n-Gramm-Modelle

Vorlesung “Computerlinguistische Techniken”

Alexander Koller

1. Dezember 2014

Wahrscheinlichkeit und Sprache

- Ausgangsfrage: Nächstes Wort vorhersagen.

- Sprache als Zufallsprozess:
  - Für jede Position t im Text: ZV X_t
  - Wort w_t an Position t wird zufällig aus X_t erzeugt (abhängig von Wörtern w_1, ..., w_{t-1})

Sprachmodellierung

- Kann daraus $P(w_1 \ldots w_N)$ berechnen:
  $P(w_1 \ldots w_N) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2)$
  $\cdot \ldots \cdot P(w_N \mid w_1, \ldots, w_{n-1})$

- Schätzen aus *relativer Häufigkeit* in Korpus:
  - $C(w_1 \ldots w_n)$ *absolute Häufigkeit*
  - $f(w_1 \ldots w_n) = C(w_1 \ldots w_n) / N$ *relative Häufigkeit*

Das Sparse-Data-Problem

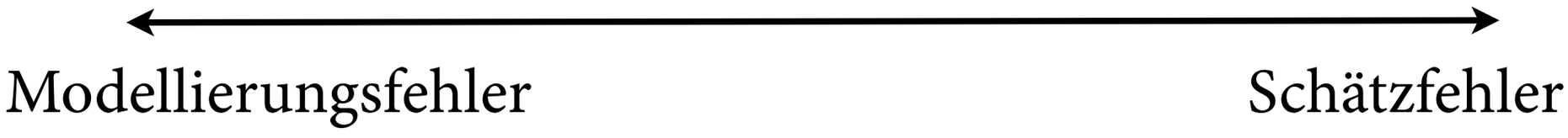
- Sagen wir, eine natürliche Sprache hat 10^5 Wörter.

n = 0	uniform	1 Parameter
n = 1	P(w	10
n = 2	P(w	10
n = 3	P(w	10

- Zum Vergleich:
  - Brown-Korpus: ca. 10^6 Tokens
  - Gigaword-Korpus: ca. 10^9 Tokens

Das Sparse-Data-Problem

- Schon für n = 3 mehr Kontexte als Tokens in den größten verfügbaren Korpora.
  - relevante Ereignisse niemals beobachtet
  - Unterscheidung ungesehen vs. unmöglich?
  - durch Zipfsches Gesetz noch weiter verschärft
- Wir haben nicht genug Daten, um W. vernünftig durch Häufigkeiten zu schätzen!



Lösungsansätze

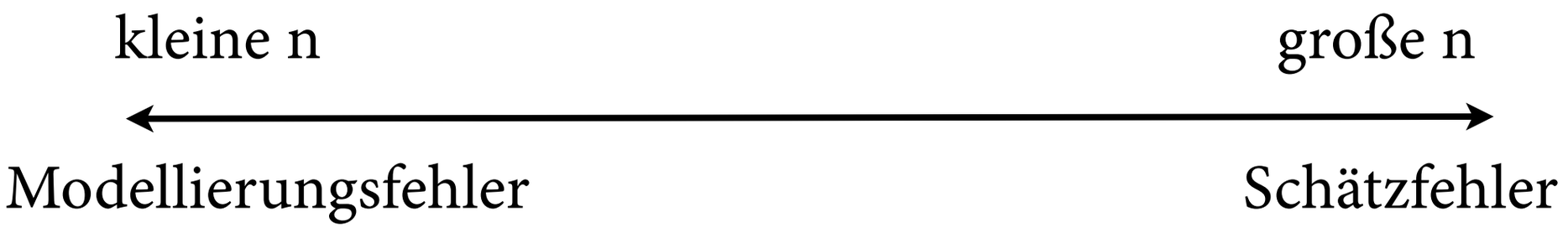
- Unabhängigkeitsannahmen:
  - wenn wir uns auf Kontexte der Länge 1-2 einschränken, haben wir für jeden Kontext genug Trainingsdaten
  - n-Gramm-Modelle
- Smoothing:
  - W.masse von gesehenen auf ungesehene Ereignisse umverteilen

n-Gramm-Modell

- Wir machen die *Markov-Annahme:*
  $P(w_t \mid w_1, ..., w_{t-1}) = P(w_t \mid w_{t-n}, ..., w_{t-1})$

- Das bedeutet: W. von w_t ist abhängig nur von den letzten n-1 Wörtern, unabhängig von allen früheren.

- Damit Tradeoff zwischen Genauigkeit und benötigter Datenmenge kontrollierbar.



“0-gramm”-Modelle

- Alle Wörter sind *gleichverteilt*,
  d.h. $P(X_t = \text{“ein”}) = P(X_t = \text{“Metall”}) = \ldots$
  - Relevanter Kontext: nicht mal X_t selbst
  - wenn es N Wörter in der Sprache gibt, ist also
    $P(X_t = w) = 1/N$ für alle Wörter w

- Größenordnung:
  - Sprache hat ca. $100.000 = 10^5$ Wörter, plus Eigennamen
  - $P(X_t = a) =$ ca. 10^{-5} für jedes Wort a

Gleichverteilung

- Dieser Ansatz ist nicht sinnvoll:
  P(“Kupfer ist ein Metall”) = 10^{-20}
  P(“Kupfer ist ein Käsebrot”) = 10^{-20}
  P(“Kupfer ist ein sich”) = 10^{-20}
  P(“investiert Karussell estnisch ja”) = 10^{-20}

- Wörter sind verschieden häufig. Häufige Wörter sollten vom Zufallsprozess häufiger erzeugt werden als seltene.

Unigramm-Modelle

- Nächster Ansatz: Schätze $P(w) \approx f(w)$.
- Ergebnis: im (deutschen) Tiger-Korpus ist
  - N = 888238
  - C(Metall) = 74, also f(Metall) = 8· 10-5
  - C(ein) = 16605, also f(ein) = 0.02
  - häufigstes Wort ist “der” mit f(der) = 0.1
  - Schätze also $P(\text{Metall}) \approx 8 \cdot 10^{-5}$, $P(\text{ein}) \approx 0.02$ usw.

