

## **content**

- answer questions
- review backoff smoothing
- word sense disambiguation

## Folie 63: Wie berechnet man den p-Wert?

### p-Wert

Beim Binomialtest gibt der berechnete Wert direkt die Wahrscheinlichkeit (= **p-Wert**) dafür an, einen Fehler zu machen, wenn wir die Nullhypothese zurückweisen.

Dagegen muss der  $\chi^2$ -Wert erst mit Hilfe einer Tabelle in einen p-Wert **umgerechnet** werden.

Der  $\chi^2$ -Wert misst die Abweichung von den erwarteten Werten in der Kontingenz-Tabelle. Je größer er ist, desto kleiner ist der p-Wert.

Im Beispiel erhalten wir einen  $\chi^2$ -Wert von 1.55, der einem p-Wert von 0.21 entspricht. Das Ergebnis ist also auch bei diesem Test **nicht signifikant**.

Wie die Ergebnisse zeigen, können sich die p-Werte verschiedener statistischer Tests deutlich unterscheiden.

### Folie: 63

convert chi square test statistic to p value:

<https://www.socscistatistics.com/pvalues/chidistribution.aspx>

### Antwort

#### p-Wert vom Binomialtest

Die Wahrscheinlichkeit, 8 oder mehr 1-Ereignisse (hier Wortpaare *new companies*) in  $n=14,307,668$  Wiederholungen eines Bernoulliversuches mit der Wahrscheinlichkeit  $p$  zu bekommen, ist somit

$$b(\geq 8, n, p) = 1 - \sum_{i=0}^{7} b(i, n, p) \approx 0.15$$

⇒ Das Ergebnis ist **nicht** statistisch signifikant.

Beim Chi-Quadrat-Test, wird den Wert  $\chi^2$  berechnet ( $\chi^2$  nennt man Teststatistik). Diesen Wert muss dann zu p-Wert convertiert werden

#### $\chi^2$ Test

Für den  $\chi^2$ -Test brauchen wir die Kontingenztabelle:

|                       | $w_2 = \text{companies}$ | $w_2 \neq \text{companies}$ |                     |
|-----------------------|--------------------------|-----------------------------|---------------------|
| $w_1 = \text{new}$    | $O_{11} = 8$             | $O_{12} = 15820$            | $O_{1-} = 15828$    |
| $w_1 \neq \text{new}$ | $O_{21} = 4667$          | $O_{22} = 14287173$         | $O_{2-} = 14291840$ |
|                       | $O_{-1} = 4675$          | $O_{-2} = 14302993$         | $O_{--} = 14307668$ |

Die  $\chi^2$ -Teststatistik wird folgendermaßen berechnet:

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^2 \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \quad \text{wobei } E_{ij} = p_{i-} p_{-j} O_{--} = \frac{O_{i-} O_{-j}}{O_{--}}$$

## Backoff-Glättung mit Interpolation

$$p(w_i | w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1} x)} + \alpha(w_{i-k}^{i-1}) p(w_i | w_{i-k+1}^{i-1})$$

Main idea:

- Without smoothing,  $p(\text{Buch} | \text{das, rote})$  could be zero especially when we use a high order of ngram.
- E.g. If we estimate  $p(\text{Buch} | \text{das, rote})$  using relative frequency, we have to compute  $f(\text{das, rote, Buch}) / f(\text{das, rote})$ .
- For longer ngrams (such as 3 or 4 or 5 grams), it is likely that the ngram does not appear in the training corpus, meaning  $f$  will be zero.
- Shorter ngrams tend to have a higher chance of occurring. Consider  $f(\text{das, rote, Buch})$  vs  $f(\text{das})$ .
- The idea of backoff smoothing is that we want to compute the probability for every ngram length as a way to prevent zero probability.
- Instead of computing only the prob of the 3 gram  $p(\text{Buch} | \text{das rote})$ , we also compute the prob of the bigram  $p(\text{Buch} | \text{rote})$  and the prob of the unigram  $p(\text{Buch})$
- so we have

$$p(\text{Buch} | \text{das rote}) = p^*(\text{Buch} | \text{das rote}) + p^*(\text{Buch} | \text{rote}) + p^*(\text{Buch})$$

(note: this is not the complete formula of backoff smoothing)

- If  $p^*(\text{Buch} | \text{das rote}) = 0$ , we still have  $p^*(\text{Buch} | \text{rote}) + p^*(\text{Buch})$  left
- If  $p^*(\text{Buch} | \text{das rote}) > 0$ , we will have all  $p^*(\text{Buch} | \text{das rote}) + p^*(\text{Buch} | \text{rote}) + p^*(\text{Buch})$

## Backoff-Glättung mit Interpolation

= relative freq with discount =  $p^*(\text{Buch} \mid \text{das rote})$

$$p(w_i | w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1} x)} + \alpha(w_{i-k}^{i-1}) p(w_i | w_{i-k+1}^{i-1})$$

$$p(\text{Buch} \mid \text{das rote}) = p^*(\text{Buch} \mid \text{das rote}) + p^*(\text{Buch} \mid \text{rote}) + p^*(\text{Buch})$$

- $p^*(\text{Buch} \mid \text{das rote})$  is estimated using the **relative freq with discount**,  $f(\text{das rote Buch}) - \text{discount} / f(\text{ das rote})$
- We also need to add a **backoff factor** so that the prob of every ngram (in the training corpus + unknown ngrams) sums up to 1  

$$p(\text{Buch} \mid \text{das rote}) = p^*(\text{Buch} \mid \text{das rote}) + \text{alpha}(\text{das rote}) [ p^*(\text{Buch} \mid \text{rote}) + \text{alpha}(\text{rote}) p^*(\text{Buch}) ]$$
- At the last backoff step (for unigram), there are different ways to compute  $p(\text{unigram})$ . For example,

(Übung4)

this is  $p(\text{Buch})$

- $p(\text{Buch} \mid \text{das rote}) = p^*(\text{Buch} \mid \text{das rote}) + [\text{alpha}(\text{das rote}) [p^*(\text{Buch} \mid \text{rote}) + \text{alpha}(\text{rote}) [p^*(\text{Buch}) + \text{alpha}(\text{ }) 1/N]]]$ 
  - $p(\text{Buch}) = p^*(\text{Buch}) + \text{alpha}(\text{ }) 1/N$
  - We can see  $1/N$  as  $p(\text{ })$ , meaning we actually stop the recursion at  $p(\text{zero gram})$

this is  $p(\text{ })$

(In the lecture slide)

this is  $p(\text{Buch})$

- $p(\text{Buch} \mid \text{das rote}) = p^*(\text{Buch} \mid \text{das rote}) + \text{alpha}(\text{das rote}) [p^*(\text{Buch} \mid \text{rote}) + \text{alpha}(\text{rote}) [p^*(\text{Buch})]]$ 
  - with  $p(\text{Buch}) = p^*(\text{Buch}) = f(\text{Buch})/N$
  - here we stop at  $p(\text{Buch})$  and compute it as relative frequency without discount

