<u>93.56</u>**
SVM-uni	86.25	86.95	90.84
SVM-bi	87.40	**89.16**	91.74
NBSVM-uni	87.80	88.29	92.40
NBSVM-bi	**89.45**	**<u>91.22</u>**	**93.18**
BoW (bnc)	85.45	87.8	87.77
BoW (bΔt$'$c)	85.8	88.23	85.65
LDA	66.7	67.42	66.65
Full+BoW	87.85	88.33	88.45
Full+Unlab'd+BoW	**88.9**	88.89	88.13
BoWSVM	87.15	–	90.00
Valence Shifter	86.2	–	–
tf.Δidf	88.1	–	–
Appr. Taxonomy	**<u>90.20</u>**	–	–
WRRBM	–	87.42	–
WRRBM + BoW(bnc)	–	**89.23**	–



We formulate our main model variants as linear classifiers, where the prediction for test case k is

$$y^{(k)} = \text{sign}(\mathbf{w}^T \mathbf{x}^{(k)} + b) \tag{1}$$

Let $\mathbf{f}^{(i)} \in \mathbb{R}^{|V|}$ be the feature count vector for training case i with label $y^{(i)} \in \{-1, 1\}$. V is the set of features, and $\mathbf{f}_j^{(i)}$ represents the number of occurrences of feature V_j in training case i. Define the count vectors as $\mathbf{p} = \alpha + \sum_{i:y^{(i)}=1} \mathbf{f}^{(i)}$ and $\mathbf{q} = \alpha + \sum_{i:y^{(i)}=-1} \mathbf{f}^{(i)}$ for smoothing parameter α. The log-count ratio is:

$$\mathbf{r} = \log\left(\frac{\mathbf{p}/||\mathbf{p}||_1}{\mathbf{q}/||\mathbf{q}||_1}\right) \tag{2}$$

Wang and Manning

2.1 Multinomial Naive Bayes (MNB)

In MNB, $\mathbf{x}^{(k)} = \mathbf{f}^{(k)}$, $\mathbf{w} = \mathbf{r}$ and $b = \log(N_+/N_-)$. N_+, N_- are the number of positive and negative training cases. However, as in (Metsis et al., 2006), we find that binarizing $\mathbf{f}^{(k)}$ is better. We take $\mathbf{x}^{(k)} = \hat{\mathbf{f}}^{(k)} = \mathbf{1}\{\mathbf{f}^{(k)} > 0\}$, where $\mathbf{1}$ is the indicator function. $\hat{\mathbf{p}}, \hat{\mathbf{q}}, \hat{\mathbf{r}}$ are calculated using $\hat{\mathbf{f}}^{(i)}$ instead of $\mathbf{f}^{(i)}$ in (2).