Unigramm-Modelle

- Das hilft schon mal weiter:
  - P(“Kupfer ist ein Metall”) = $1.5 \cdot 10^{-13}$
  - P(“investiert Karussell estnisch ja”) = $6 \cdot 10^{-20}$
- Probleme:
  - Reihenfolge der Wörter wird ignoriert:
    P(“Metall ein ist Kupfer”) = $1.5 \cdot 10^{-13}$
  - Unabhängigkeitsannahme in der Praxis verletzt:
    P(“der der der der”) = 10^{-4}
  - Manche Wörter kommen im Korpus nicht vor:
    P(“Hans isst ein Käsebrot”) = 0

Bigramm-Modelle

- Verwende Kontext der Länge 1: $P(w_2 \mid w_1)$

- Löst einige Probleme von Unigramm-Modellen:
  - P(“Metall ein ist Kupfer”)
    = $P(\text{Metall}) \cdot P(\text{ein} \mid \text{Metall}) \cdot P(\text{ist} \mid \text{ein}) \cdot P(\text{Kupfer} \mid \text{ist})$
    sollte viel kleiner sein als P(“Kupfer ist ein Metall”)
  - P(“der der der der”) mit Unigramm: $P(\text{der})^4 = 8 \cdot 10^{-6}$
    — mit Bigramm: $P(\text{der}) \cdot P(\text{der} \mid \text{der})^3 = 1.2 \cdot 10^{-11}$

- Ist weiterhin eingeschränkt:
  - P(“der zug kommt auf gleis 3 an”)
    = … $\cdot$ $P(\text{auf} \mid \text{kommt})$ $\cdot$ … $P(\text{an} \mid 3)$

Training

- Wie schätzt man n-Gramm-Modelle?
  - für jedes n-Tupel von Wörtern brauchen wir $P(w_n \mid w_1, ..., w_{n-1})$

- Idee, wie schon erwähnt:
  W. durch relative Häufigkeit abschätzen.

Training: Bigramm-Beispiel

(Brown-Korpus)

don't eat the daisies

cannot eat the skin

don't eat the cereal

we eat the entire

would eat up all

to eat up his

to eat up my

they eat chickens sometimes

they eat chickens and

C(eat) = 61

C(eat, the) = 5
⇒ P(the | eat)= 0.08

C(eat, up) = 4
⇒ P(up | eat) = 0.06

C(eat, trains) = 0
⇒ P(trains | eat) = 0

Training: Bigramm-Modelle

- $P(w_2 \mid w_1)$ ist die W., dass w_2 kommt,
  nachdem wir direkt davor w_1 gesehen haben.

- Berechne also:
  - Häufigkeit $C(w_1\, w_2)$ des Teilstrings $w_1\, w_2$ im Korpus
  - Häufigkeit von w_1 im Korpus:

$$C(w_1) = \sum_w C(w_1 w)$$

- Schätze:

$$P(w_2|w_1) = \frac{C(w_1 w_2)}{C(w_1)}$$

Training: Allgemein

- Idee fortsetzen auf $P(w_n \mid w_1, ..., w_{n-1})$:
  - berechne $C(w_1 ... w_n)$ aus Korpus
  - berechne $C(w_1 ... w_{n-1})$ aus Korpus

- Dann schätze:

$$P(w_n|w_1, \ldots, w_{n-1}) = \frac{C(w_1 \ldots w_{n-1} w_n)}{C(w_1 \ldots w_{n-1})}$$

(Anteil des Ereignisses $w_1 \ldots w_n$ an den Malen, die es die Chance hatte, aufzutreten)

Grundprinzip

- Umrechnen von gemeinsamer Verteilung P(X = a, Y = b) in bedingte Verteilung P(X = a | Y = b).

- *Marginalverteilung:* $P(Y = b) = \sum_{a'} P(X = a', Y = b)$

- Damit geht es dann so:

$$P(X = a|Y = b) = \frac{P(X = a, Y = b)}{P(Y = b)}$$

$$= \frac{P(X = a, Y = b)}{\sum_{a'} P(X = a', Y = b)}$$

Angewandt auf n-Gramme

$$P(X = a|Y = b) = \frac{P(X = a, Y = b)}{\sum_{a'} P(X = a', Y = b)}$$

- Daher:

$$P(w_n|w_1, \ldots, w_{n-1}) = \frac{P(w_1, \ldots, w_{n-1}, w_n)}{\sum_{w'} P(w_1, \ldots, w_{n-1}, w')}$$

$$\approx \frac{C(w_1, \ldots, w_{n-1}, w_n)}{N} \cdot \frac{N}{\sum_{w'} C(w_1, \ldots, w_{n-1}, w')}$$

$$= \frac{C(w_1, \ldots, w_n)}{\sum_{w'} C(w_1, \ldots, w_{n-1}, w')}$$

$$= \frac{C(w_1, \ldots, w_n)}{C(w_1, \ldots, w_{n-1})}$$

Maximum Likelihood Estimation

- *Likelihood* eines Korpus gegeben Modell: $P(K \mid M)$
  - für Bigramme: $P(K \mid M) = P(w_1) \cdot P(w_2 \mid w_1) \cdot \ldots \cdot P(w_N \mid w_{N-1})$
  - Modell definiert durch *Parameter* $P(w_t \mid w_{t-1})$

- *Maximum-Likelihood-Schätzung* berechnet ein Modell M, das Likelihood des Korpus maximiert.

- Man kann zeigen: Schätzung mit relativen Häufigkeiten ist eine Form von ML-Schätzung.

Ungelöstes Problem

- P(Kupfer ist ein Metall)
  = P(Kupfer) · P(ist | Kupfer) · P(ein | ist) · P(Metall | ein)
  = 4.5e-6 · 0.5 · 0.04 · 0
  = 0

- P(Metall ein ist Kupfer)
  = P(Metall) · P(ein | Metall) · P(ist | ein) * P(Kupfer | ist)
  = 7.4e-5 · 0 · 0 · 0
  = 0

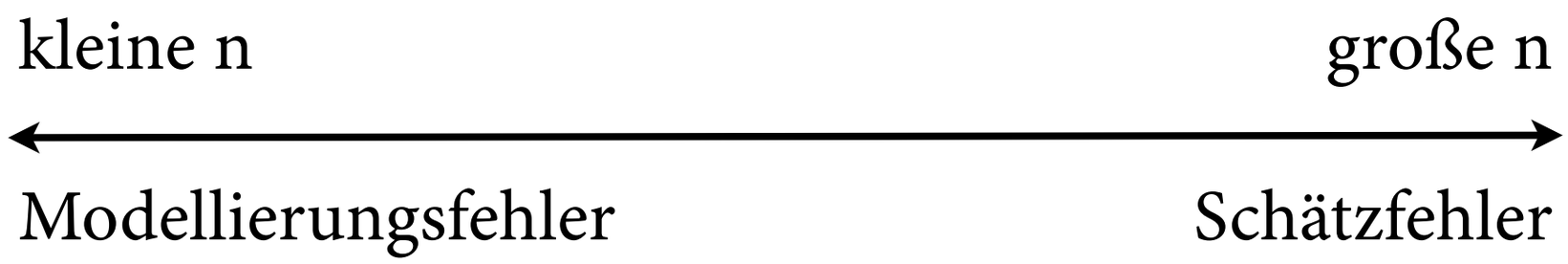
- Wie gehen wir mit ungesehenen Bigrammen um?

Smoothing

- n-Gramme: Systematische Kontrolle über Data Sparseness.