- Es berechnet für jede Sprache  $L$  eine logarithmierte Wahrscheinlichkeit gemäß der Formel:

$$lp_L(a_1, \dots, a_n) = \sum_{i=1}^n \log p_L(a_i | a_{i-k}, \dots, a_{i-1})$$

wobei  $p_L(a_i | a_{i-k}, \dots, a_{i-1})$  jeweils rekursiv wie folgt berechnet wird:

$$p_L(a_i | a_1^{i-1}) = p_L^*(a_i | a_1^{i-1}) + \alpha(a_1^{i-1}) p_L(a_i | a_2^{i-1}) \text{ für } 1 < i \leq n$$

$$p(a_1) = p_L^*(a_1) + \alpha() \frac{1}{1000}$$


---

### Interpolierte Backoff-Glättung

Beispiel:

$$\begin{aligned} p(\text{Buch} | \text{das, rote}) &= p^*(\text{Buch} | \text{das, rote}) + \alpha(\text{das, rote}) ( \\ &\quad p^*(\text{Buch} | \text{rote}) + \alpha(\text{rote}) ( \\ &\quad \quad p^*(\text{Buch})) ) \end{aligned}$$

$$p^*(\text{Buch} | \text{das, rote}) = (f(\text{das, rote, Buch}) - \delta_2) / f(\text{das, rote})$$

$$p^*(\text{Buch} | \text{rote}) = (f(\text{rote, Buch}) - \delta_1) / f(\text{rote})$$

$$\underline{p^*(\text{Buch}) = f(\text{Buch}) / N}$$

# Übung

## Backoff-Glättung mit Interpolation

$$p(w_i | w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1} x)} + \alpha(w_{i-k}^{i-1}) p(w_i | w_{i-k+1}^{i-1})$$

How to compute backoff smoothing with interpolation for

1.  $p(w_3 | w_1, w_2) =$
2.  $p(\text{Uni} | \text{an der}) =$

# Übung

## Backoff-Glättung mit Interpolation

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

How to compute backoff smoothing with interpolation for

1.  $p(w_3|w_1, w_2) = p^*(w_3|w_1, w_2) + \text{alpha}(w_1, w_2) [p^*(w_3|w_2) + \text{alpha}(w_2) [p^*(w_3)]]$  where

- $p^*(w_3|w_1, w_2) = \max(0, f(w_1, w_2, w_3) - \text{discount\_2} / f_{\text{context}}(w_1, w_2))$
- $p^*(w_3|w_2) = \max(0, f(w_2, w_3) - \text{discount\_1} / f_{\text{context}}(w_2))$
- $p^*(w_3) = f(w_3) / N$  (the lecture slide version)

- $\text{discount\_k} = \delta = \frac{N_1}{N_1 + 2N_2}$

- $f_{\text{context}}(w_{i-k} \dots w_{i-1}) = \sum_x f(w_{i-k}^{i-1}x)$

- $\text{alpha}(\text{context}) =$

$$\alpha(C) = 1 - \sum_{w:f(C,w)>0} \frac{f(C, w) - \delta}{f(C)}$$

note:  $N$  is the same as  $f_{\text{context}}(\cdot)$ .  
It is the freq of the empty context.

$$2. p(\text{Uni} \mid \text{an der}) = p^*(\text{Uni} \mid \text{an, der}) + \alpha(\text{an, der}) [ p^*(\text{Uni} \mid \text{der}) + \alpha(\text{der}) [ p^*(\text{Uni}) ] ]$$

- $p^*(\text{Uni} \mid \text{an, der}) = \max(0, f(\text{an, der, Uni}) - \text{discount\_2}) / f(\text{an, der})$

- $p^*(\text{Uni} \mid \text{der}) = \max(0, f(\text{der, Uni})) - \text{discount\_1} / f(\text{der})$

- $p^*(\text{Uni}) = f(\text{Uni}) / N$  (the lecture slide version)

- the rest is the same as 1.

## f(ngram) and f\_context(context ngram) is not the same

$$1. \ p(w_3 | w_1, w_2) = p^*(w_3 | w_1, w_2) + \text{alpha}(w_1, w_2) [ p^*(w_3 | w_2) + \text{alpha}(w_2) [ p^*(w_3) ] ]$$

- $p^*(w_3 | w_1, w_2) = \max(0, f(w_1, w_2, w_3) - \text{discount\_2} / f_{\text{context}}(w_1, w_2))$
- $p^*(w_3 | w_2) = \max(0, f(w_2, w_3) - \text{discount\_1} / f_{\text{context}}(w_2))$
- $p^*(w_3) = f(w_3) / N$

- To compute the first ngram,  $f(w_1, w_2, w_3)$ , look at the 3-gram frequency table we extracted from the corpus.
- To compute  $f(\text{reduced ngram})$ , we can use either the standard count or the Kneser-Ney count.

Bei der bisherigen Berechnung der  $n-1$ -Gramm-Häufigkeiten zur Schätzung der Backoff-Verteilung summieren wir die Häufigkeiten über alle möglichen Vorgängerwörter  $w'$ :

$$f(C, w) = \sum_{w'} f(w', C, w) \quad \text{standard}$$

C ist eine (eventuell leere) Folge von Wörtern.

Bei Kneser-Ney zählen wir, wieviele **unterschiedliche** Wörter vor dem Wort- $n$ -Gramm aufgetreten sind:

$$f^*(C, w) = \sum_{w'} \mathbf{1}_{f(w', C, w) > 0} \quad \text{Kneser-Ney}$$

$\mathbf{1}_{\text{test}}$  ist 1, falls  $\text{test}$  wahr ist und sonst 0.

Die Kneser-Ney-Methode zählt n-Gramm-Types (statt -Tokens).

### For example

#### - Standard count

$$f(w_2, w_3) = \sum_w f(w, w_2, w_3)$$

If we have the following values in the 3-gram freq table

$f(\text{the}, \text{dog}, \text{barks}) : 3$

$f(\text{the}, \text{dog}, \text{eats}) : 2$

$f(\text{my}, \text{cat}, \text{eats}) : 1$

$f(\text{her}, \text{dog}, \text{barks}) : 5$

and we want to compute  $f(\text{dog}, \text{bargs})$ , we will compute  
 $= f(\text{the}, \text{dog}, \text{barks}) + f(\text{her}, \text{dog}, \text{barks}) = 3 + 5$

#### -Kneser-Ney count

$$f(w_2, w_3) = \sum_w 1 \text{ if } f(w, w_2, w_3) > 0$$

$$= f_{\text{kneser}}(\text{the}, \text{dog}, \text{barks}) + f_{\text{kneser}}(\text{her}, \text{dog}, \text{barks}) = 1 + 1$$

## **f(ngram) and f\_context(context ngram) is not the same**

1.  $p(w_3 | w_1, w_2) = p^*(w_3 | w_1, w_2) + \text{alpha}(w_1, w_2) [ p^*(w_3 | w_2) + \text{alpha}(w_2) [ p^*(w_3) ] ]$

- $p^*(w_3 | w_1, w_2) = \max(0, f(w_1, w_2, w_3) - \text{discount\_2} / f_{\text{context}}(w_1, w_2))$
- $p^*(w_3 | w_2) = \max(0, f(w_2, w_3) - \text{discount\_1} / f_{\text{context}}(w_2))$
- $p^*(w_3) = f(w_3) / N$

**For example**

$$f_{\text{context}}(w_{i-k \dots i-1}) = \sum_x f(w_{i-k}^{i-1} x)$$

$$f_{\text{context}}(w_1, w_2) = \sum_w f(w_1, w_2, w)$$

If we have the following values in the 3-gram freq table

- f(the, dog, barks) : 3
- f(the, dog, eats) : 2
- f(my, cat, eats) : 1
- f(her, dog, barks) : 5

and we want to compute  $f_{\text{context}}(\text{the}, \text{dog})$ , we will compute

$$= f(\text{the}, \text{dog}, \text{barks}) + f(\text{the}, \text{dog}, \text{eats}) = 3 + 2$$

## Reason why using Kneser-Ney method could be beneficial

### Problem:

Im *San Francisco Chronicle* ist das Wortpaar **San Francisco** sehr häufig und die beiden Einzelwörter **San** und **Francisco** sind ähnlich häufig.