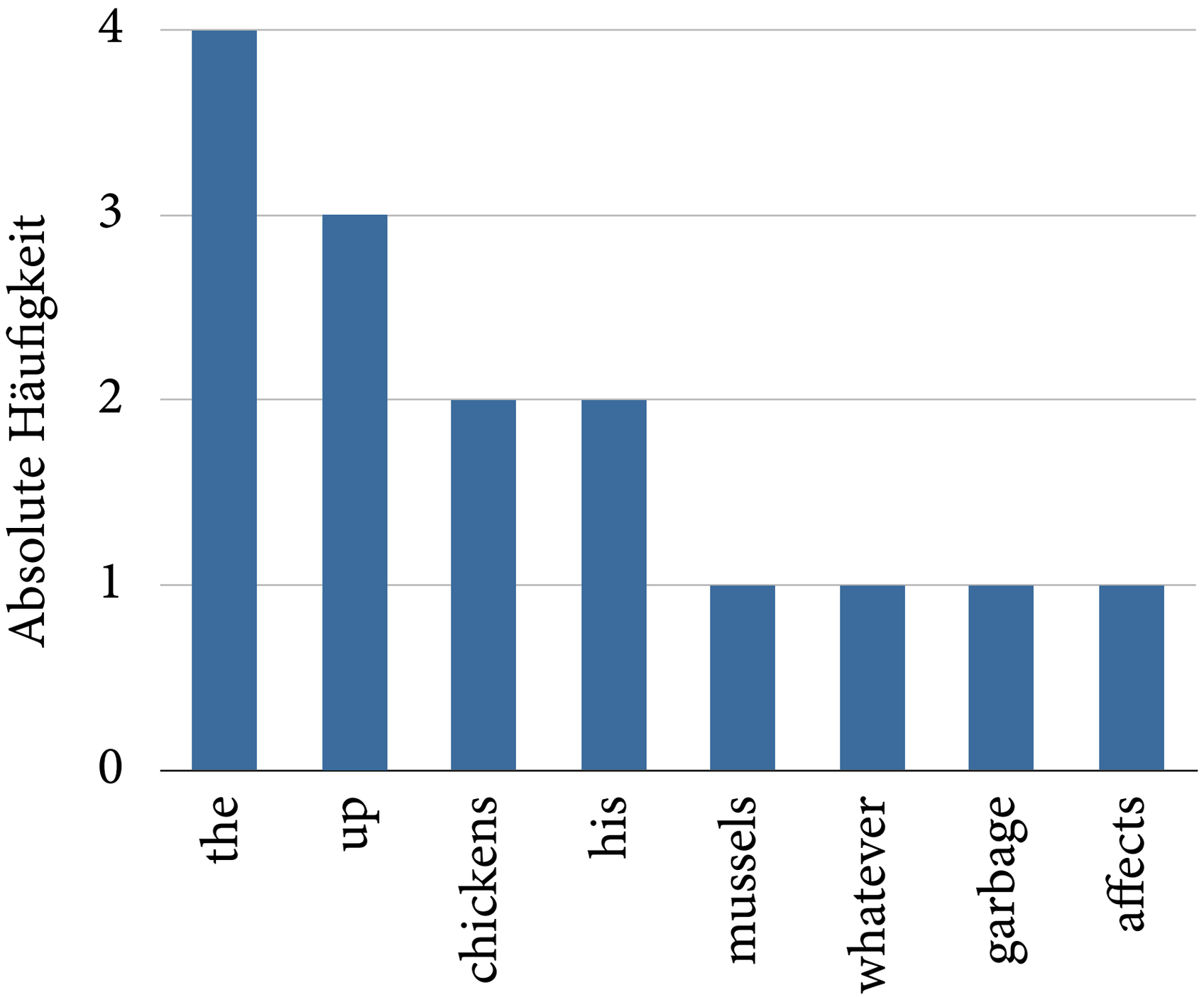
- Aber Data Sparseness ist immer noch ein Problem (zur Erinnerung: 10^{10} Bigramme)

- Wie unterscheidet man zwischen “unmöglich” und “zufällig nicht beobachtet”?