Wenn nun das Wort **sunny** weder vor **San** noch vor **Francisco** aufgetaucht ist, würde bei Backoff-Glättung gelten:

$$p(\text{San}|\text{sunny}) = \alpha(\text{sunny}) p(\text{San})$$

$$p(\text{Francisco}|\text{sunny}) = \alpha(\text{sunny}) p(\text{Francisco})$$

Beide Wahrscheinlichkeiten wären also annähernd gleich.

**Aber:**  $p(\text{Francisco}|\text{sunny})$  sollte viel kleiner sein, weil *Francisco* fast nur nach *San* erscheint, während *San* in vielen Kontexten auftaucht.

## Kneser-Ney Backoff-Verteilung

Je mehr Kontexte es gibt, in denen ein Wort-n-Gramm aufgetaucht ist,

- desto eher erscheint es in einem neuen Kontext
- desto größer sollte seine Backoff-Wahrscheinlichkeit sein.

⇒ Wähle die Backoff-Wahrscheinlichkeit proportional zur Zahl der unterschiedlichen Kontexte, in denen das Wort-n-Gramm aufgetreten ist

## Kneser-Ney Backoff-Verteilung

Bei der bisherigen Berechnung der n-1-Gramm-Häufigkeiten zur Schätzung der Backoff-Verteilung summieren wir die Häufigkeiten über alle möglichen Vorgängerwörter  $w'$ :

$$f(C, w) = \sum_{w'} f(w', C, w)$$

$C$  ist eine (eventuell leere) Folge von Wörtern.

Bei Kneser-Ney zählen wir, wieviele **unterschiedliche** Wörter vor dem Wort-n-Gramm aufgetreten sind:

$$f^*(C, w) = \sum_{w'} \mathbf{1}_{f(w', C, w) > 0}$$

$\mathbf{1}_{\text{test}}$  ist 1, falls  $\text{test}$  wahr ist und sonst 0.

Die Kneser-Ney-Methode zählt n-Gramm-Types (statt -Tokens).

Aus den so ermittelten Häufigkeiten, werden dann die Parameter der Backoff-Wahrscheinlichkeits-Verteilungen geschätzt.

$$p_{\text{backoff}}(w|C) = \frac{f^*(C, w)}{\sum_{w'} f^*(C, w')}$$

**Aufgabe 5)** Wie berechnen Sie mit **interpoliertem Backoff** die geglättete Wahrscheinlichkeit  $p(\text{Manchester} \mid \text{Bayern, schlägt})$  aus den Häufigkeiten der Wort-Unigramme (bspw.  $f(\text{Manchester})$ ), -Bigramme (bspw.  $f(\text{schlägt}, \text{Manchester})$ ) und -Trigramme (bspw.  $f(\text{Bayern, schlägt}, \text{Manchester})$ ), sowie den Discount-Werten ( $\delta_1$  etc.) und den Backoff-Faktoren ( $\alpha(\text{Bayern, schlägt})$  etc.)? Bei den Unigrammen sollen die Wahrscheinlichkeiten nicht mehr geglättet werden. (Sie können statt  $\alpha$  auch alpha schreiben, falls Sie eine Textdatei erstellen.)

(3 Punkte)

**Aufgabe 5)** Wie berechnen Sie mit interpoliertem Backoff die geglättete Wahrscheinlichkeit  $p(\text{Manchester}|\text{Bayern}, \text{schlaegt})$  aus den Häufigkeiten der Wort-Unigramme (bspw.  $f(\text{Manchester})$ ), -Bigramme (bspw.  $f(\text{schlägt}, \text{Manchester})$ ) und -Trigramme (bspw.  $f(\text{Bayern}, \text{schlägt}, \text{Manchester})$ ), sowie den Discount-Werten ( $\delta_1$  etc.) und den Backoff-Faktoren ( $\alpha(\text{Bayern}, \text{schlägt})$  etc.)? Bei den Unigrammen sollen die Wahrscheinlichkeiten nicht mehr geglättet werden.

**Antwort:**

$$p(\text{Manchester}|\text{Bayern}, \text{schlaegt}) =$$

$$\frac{f(\text{Bayern}, \text{schlaegt}, \text{Manchester}) - \delta_2}{\sum_w f(\text{Bayern}, \text{schlaegt}, w)} + \alpha(\text{Bayern}, \text{schlaegt}) \cdot \left( \frac{f(\text{schlaegt}, \text{Manchester}) - \delta_1}{\sum_w f(\text{schlaegt}, w)} + \alpha(\text{schlaegt}) \cdot \frac{f(\text{Manchester})}{\sum_w f(w)} \right)$$

it is OK if you don't write  
 $\max(0, \dots)$

this is the same as  
 $f(\text{Manchester}) / N$

this is  $f_{\text{context}}(\text{Bayern}, \text{schaegt})$

**Aufgabe 6)** Erklären Sie in Worten, wie die Discount-Werte  $\delta_i$  und die Backoff-Faktoren  $\alpha(\dots)$  bei der interpolierten Backoff-Glättung berechnet werden.

**Aufgabe 6)** Erklären Sie in Worten, wie die Discount-Werte  $\delta_i$  und die Backoff-Faktoren  $\alpha(\dots)$  bei der interpolierten Backoff-Glättung berechnet werden.

**Antwort:** (Die zugehörige Formel konnte nachgeschlagen werden.)

Zur Berechnung des Discountwertes für die Häufigkeit von n-Grammen zählt man zunächst, wieviele n-Gramm-Types genau einmal ( $N_1$ ) bzw. genau zweimal ( $N_2$ ) aufgetaucht sind. Dann teilt man  $N_1$  durch die Summe aus  $N_1$  und dem Doppelten von  $N_2$ .

Zur Berechnung des Backoff-Faktors  $\alpha(C)$  iteriert man über alle möglichen Vorhersagewerte  $w$ , subtrahiert den Discount von der Häufigkeit des Paars  $C, w$  und teilt das Ergebnis durch die Häufigkeit des Kontextes  $C$ . Die erhaltenen Ergebniswerte werden über alle Vorhersagen  $w$  summiert und dann von 1 subtrahiert.

$$\alpha(C) = 1 - \sum_{w:f(C,w)>0} \frac{f(C, w) - \delta}{f(C)}$$

$$\delta = \frac{N_1}{N_1 + 2N_2}$$

# Beispiel

## Gegebene Häufigkeiten

$$\begin{array}{ll} f(a,a) = 1 & f(b,a) = 0 \\ f(a,b) = 2 & f(b,b) = 1 \end{array}$$

## Discount-Berechnung

$$\begin{array}{ll} N_1 = 2 & \delta = 2 / (2 + 2*1) = 0.5 \\ N_2 = 1 & \end{array}$$

## Berechnung der Backoff-Wahrscheinlichkeitsverteilung

### Standard-Backoff-Verfahren

$$f(a) = f(a,a) + f(b,a) = 1$$

$$f(b) = f(a,b) + f(b,b) = 3$$

$$p(a) = f(a) / (f(a) + f(b)) = 1/4$$

$$p(b) = f(b) / (f(a) + f(b)) = 3/4$$

### Kneser-Ney-Verfahren

$$f^*(a) = |\{(a, a)\}| = 1$$

$$f^*(b) = |\{(a, b), (b, b)\}| = 2$$

$$p(a) = f^*(a) / (f^*(a) + f^*(b)) = 1/3$$

$$p(b) = f^*(b) / (f^*(a) + f^*(b)) = 2/3$$

## Berechnung der Backoff-Faktoren (interpolierte Glättung)

$$\alpha(a) = 1 - r(a|a) - r(b|a) = 1 - 1/6 - 3/6 = 1/3$$

$$\alpha(b) = 1 - r(a|b) - r(b|b) = 1 - 0 - 0.5 = 1/2$$

## Berechnung der geglätteten Wahrscheinlichkeiten

$$p(a|a) = r(a|a) + \alpha(a) \quad p(a) = 1/6 + 1/3 * 1/3 = 5/18$$

$$p(b|a) = r(b|a) + \alpha(a) \quad p(b) = 3/6 + 1/3 * 2/3 = 13/18$$

$$p(a|b) = r(a|b) + \alpha(b) \quad p(a) = 0 + 1/2 * 1/3 = 1/6$$

$$p(b|b) = r(b|b) + \alpha(b) \quad p(b) = 1/2 + 1/2 * 2/3 = 5/6$$

## Berechnung der relativen Häufigkeiten mit Discount (Standard)

$$r(a|a) = \max(f(a,a)-\delta)/(f(a,a)+f(a,b)) = \max(0,(1-0.5))/(1+2) = 1/6$$

$$r(b|a) = \max(f(a,b)-\delta)/(f(a,a)+f(a,b)) = \max(0,(2-0.5))/(1+2) = 3/6$$

$$r(a|b) = \max(f(b,a)-\delta)/(f(b,a)+f(b,b)) = \max(0,(0-0.5))/(0+1) = 0$$

$$r(b|b) = \max(f(b,b)-\delta)/(f(b,a)+f(b,b)) = \max(0,(1-0.5))/(0+1) = 0.5$$

**Example:** we want to compute  $p(b|a) = r(b|a) + \alpha(a) p(b)$

### Backoff-Glättung mit Interpolation

$$p(w_i | w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1} x)} + \alpha(w_{i-k}^{i-1}) p(w_i | w_{i-k+1}^{i-1})$$

here we call this part  $r(w_i | w_{i-k} \dots w_{i-1})$

### Gegebene Häufigkeiten

|              |              |
|--------------|--------------|
| $f(a,a) = 1$ | $f(b,a) = 0$ |
| $f(a,b) = 2$ | $f(b,b) = 1$ |

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

$$= f(a,b) - \text{discount} / f(a)$$

this is the last backoff prob,  $p(\text{unigram})$ .

In this example, we will define it as follows (same as Slide 92).

$$p(b) = f(b) / f(\cdot)$$

- This is simply the relative freq but without discount
- $f(\cdot)$  is the same as  $N$

$$1 - [r(b|a) + r(\dots|a) + r(\dots|a) + \dots]$$

$$\alpha(C) = 1 - \sum_{w:f(C,w)>0} \frac{f(C,w) - \delta}{f(C)}$$

this is  $r(\text{any}_w | a)$

= „the relative frequency with discount“ of the ngrams(here bi-gram) that has  $a$  as context

## Backoff-Glättung mit Interpolation

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

$$= f(a,b) - \text{discount} / f(a)$$

$$= 2 - \text{discount} / f(a)$$

$$= 2 - 0.5 / f(a)$$

$$= 2 - 0.5 / 3$$

$$= 1.5/3 = 1 / 2$$

**Gegebene Häufigkeiten**

$$\begin{array}{ll} f(a,a) = 1 & f(b,a) = 0 \\ \underline{f(a,b) = 2} & f(b,b) = 1 \end{array}$$

**Discount-Berechnung**

$$\delta = \frac{N_1}{N_1 + 2N_2} \quad N_1 = 2 \quad \delta = 2 / (2 + 2*1) = 0.5 \\ N_2 = 1$$

$$\sum_x f(w_{i-k}^{i-1}x) \quad f(\underline{a},\underline{a}) + f(\underline{a},b) = 1+2$$

note: this is  $f_{\text{context}}$

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

**Berechnung der Backoff-Faktoren (interpolierte Glättung)**

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

$\frac{1}{2}$

$$\alpha(C) = 1 - \sum_{w:f(C,w)>0} \frac{f(C,w) - \delta}{f(C)}$$

this is  $r$  of every bigram that has context  $a$

$$r(a|a) = \max(f(a,a) - \delta) / (f(a,a) + f(a,b)) = \max(0, (1-0.5)) / (1+2) = 1/6$$

$$r(b|a) = \max(f(a,b) - \delta) / (f(a,a) + f(a,b)) = \max(0, (2-0.5)) / (1+2) = 3/6$$

$$r(a|b) = \max(f(b,a) - \delta) / (f(b,a) + f(b,b)) = \max(0, (0-0.5)) / (0+1) = 0$$

$$r(b|b) = \max(f(b,b) - \delta) / (f(b,a) + f(b,b)) = \max(0, (1-0.5)) / (0+1) = 0.5$$

$$\alpha(a) = 1 - r(a|a) - r(b|a) = 1 - 1/6 - 3/6 = \underline{1/3}$$

## Backoff-Glättung mit Interpolation

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

1 / 2                  1/3

$p(b) = f(b) / f(\ )$

standard Backoff count       $p(b) = f(b) / (f(a) + f(b)) = 3/4$

f(b) is the freq of the reduced ngram. It is not computed the same way as f\_context. You can compute it using the standard count or the kneser-ney count

Kneser-Ney Backoff count       $\underline{p(b) = f^*(b) / (f^*(a) + f^*(b)) = 2/3}$

## Gegebene Häufigkeiten

|              |              |
|--------------|--------------|
| $f(a,a) = 1$ | $f(b,a) = 0$ |
| $f(a,b) = 2$ | $f(b,b) = 1$ |