Ein Beispiel

Bigramme für “eat X” im Brown-Korpus



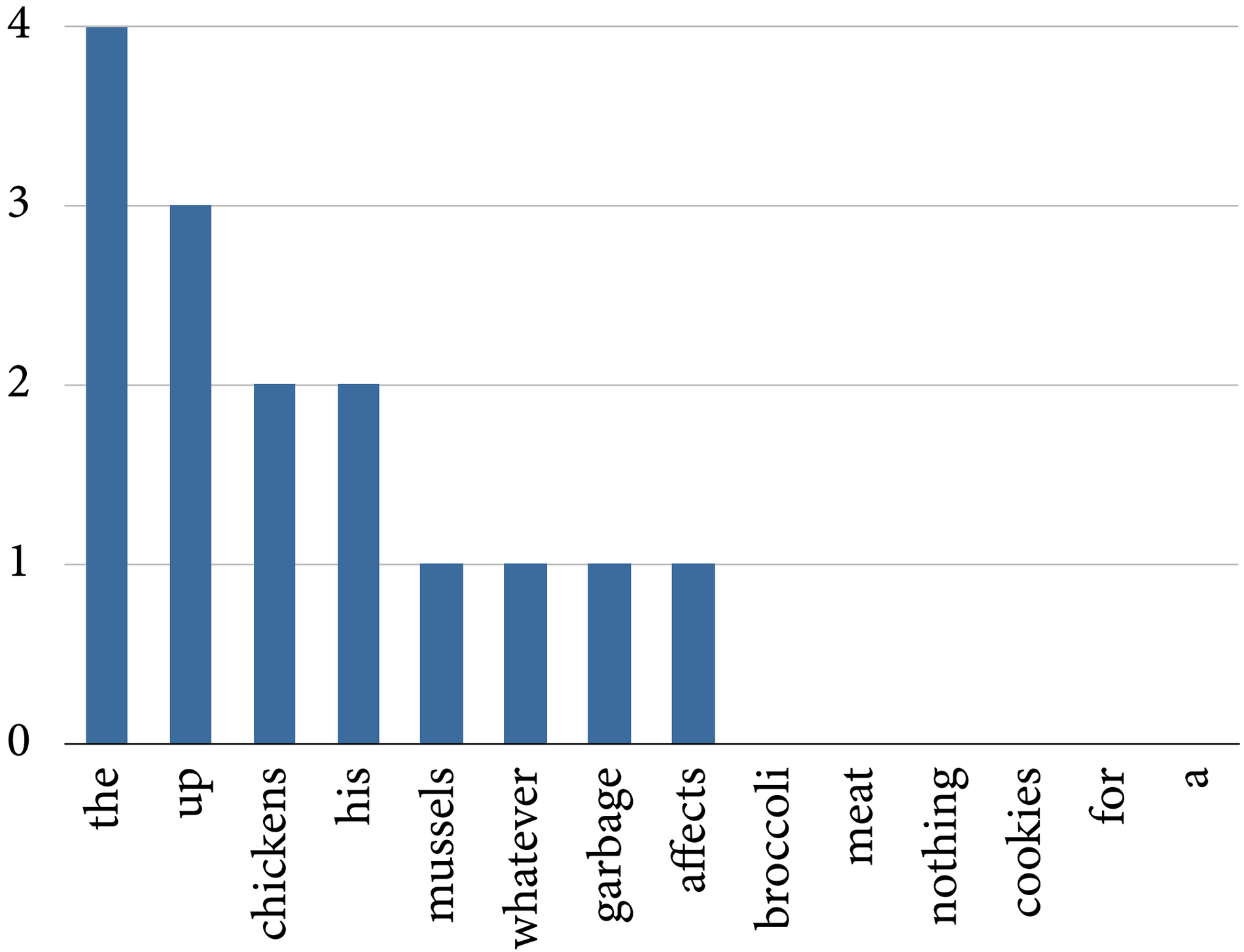
Ungesehen vs. unmöglich

- Im ganzen Brown-Korpus kommt “eat” nur mit 38 verschiedenen nächsten Wörtern vor.

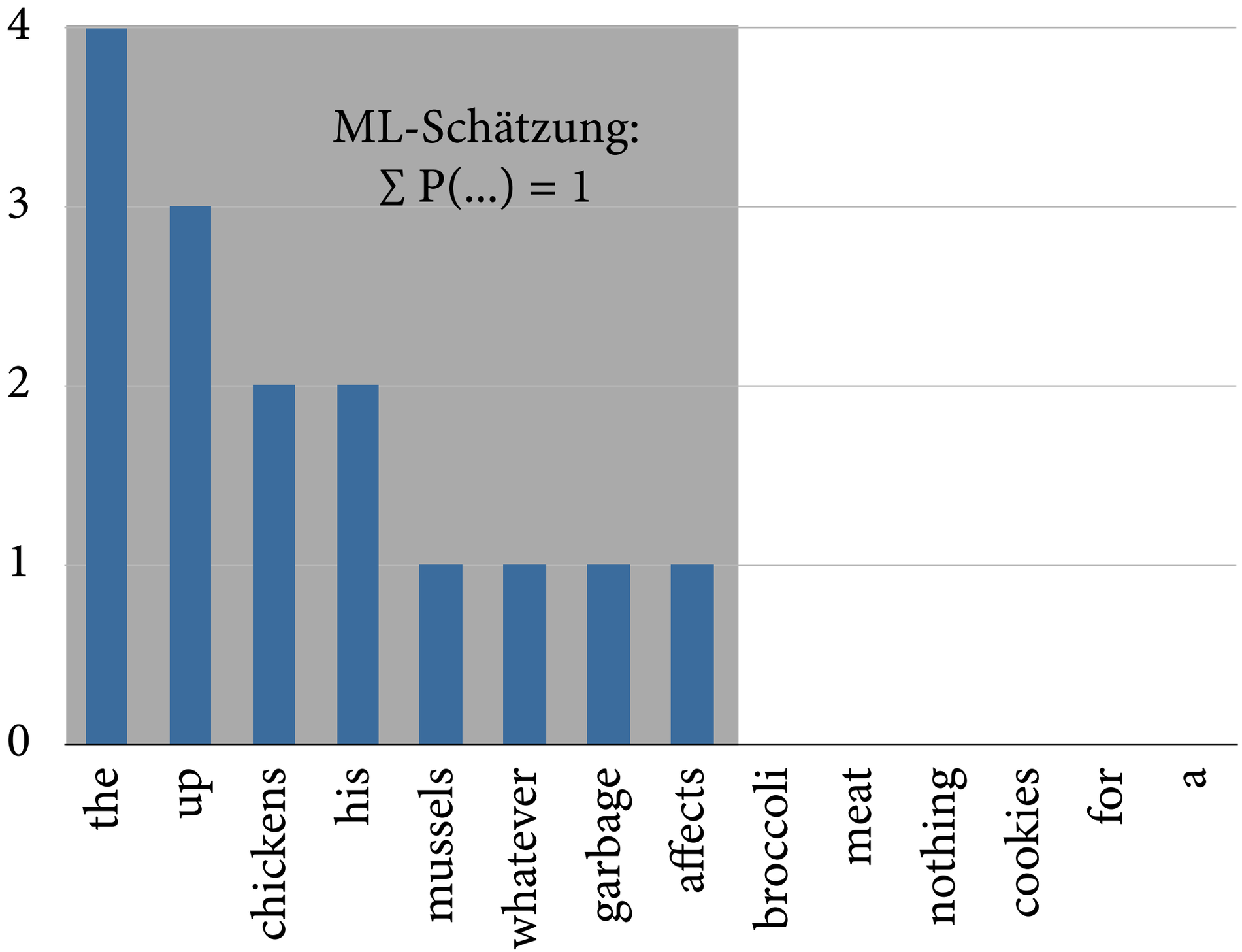
- Das bedeutet nicht, dass nach “eat” keine anderen Wörter kommen können; wir haben sie nur nicht beobachtet.

- *Smoothing:* einige W.masse auf die ungesehenen Ereignisse verteilen.