## Berechnung der Backoff-Wahrscheinlichkeitsverteilung

Standard-Backoff-Verfahren

$$\underline{f(a) = f(a,a) + f(b,a) = 1}$$

$$f(b) = f(a,b) + f(b,b) = 3$$

Kneser-Ney-Verfahren

$$\underline{f^*(a) = |\{(a, a)\}| = 1}$$

$$\underline{f^*(b) = |\{(a, b), (b, b)\}| = 2}$$

$f_{\text{context}}(\ )$  is the freq of empty context.  
If you use the standard counting method, it is simply the sum of the freq of every unigram or N

$$\sum_x f(w_{i-k}^{i-1}x)$$

If you choose to do Kneser-Ney backoff, then use Kneser-Ney count

Finished

standard backoff

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

$$\begin{matrix} 1/2 & 1/3 & 3/4 \end{matrix}$$

Kneser-Ney backoff

$$p(b|a) = r(b|a) + \alpha(a) p(b)$$

$$\begin{matrix} 1/2 & 1/3 & 2/3 \end{matrix}$$

See the difference between standard count and Kneser-ney

### Berechnung der Backoff-Wahrscheinlichkeitsverteilung

#### Standard-Backoff-Verfahren

$$f(a) = f(a,a) + f(b,a) = 1$$

$$f(b) = f(a,b) + f(b,b) = 3$$

$$p(a) = f(a) / (f(a) + f(b)) = 1/4$$

$$p(b) = f(b) / (f(a) + f(b)) = 3/4$$

#### Kneser-Ney-Verfahren

$$f^*(a) = |\{(a, a)\}| = 1$$

$$f^*(b) = |\{(a, b), (b, b)\}| = 2$$

$$p(a) = f^*(a) / (f^*(a) + f^*(b)) = 1/3$$

$$p(b) = f^*(b) / (f^*(a) + f^*(b)) = 2/3$$

#### Gegebene Häufigkeiten

$$f(a,a) = 1 \quad f(b,a) = 0$$

$$f(a,b) = 2 \quad f(b,b) = 1$$

Bei der bisherigen Berechnung der n-1-Gramm-Häufigkeiten zur Schätzung der Backoff-Verteilung summieren wir die Häufigkeiten über alle möglichen Vorgängerwörter  $w'$ :

$$\frac{f(C, w)}{f(b)} = \sum_{w'} f(w', C, w)$$

C ist eine (eventuell leere) Folge von Wörtern.

Bei Kneser-Ney zählen wir, wieviele **unterschiedliche** Wörter vor dem Wort-n-Gramm aufgetreten sind:

$$f^*(C, w) = \sum_{w'} \mathbf{1}_{f(w', C, w) > 0}$$

$\mathbf{1}_{test}$  ist 1, falls  $test$  wahr ist und sonst 0.

$$f^*(b)$$

$$1(a, b) + 1(b, b) = 1+1 = 2$$

note: we count only the bigram that do not have zero freq

Die Kneser-Ney-Methode zählt n-Gramm-Types (statt -Tokens).

# Musterlösung Übung4

## train-ngram-model.py

```
3 import sys
4 from collections import defaultdict
5 import json
6
7
8 if len(sys.argv) != 4:
9     sys.exit('Usage: '+sys.argv[0]+' n-gram-length text-file model-file')
10
11 # Länge der verwendeten N-Gramme
12 L = int(sys.argv[1])
13
14 # Trainingstext aus Datei einlesen
15 with open(sys.argv[2]) as file:
16     text = file.read()
17
18 # N-Gramm-Häufigkeiten berechnen
19 ngramfreq = defaultdict(int)
20 for i in range(len(text) - L+1):
21     ngram = text[i:i+L]
22     ngramfreq[ngram] += 1
23
```

Suppose we choose the n-gram length to be 3.

In this part, we read the training corpus and create 3 grams frequency table (dictionary)

3 grams

the dog eats : 1

the cat eats : 2

my cat eats : 3

the cat sleeps : 1

her dog eats : 2

```

23
24 # Wahrscheinlichkeiten für alle NGramm-Längen berechnen
25 prob = {}
26 for _ in range(L):
27
28     # Berechnung des Discounts
29     N1 = sum([1 for v in ngramfreq.values() if v == 1])
30     N2 = sum([1 for v in ngramfreq.values() if v == 2])
31
32     if N1 > 100:
33         discount = N1 / (N1 + 2.0 * N2)
34     else:
35         # Defaultwert für den Discount verwenden, falls die Zahl
36         # der einmal aufgetretenen N-Gramme zu klein ist.
37         discount = 0.5
38
39     # Berechnung aller Kontexthäufigkeiten
40     contextfreq = defaultdict(int)
41     for ngram, freq in ngramfreq.items():
42         context = ngram[:-1]
43         contextfreq[context] += freq
44
45     # Berechnung der relativen Häufigkeiten mit Discount
46     for ngram, f in ngramfreq.items():
47         context = ngram[:-1]
48         prob[ngram] = (ngramfreq[ngram] - discount) / contextfreq[context]
49
50     # Berechnung der N-1Gramm-Häufigkeiten
51     sngramfreq = defaultdict(int)
52     for ngram, freq in ngramfreq.items():
53         sngramfreq[ngram[1:]] += freq
54     ngramfreq = sngramfreq
55
56 # Parameter in Datei speichern
57 with open(sys.argv[3], "wt") as file:
58     json.dump(prob, file)
59

```

In this for-loop, we will use the 3-gram freq to compute each element of the backoff formula for all ngram lengths (3,2,1).

## Backoff-Glättung mit Interpolation

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

ngramfreq

3 gram

the dog eats : 1  
the cat eats : 2  
my cat eats : 3  
the cat sleeps : 1  
her dog eats : 2

compute the reduced ngram

2 gram

dog eats : 1 + 2  
cat eats : 2 + 3  
cat sleeps : 1

the second loop

context length 1

dog : 1 + 2  
cat: 2 + 3 + 1

compute discount and context freq

context length 2

the dog : 1  
the cat: 2  
my cat: 3  
her dog: 2

compute the relative freq with discount (prob) of 3-gram

the dog eats : 1.43  
the cat eats : 2.3  
my cat eats : 0.3  
the cat sleeps : 1..  
her dog eats : 2..

compute the relative freq with discount (prob) of 2-gram

dog eats : 0.33  
cat eats : 0.0043  
cat sleeps : 0.0001

## guess-language.py

```

9
10 if len(sys.argv) != 2:
11     sys.exit('Usage: '+sys.argv[0]+' text-file\n\nThe models are read from the directory "models".')
12
13 # Einlesen der verschiedenen Sprachmodelle
14 print("loading models...", file=sys.stderr)
15 ngram_prob = {}
16 backoff = {}
17 L = 0 # Maximale N-Gramm-Länge (wird beim Einlesen der Modelle bestimmt)
18 for lang in os.listdir("models"):
19     with open('models/'+lang, 'rt') as file:
20         ngram_prob[lang] = json.load( file )
21
22 # Berechnung der Backoff-Faktoren
23 backoff[lang] = {}
24 for ngram, p in ngram_prob[lang].items():
25     context = ngram[0:-1]
26     backoff[lang][context] = backoff[lang].get(context, 1.0) - p
27     if L < len(ngram):
28         L = len(ngram)
29 print("done", file=sys.stderr)
30
31 # Eingabetext einlesen
32 with open(sys.argv[1]) as file:
33     text = file.read()
34
35 def smoothed_prob( ngram, lang ):
36     if len(ngram) == 0:
37         # Unigrammwahrscheinlichkeiten werden mit einer uniformen Verteilung geglättet
38         return 1.0 / 1000 # Es wird von 1000 möglichen Zeichen ausgegangen
39     context = ngram[0:-1] # Kontext = N-Gramm ohne letztes Zeichen
40     ngram2 = ngram[1:] # Backoff-NGramm = N-Gramm ohne erstes Zeichen
41     p = ngram_prob[lang].get(ngram, 0.0)
42     bof = backoff[lang].get(context, 1.0)
43     bop = smoothed_prob(ngram2, lang)
44     return p + bof * bop
45

```

Compute the backoff factor for every context length

## Backoff-Glättung mit Interpolation

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^i) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1}) p(w_i|w_{i-k+1}^{i-1})$$

Für  $k = 2$  bekommen wir:

$$\begin{aligned} p(\text{Er ölt}) = \\ p(\text{E}|\langle\text{s}\rangle,\langle\text{s}\rangle) p(\text{r}|\langle\text{s}\rangle,\text{E}) p(\text{-}|\text{E},\text{r}) p(\text{o}|\text{r},\text{-}) p(\text{l}|\text{-},\text{o}) p(\text{t}|\text{o},\text{l}) p(\langle/\text{s}\rangle|\text{l},\text{t}) \end{aligned}$$

define a function to compute the smoothed prob

```

45
46 # Am Textanfang Leerzeichen als Kontext für die ersten Buchstaben hinzufügen
47 text_length = len(text)
48 text = ' '*(L-1) + text
49
50 # Berechnung der logarithmierten Wahrscheinlichkeit des Textes für jede Sprache
51 score = {}
52 for lang in ngram_prob:
53     logp = 0.0
54     for i in range(text_length):
55         ngram = text[i:i+L]
56         logp += math.log( smoothed_prob( ngram, lang ), 2.0 )
57     # berechne Crossentropie = negative logarithmierte Wahrscheinlichkeit
58     # geteilt durch Textlänge
59     score[lang] = -logp / text_length
60
61 # Sprachen aufsteigend nach Crossentropie sortiert ausgeben
62 for lang in sorted(score.keys(), key=score.get):
63     print(lang, score[lang], sep="\t")
64

```

For each language candidate, compute the (log) prob of the input text, convert the prob to cross entropy, rank the language from low to high entropy.

Für  $k = 2$  bekommen wir:

$$p(\text{Er ölt}) = \\ p(E|\langle s \rangle, \langle s \rangle) p(r|\langle s \rangle, E) p(_-|E, r) p(\ddot{o}|r, -) p(l|-, \ddot{o}) p(t|\ddot{o}, l) p(\langle /s \rangle|l, t)$$


---

# Naïve-Bayes-Modelle: Wortbedeutungs-Desambiguierung

Er hat ein Konto bei der **Bank** (→ Geldinstitut)

Nach der **Bank** müssen Sie rechts abbiegen (→ Gebäude)

Die **Bank** hat den Kredit bewilligt (→ Geldinstitut, Mitarbeiter?)

Hahn → tap, cock

Karte → ticket, card, map

Bank → bank, bench

Ufer, Bank ← bank

Die Anwendung des Naïve Bayes-Klassifikators ist einfach.

Für ein gegebenes mehrdeutiges Wort  $w$  im Kontext der Wörter  $w_1 \dots w_n$

- berechnet man das Produkt  $p(s) \prod_{i=1}^n p(w_i|s)$  für jede Bedeutung  $s$
- und wählt die Bedeutung mit dem höchsten Wert.

input sentence and the word  
to disambiguate

Er hat ein Konto bei der **Bank**

context words  
(without stopwords)

naive bayes model for  
word sense  
disambiguation

Bank

sense1: Geldinstitut

sense2: Gebäude

sense3: Mitarbeiter

Hahn

sense1: Wasserhahn

sense2: männliches Haushuhn

argmax over the senses  $s$

$$\hat{s} = \arg \max_s p(s|C)$$

$$= \arg \max_s \frac{p(s, C)}{p(C)}$$

$$= \arg \max_s p(s, C)$$

$$p(s, C) = p(s) \prod_{i=1}^{n+1} p(w_i|s)$$

output

sense1 :  
Geldinstitut

s: sense candidate

C: sequence of context words (content words near the target word, Bank)

- This model is a supervised learning algorithm.
- The training corpus could look like below.
- Each word is annotated with POS and the ambiguous words are annotated with their sense.

Er hat ein Konto bei der **Bank** (**Bank:sense1**).

Pro, V, Art, N, Prep, ....

Nach der **Bank**(**Bank: sense2**) müssen Sie rechts abbiegen

Pro, V, Art, N, Prep, ....

Die **Bank**(**Bank:sense3**) hat den Kredit bewilligt

Pro, V, Art, N, Prep, ....

Sie öffnete ein Konto bei der **Bank**(**Bank:sense1**).

Pro, V, Art, N, Prep, ....

Geld bei der **Bank**(**Bank:sense1**) abheben.

Pro, V, Art, N, Prep, ....

Der **Hahn**(**Hahn:sense1**) am Waschbecken im Badezimmer tropft.

..

## Parameterschätzung

Um einen Wortbedeutungsdesambiguierer für ein bestimmtes Wort zu erstellen, braucht man ein Korpus, in dem jedes Vorkommen dieses Wortes manuell mit seiner korrekten Bedeutung annotiert wurde.

Man zählt die Bedeutungen ( $\rightarrow f(s)$ ) und berechnet ihre Apriori-Wahrscheinlichkeit  $p(s)$ :

$$p(s) = \frac{f(s)}{\sum_{s'} f(s')}$$

Dann zählt man alle Wörter, die im Kontext der Bedeutung  $s$  des ambigen Wortes auftauchen, ( $\rightarrow f(w, s)$ ) und berechnet:

$$p(w|s) = \frac{f(w, s)}{\sum_{w'} f(w', s)}$$

$p(w|s)$  muss noch **geglättet** werden, bspw. durch Backoff-Glättung mit der Backoff-Verteilung  $p(w) = f(w)/N$  mit  $f(w) = \sum_{s'} f(w, s')$  und  $N = \sum_{w'} f(w')$ .

### Corpus

Hahn: sense1

Der Hahn am Waschbecken im Badezimmer tropft.

Der Hahn auf dem benachbarten Bauernhof kräht jeden Morgen.

Hahn: sense2

## Word Sense disambiguation model

$$\begin{aligned}\hat{s} &= \arg \max_s p(s|C) \\ &= \arg \max_s \frac{p(s, C)}{p(C)} \\ &= \arg \max_s p(s, C)\end{aligned}$$

we will compute  
this prob using  
naive bayes  
model

## Wortbedeutungs-Desambiguierung mit Naïve Bayes

Der Kontext  $C$  wird durch eine Menge von Kontextwörtern  $= w_1 \dots w_n$  repräsentiert (z.B. alle Inhaltswörter, die höchstens 50 Positionen entfernt sind).

$$p(s, C) = p(s, w_1 \dots w_n) = p(s) \prod_{i=1}^{n+1} p(w_i | s, w_1 \dots w_{i-1})$$

Auch hier fügen wir ein Endesymbol hinzu.