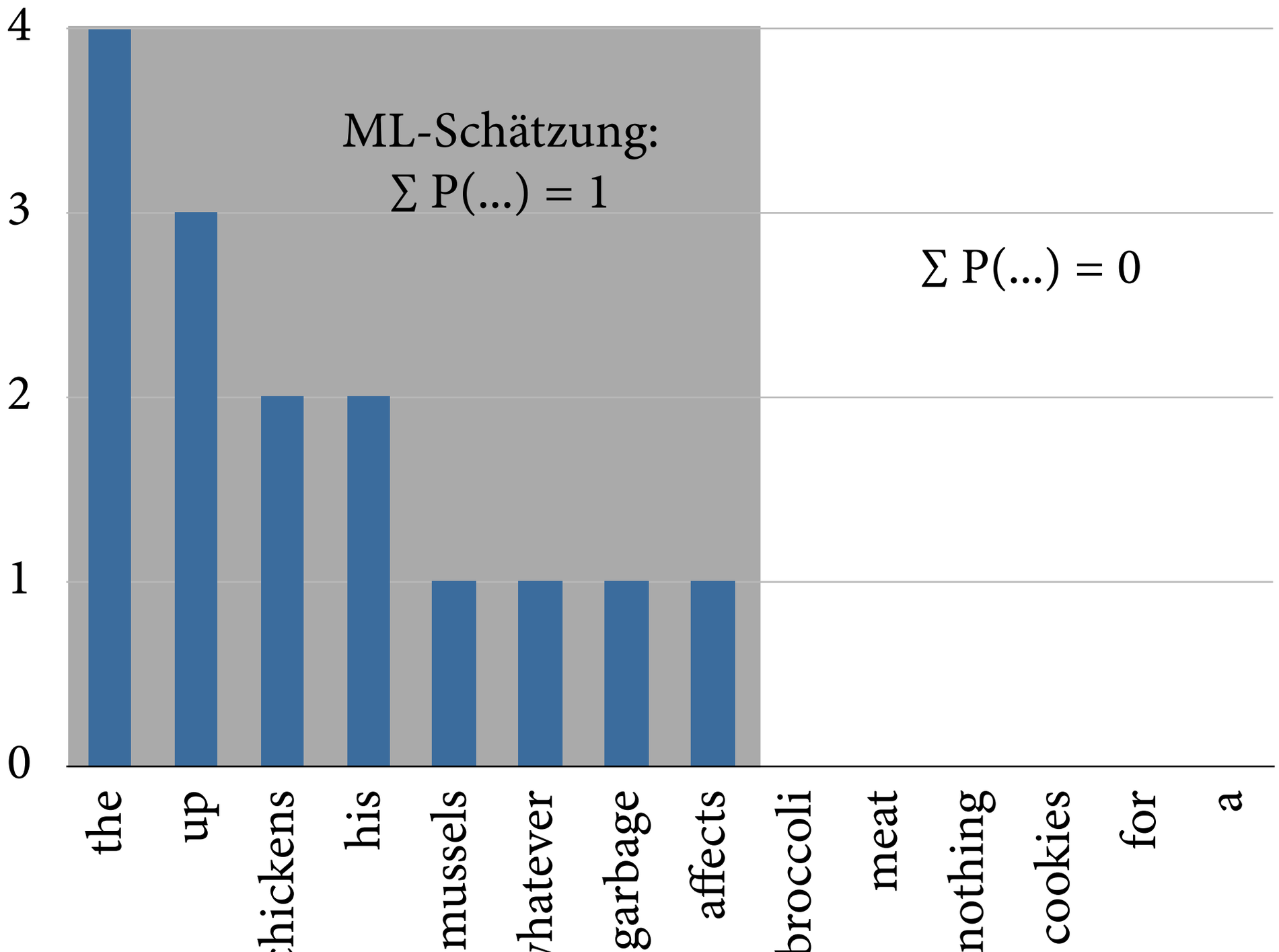
Smoothing


4
3
2
1
0
the
up
chickens
his
mussels
whatever
garbage
affects
broccoli
meat
nothing
cookies
for
a

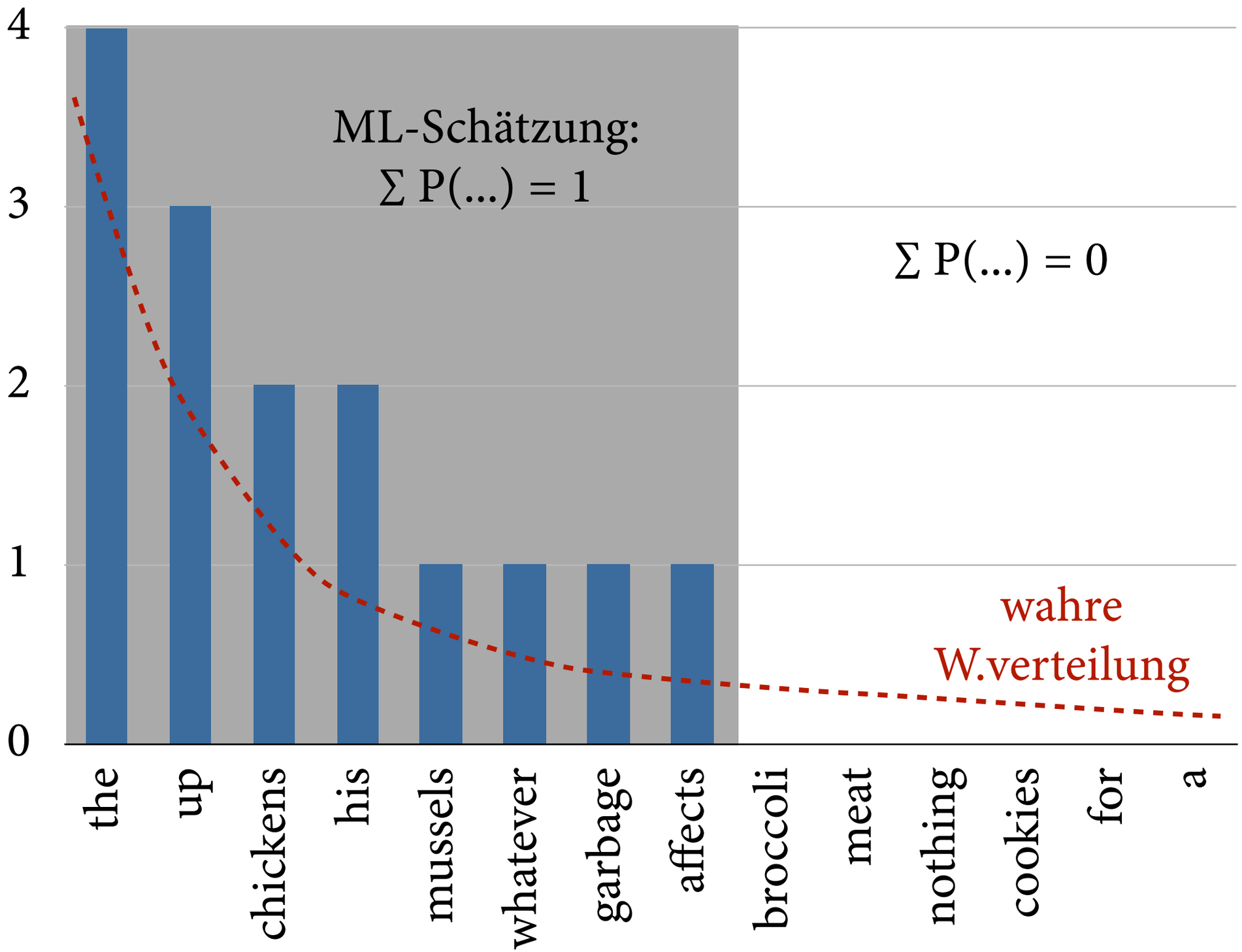
Smoothing


4
3
2
1
0
ML-Schätzung:
Σ P(...) = 1
the
up
chickens
his
mussels
whatever
garbage
affects
broccoli
meat
nothing
cookies
for
a