Zur Vereinfachung des Modells wird angenommen, dass die Kontextwörter statistisch unabhängig sind, wenn die Bedeutung  $s$  gegeben ist:

$$p(s, C) = p(s) \prod_{i=1}^{n+1} p(w_i | s)$$

```

graph LR
    C[Wasserhahn] -- "p(Wasserhahn)" --> Waschbecken((p(Waschbecken|Wasserhahn)))
    C -- "p(Wasserhahn)" --> Badezimmer((p(Badezimmer|Wasserhahn)))
    C -- "p(Wasserhahn)" --> tropft((p(tropft|Wasserhahn)))
  
```

Damit erhalten wir den **Naïve Bayes-Klassifikator**:

$$\hat{s} = \arg \max_s p(s) \prod_{i=1}^{n+1} p(w_i | s)$$

Suppose we have the following text as a new input and want to compute the prob the sense candidate „Wasserhahn“ given this text.

**Text:** Der Hahn am Waschbecken im Badezimmer tropft.

**Klasse:** Wasserhahn

The prob is defined according to the model as follows

$$\begin{aligned}p(\text{Wasserhahn}, \text{Waschbecken}, \text{Badezimmer}, \text{tropft}) &= p(\text{Wasserhahn}) * \\ p(\text{Waschbecken}|\text{Wasserhahn}) * p(\text{Badezimmer}|\text{Wasserhahn}) * p(\text{tropft}|\text{Wasserhahn}) * \\ p(\langle /s \rangle|\text{Wasserhahn})\end{aligned}$$

Here is how to compute each component

$$p(s_1, w_1, w_2, w_3) = p(s_1) * p(w_1 | s_1) * p(w_2 | s_1) * p(w_3 | s_1)$$

$$p(s) = \frac{f(s)}{\sum_{s'} f(s')}$$

$$p(w | s) = \frac{f(w, s)}{\sum_{w'} f(w', s)}$$

To get these frequencies, we look in the training corpus.

$$p(s_1 | w_1, w_2, w_3) = \frac{p(s_1)}{\sum_{s'} p(s')} * p(w_1 | s_1) * p(w_2 | s_1) * p(w_3 | s_1)$$

$$p(s) = \frac{f(s)}{\sum_{s'} f(s')}$$

$p(\text{Bank1}) = f(\text{Bank1}) / f(\text{Bank1}) + f(\text{Bank2}) + f(\text{Bank3})$   
 $p(\text{Bank2}) = f(\text{Bank2}) / f(\text{Bank1}) + f(\text{Bank2}) + f(\text{Bank3})$   
(if Bank has 3 senses)

$$p(\text{Hahn1}) = f(\text{Hahn1}) / f(\text{Hahn1}) + f(\text{Hahn2})$$

$$p(\text{Hahn2}) = f(\text{Hahn2}) / f(\text{Hahn1}) + f(\text{Hahn2})$$

$$p(s_1 | w_1, w_2, w_3) = \frac{p(s_1)}{\sum_{s'} p(s')} * p(w_1 | s_1) * p(w_2 | s_1) * p(w_3 | s_1)$$

$$p(w | s) = \frac{f(w, s)}{\sum_{w'} f(w', s)}$$

e.g.  $p(\text{rechts} | \text{Bank1}) = f(\text{rechts}, \text{Bank1}) /$   
 $f(\text{rechts}, \text{Bank1}) f(\text{Konto}, \text{Bank1}) f(\text{öffnete}, \text{Bank1}) \dots$

$p(w | s)$  muss noch **geglättet** werden, bspw. durch Backoff-Glättung mit der  
Backoff-Verteilung  $p(w) = f(w) / N$  mit  $f(w) = \sum_{s'} f(w, s')$  und  $N = \sum_{w'} f(w')$ .

Er hat ein **Konto** bei der **Bank(Bank:sense1)**

Nach der **Bank(Bank: sense2)** müssen **Sie rechts abbiegen**  
Die **Bank(Bank:sense3)** hat den Kredit bewilligt

Sie öffnete ein Konto bei der **Bank(Bank:sense1)**  
Geld bei der **Bank(Bank:sense1)** abheben.

Der **Hahn(Hahn:sense1)** am Waschbecken im Badezimmer tropft.  
..



$f(s)$                    $f(w, s)$

|       |           |
|-------|-----------|
| Bank1 | w1, Bank1 |
| bank2 | w2, Bank1 |
| bank3 | w3, Bank1 |
| Hahn1 | ..        |
| hahn2 | w1, Bank2 |
| ...   | ...       |

The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word. This is done by looking at the context of the word's use.

This is how the problem has normally been construed in the word sense disambiguation literature. A word is assumed to have a finite number of discrete senses, often given by a dictionary, thesaurus, or other reference source, and the task of the program is to make a forced choice between these senses for the meaning of each usage of an ambiguous word, based on the context of use. However, it is important to realize at the outset that there are a number of reasons to be quite unhappy with such a statement of the task. The word *bank* is perhaps the most famous example of an ambiguous word, but it is really quite atypical. A more typical situation

You can find the book in  
the Tutorium page

## 7.1 Methodological Preliminaries

Several important methodological issues come up in the context of word sense disambiguation. They are of general relevance to NLP, but have received special attention in this context. These are: supervised vs. unsupervised learning; the use of artificial evaluation data, known in the word sense disambiguation context as *pseudowords*; and the development of upper and lower bounds for the performance of algorithms, so that their success can be meaningfully interpreted.

## 7.1.2 Pseudowords

PSEUDOWORDS

In order to test the performance of disambiguation algorithms on a natural ambiguous word, a large number of occurrences has to be disambiguated by hand – a time-intensive and laborious task. In cases like this in which test data are hard to come by, it is often convenient to generate artificial evaluation data for the comparison and improvement of text processing algorithms. In the case of word sense disambiguation these artificial data are called *pseudowords*.

Gale et al. (1992e) and Schütze (1992a) show how pseudowords, i.e., artificial ambiguous words, can be created by conflating two or more natural words. For example, to create the pseudoword *banana-door*, one replaces all occurrences of *banana* and *door* in a corpus by the artificial word ***banana-door***. Pseudowords make it easy to create large-scale training and test sets for disambiguation while obviating the need for hand-labeling: we regard the text with pseudowords as the ambiguous source text, and the original as the text with the ambiguous words disambiguated.

## 7.2 Supervised Disambiguation

In supervised disambiguation, a disambiguated corpus is available for training. There is a training set of exemplars where each occurrence of the ambiguous word  $w$  is annotated with a semantic label (usually its contextually appropriate sense  $s_k$ ). This setting makes supervised disambiguation an instance of statistical classification, the topic of chapter 16. The task is to build a classifier which correctly classifies new cases based on their context of use  $c_i$ . This notation, which we will use throughout the remainder of the chapter, is shown in table 7.1.

We have selected two of the many supervised algorithms that have been applied to word sense disambiguation that exemplify two important theoretical approaches in statistical language processing: Bayesian classification (the algorithm proposed by Gale et al. (1992b)) and Information Theory (the algorithm proposed by Brown et al. (1991b)). They also demonstrate that very different sources of information can be employed successfully for disambiguation. The first approach treats the context of occurrence as a bag of words without structure, but it integrates information from many words in the context window. The second approach looks at only one informative feature in the context, which may be sensitive to text structure. But this feature is carefully selected from a large number of potential ‘informants.’