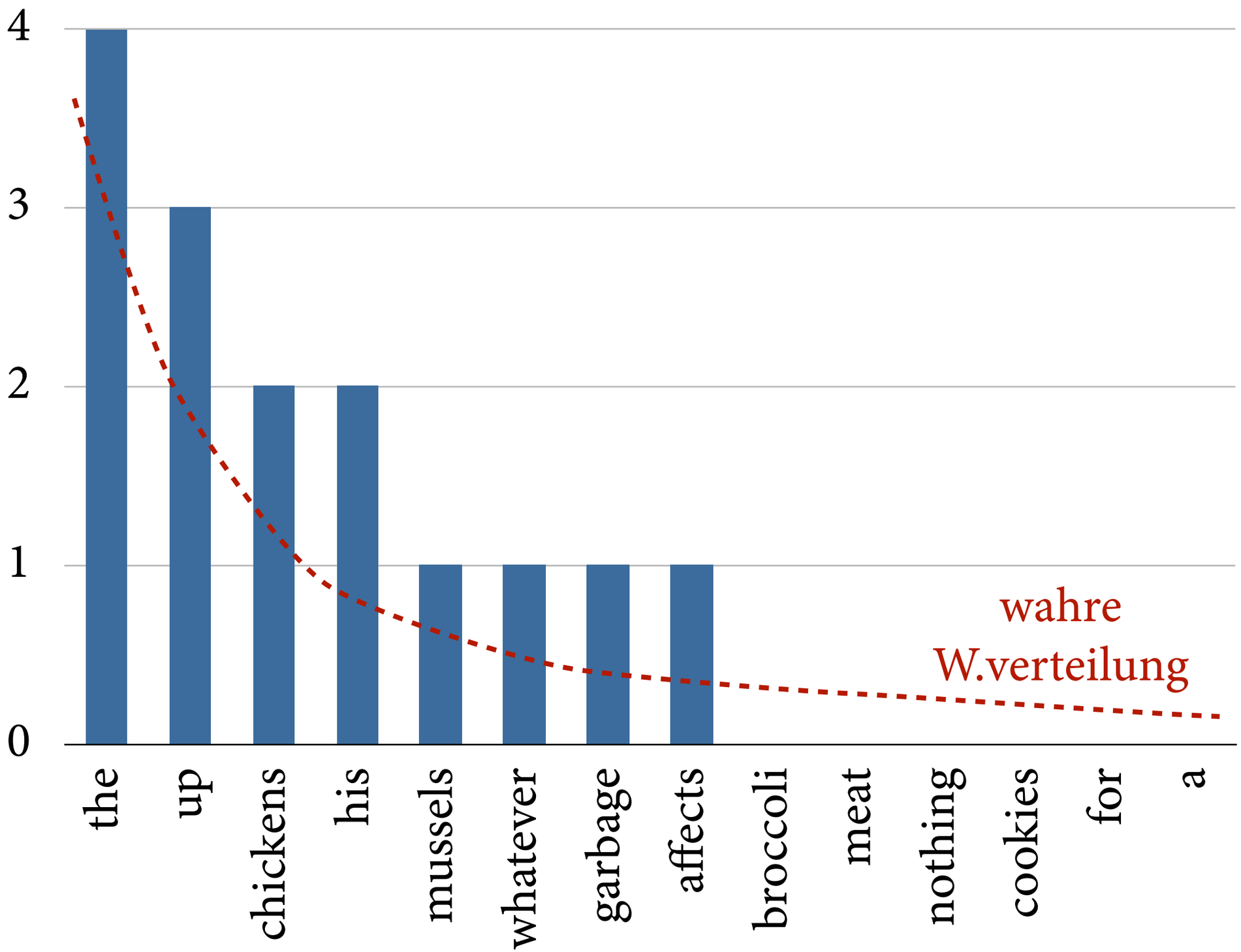
Smoothing


4
3
2
1
0
ML-Schätzung:
∑ P(...) = 1
∑ P(...) = 0
the
up
chickens
his
mussels
whatever
garbage
affects
broccoli
meat
nothing
cookies
for
a

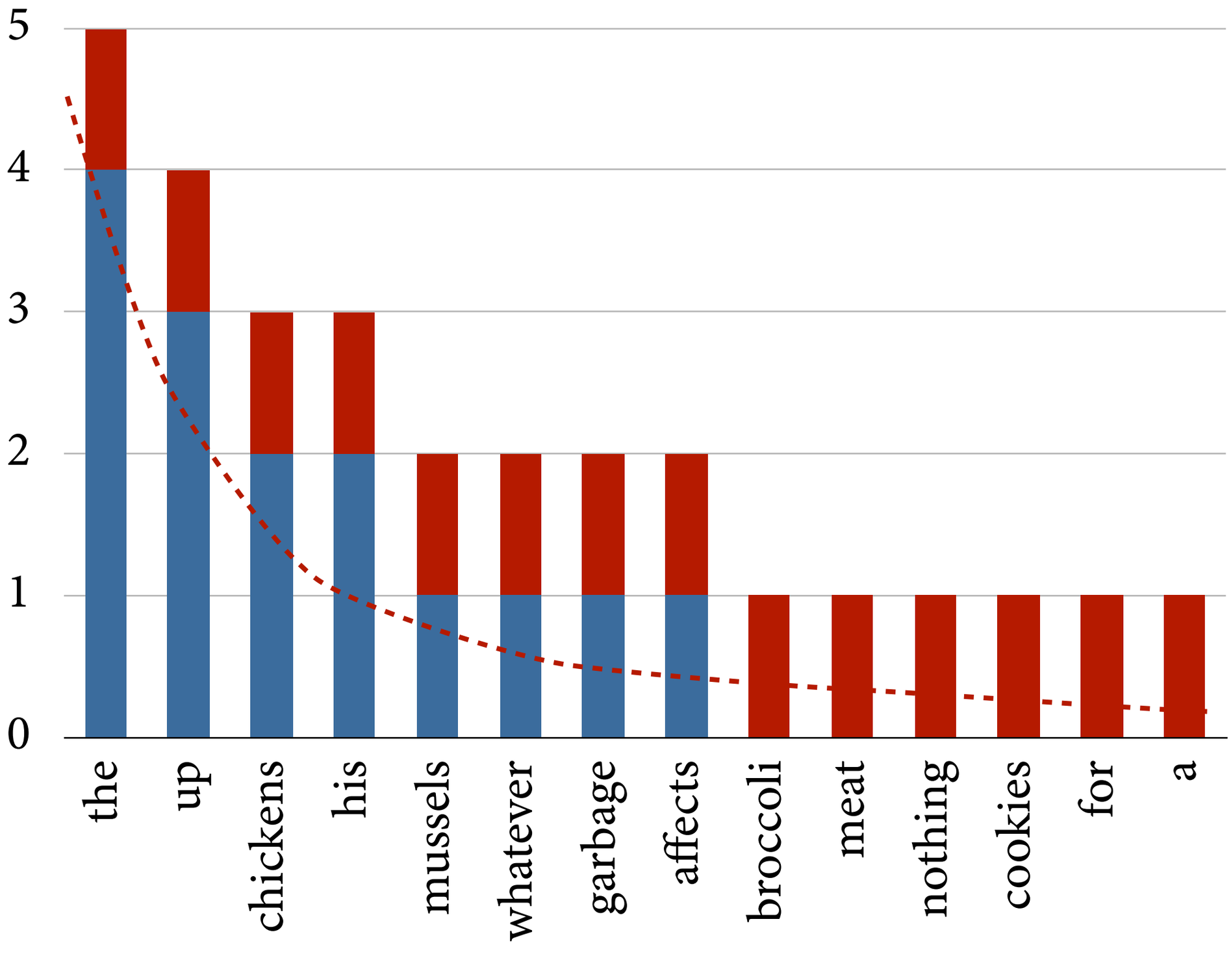
Smoothing


4
3
2
1
0
ML-Schätzung:
∑ P(...) = 1
∑ P(...) = 0
wahre
W.verteilung
the
up
chickens
his
mussels
whatever
garbage
affects
broccoli
meat
nothing
cookies
for
a

Smoothing

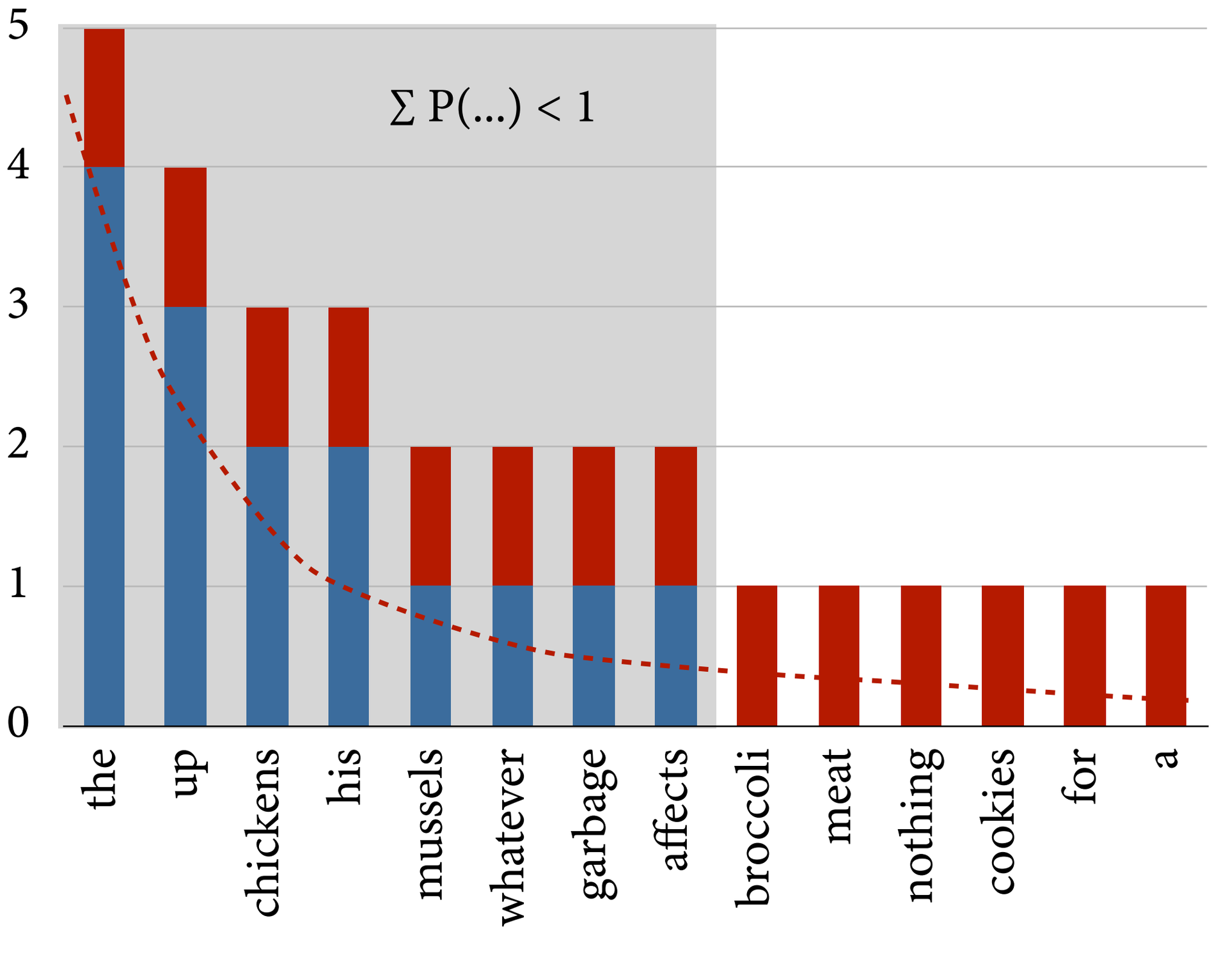

4
3
2
1
0
wahre
W.verteilung
the
up
chickens
his
mussels
whatever
garbage
affects
broccoli
meat
nothing
cookies
for
a

Add-One-Smoothing



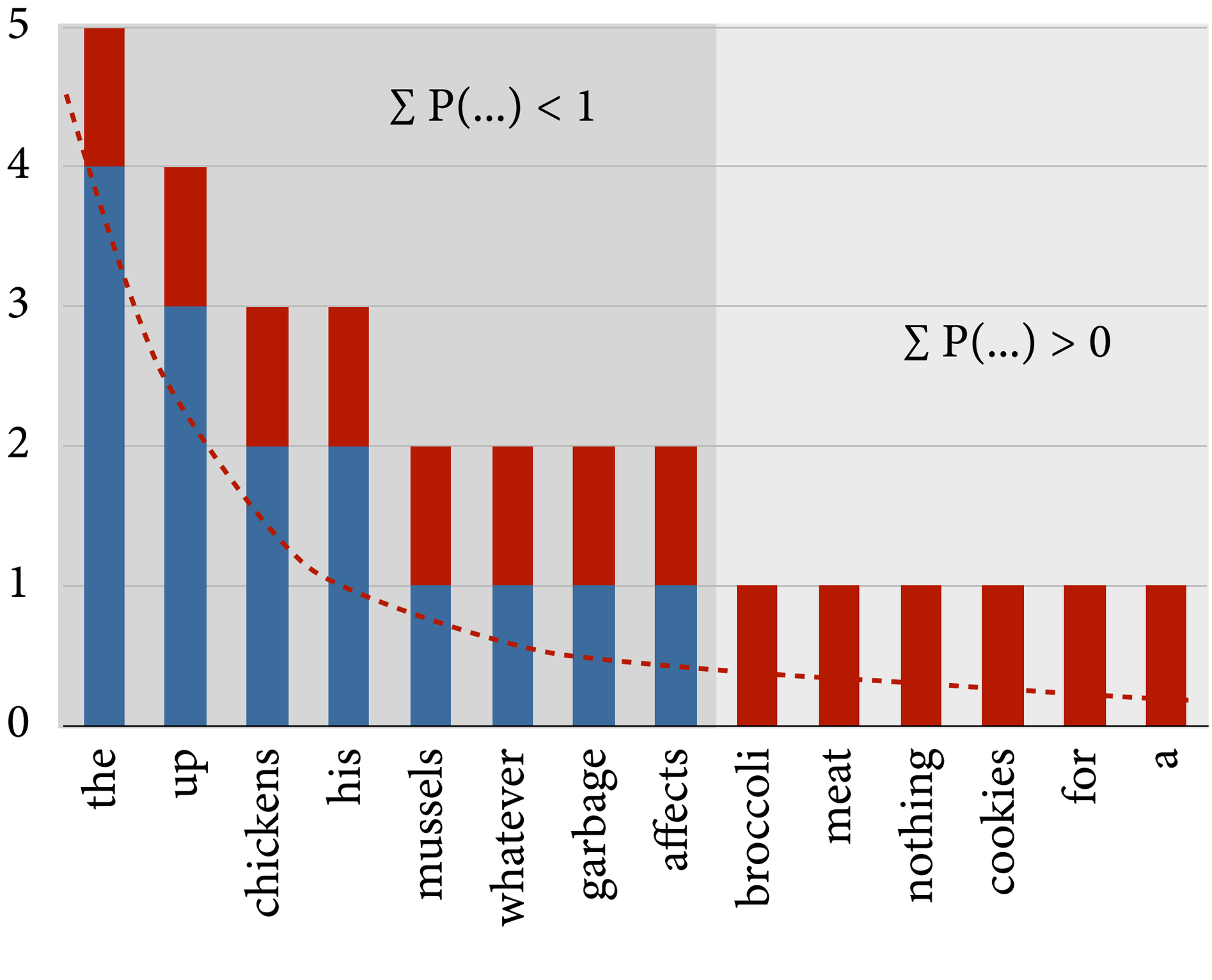
(Laplace)

Add-One-Smoothing



(Laplace)

Add-One-Smoothing



(Laplace)

Add-One-Smoothing

- Idee: Zähle zu allen absoluten Häufigkeiten 1 dazu, auch wenn C vorher 0 war.

- Wenn es V verschiedene Wörter gibt, bekommt man:

$$P(w_2|w_1) \approx \frac{C(w_1 w_2) + 1}{C(w_1) + V}$$

- Für allgemeine n-Gramme:

$$P(w_n|w_1, \ldots, w_{n-1}) \approx \frac{C(w_1 \ldots w_n) + 1}{C(w_1 \ldots w_{n-1}) + V}$$

Add-one in Tiger

- “Kupfer ist ein Metall”:
  - P(ist | Kupfer): C(Kupfer ist) / C(Kupfer) = 2 / 4 = 0.5
    → (C(Kupfer ist)+1) / (C(Kupfer)+V) = 3 / 85236 = 3.4e-5
  - P(ein | ist): C(ist ein) / C(ist) = 181 / 4467 = 0.04
    → (C(ist ein) + 1) / (C(ist) + V) = 182 / 89699 = 0.002
  - P(Metall | ein): C(ein Metall) / C(ein) = 0
    → 1 / V = 1 / 85232 = 1.2e-5
  - P(“Kupfer ist ein Metall”): 0 → 3.7e-18

(ohne Smoothing; mit add-one-Smoothing)

(Andere Version von Tiger als letztes Mal; hier ist N = 888578, V = 85232)

Add-one in Tiger

- “Metall ein ist Kupfer”:
  - P(ein | Metall) = 1/V = 1 / 85232 = 1.2e-5
  - P(ist | ein) = 1/V = 1 / 85232 = 1.2e-5
  - P(Kupfer | ist) = 1/V = 1 / 85232 = 1.2e-5
  - P(“Metall ein ist Kupfer”): 0 → 1.3e-19

Wahrscheinlichkeitsmasse

- Smoothing verteilt W.masse um:
  P(beobachtete) < 1, P(unbeobachtete) > 0.

- Jedes *unbeobachtete* Ereignis w_2 bekommt

$$P(w_2|w_1) = \frac{1}{C(w_1) + V}$$

- W. jedes *beobachteten* Ereignisses w_1 w_2 ungefähr um Faktor $\frac{C(w_1)}{C(w_1) + V}$ reduziert.

Wieviel W.masse?

- In der Praxis ist V groß. Add-One nimmt also sehr viel W.masse von beobachteten Ereignissen weg.
  - Im Restaurant-Beispiel geht P(ist | Kupfer) von 0.5 auf 0.00003 herunter.
  - daher funktioniert Laplace-Smoothing in Praxis nicht gut
- Alternativen:
  - “Lidstone-Smoothing”: statt 1 eine kleinere Zahl für ungesehene Wörter verwenden
  - Discounting (Good-Turing, Witten-Bell, Kneser-Ney): W.masse für gesehene Ereignisse systematisch verringern, Rest auf ungesehene verteilen.

Backoff-Modelle

- Es ist schwerer, ein n-Gramm-Modell zu schätzen als ein n-1-Gramm-Modell.

- Für häufige Wörter bekommt man aber bessere n-Gramm-Schätzungen als für seltene.

- Backoff-Modelle (Grundidee):
  Verwende $P(w_n \mid w_1, ..., w_{n-1})$ für häufige n-Gramme,
  sonst verwende (n-1)-Gramm $P(w_n \mid w_2, ..., w_{n-1})$.

n-Gramme in NLTK

- n-Gramm-Modell in NLTK (2!) implementiert als Klasse NgramModel.

- Konstruktor nimmt n und eine Liste von Wörtern (= Trainingskorpus) als Argumente:

```
ngram = NgramModel(n, words)
```

- "estimator" zum Schätzen der W.en darf angegeben werden.
  - Dort Smoothing; Default: Katz-Backoff.

n-Gramme in NLTK

- Für gegebene Wörter `c = [w1, ..., wn-1]`:
  `ngram[c]` ist W.verteilung über nächste Wörter.

- (Bedingte) W.verteilungen in NLTK:
  Klasse `nltk.probability.ProbDistI`;
  siehe deren Methoden.

Zusammenfassung

- Jedes Wort ist statistisch abhängig von den Wörtern, die vorher gekommen sind.

- Aber: Zu wenig Daten, um Modell mit beliebigen Kontexten zu schätzen.

- Lösungsansätze:
  - n-Gramm-Modelle: Kontextlänge beschränken
  - Smoothing: positive W. für ungesehene Ereignisse