### 7.2.1 Bayesian classification

The idea of the Bayes classifier which we will present for word senses is that it looks at the words around an ambiguous word in a large context window. Each content word contributes potentially useful information about which sense of the ambiguous word is likely to be used with it. The classifier does no feature selection. Instead it combines the evidence

236

7 Word Sense Disambiguation

BAYES CLASSIFIER  
BAYESDECISIONRULE

(7.2)

Bayes decision rule  
Decides’ if  $P(s'|c) > P(s_k|c)$  for  $s_k \neq s'$

The Bayes decision rule is optimal because it minimizes the probability

The Bayes decision rule is optimal because it minimizes the probability of error. This is true because for each individual case it chooses the class (or sense) with the highest conditional probability and hence the smallest error rate. The error rate for a sequence of decisions (for example, disambiguating all instances of  $w$  in a multi-page text) will therefore also be as small as possible.

**BAYES' RULE** We usually do not know the value of  $P(s_k|c)$ , but we can compute it using *Bayes' rule* as in section 2.1.10:

$$P(s_k|c) = \frac{P(c|s_k)}{P(c)} P(s_k)$$

**PRIOR PROBABILITY**  $P(s_k)$  is the *prior probability* of sense  $s_k$ , the probability that we have an instance of  $s_k$  if we do not know anything about the context.  $P(s_k)$  is updated with the factor  $\frac{P(c|s_k)}{P(c)}$  which incorporates the evidence which we have about the context, and results in the *posterior probability*  $P(s_k|c)$ .

If all we want to do is choose the correct class, we can simplify the classification task by eliminating  $P(c)$  (which is a constant for all senses and hence does not influence what the maximum is). We can also use logs of probabilities to make the computation simpler. Then, we want to assign  $w$  to the sense  $s'$  where:

$$\begin{aligned} (7.3) \quad s' &= \arg \max_{s_k} P(s_k|c) \\ &= \arg \max_{s_k} \frac{P(c|s_k)}{P(c)} P(s_k) \\ &= \arg \max_{s_k} P(c|s_k) P(s_k) \\ &= \arg \max_{s_k} [\log P(c|s_k) + \log P(s_k)] \end{aligned}$$

## NAIVE BAYES

### NAIVE BAYES ASSUMPTION

(7.4)

### BAG OF WORDS

Gale et al.'s classifier is an instance of a particular kind of Bayes classifier, the *Naive Bayes* classifier. Naive Bayes is widely used in machine learning due to its efficiency and its ability to combine evidence from a large number of features (Mitchell 1997: ch. 6). It is applicable if the state of the world that we base our classification on is described as a series of attributes. In our case, we describe the context of  $w$  in terms of the words  $v_j$  that occur in the context.

The *Naive Bayes assumption* is that the attributes used for description are all conditionally independent:

### Naive Bayes assumption

$$P(c|s_k) = P(\{v_j | v_j \text{ in } c\} | s_k) = \prod_{v_j \in c} P(v_j | s_k)$$

In our case, the Naive Bayes assumption has two consequences. The first is that all the structure and linear ordering of words within the context is ignored. This is often referred to as a bag of words model.<sup>3</sup> The other is that the presence of one word in the bag is independent of another. This is clearly not true. For example, *president* is more likely to occur in a context that contains *election* than in a context that contains *poet*. But, as in many other cases, the simplifying assumption makes it possible to adopt an elegant model that can be quite effective despite its shortcomings. Obviously, the Naive Bayes assumption is inappropriate if there are strong conditional dependencies between attributes. But

## Pseudowort-Evaluierung

Die manuelle Annotation von Daten ist teuer.

Zur Evaluierung einer Desambiguierungsmethode kann man jedoch ein großes perfekt annotiertes Trainingskorpus erstellen, indem man zwei Wörter zu einem neuen mehrdeutigen "Pseudo"-Wort zusammenfasst:

### **Pseudowort-Evaluierung:**

- Nimm ein großes Textkorpus.
- Wähle zwei Wörter  $w_1$  und  $w_2$  und ersetze alle Vorkommen von  $w_1$  und  $w_2$  durch das Pseudowort  $w_1-w_2$ .
- Behalte das ursprüngliche Wort als „Bedeutungs“-Annotation.
- Teile das Korpus in zwei Teile.
- Trainiere den Desambiguierer auf dem ersten Teil, zwischen den Bedeutungen  $w_1$  und  $w_2$  des Pseudowortes  $w_1-w_2$  zu unterscheiden.
- Evaluiere den Desambiguierer auf dem zweiten Teil durch Vergleich der Klassifikatorausgabe mit dem Originalwort.

# Pseudowort-Evaluierung

## Initial corpus (unannotated)

Ich habe ein Kind  
Ich komme nach Hause  
Er hat Hunger  
Der Bär ist süß  
Sie ist ein Kind  
Als Staat bezeichnet man in der Volkswirtschaftslehre  
Der Staat als Arbeitgeber  
Die EU ist ein Staat

Suppose we pick „Kind“ and „Staat“ to build the pseudo word, we will adjust the corpus as follows.

Ich habe ein **Kind:Staat**  
Ich komme nach Hause  
Er hat Hunger  
Der Bär ist süß  
Sie ist ein **Kind:Staat**  
Als **Kind:Staat** bezeichnet man in der Volkswirtschaftslehre  
Der **Kind:Staat** als Arbeitgeber  
Die EU ist ein **Kind:Staat**

**Kind:Staat** represents an ambiguous word that can have 2 meanings, namely, **Staat** and **Kind**.

Ich habe ein **Kind:Staat (sense: Kind)**  
Ich komme nach Hause  
Er hat Hunger  
Der Bär ist süß  
Sie ist ein **Kind:Staat (sense: Kind)**  
Als **Kind:Staat (sense: Staat)** bezeichnet man in der Volkswirtschaftslehre  
Der **Kind:Staat (sense: Staat)** als Arbeitgeber  
Die EU ist ein **Kind:Staat (sense: Staat)**

We finally get an (artificial) annotated corpus created to train only one ambiguous word „**Kind:Staat**“.

We can use this corpus to train and evaluate different word sense disambiguation models to see which model works best.

# Exams

**Aufgabe 7)** Bei der Wortbedeutungsdesambiguierung geht es darum, die wahrscheinlichste Bedeutung  $s$  eines ambigen Wortes  $w$  auf Basis der Kontextwörter  $C = w_1^n = w_1, \dots, w_n$  zu berechnen, also

$$\arg \max_s p(s|w_1^n)$$

Zeigen Sie, wie man durch Umformen dieses Ausdrucks die Naive-Bayes-Formel herleitet:

$$\arg \max_s p(s) \prod_{i=1}^n p(w_i|s)$$

Welche vereinfachenden Annahmen müssen Sie dabei machen? (2 Punkte)

**Aufgabe 7)** Bei der Wortbedeutungsdesambiguierung geht es darum, die wahrscheinlichste Bedeutung  $s$  eines ambigen Wortes  $w$  auf Basis der Kontextwörter  $C = w_1^n = w_1, \dots, w_n$  zu berechnen, also

$$\arg \max_s p(s|w_1^n)$$

Zeigen Sie, wie man durch Umformen dieses Ausdrucks die Naive-Bayes-Formel herleitet:

$$\arg \max_s p(s) \prod_{i=1}^n p(w_i|s)$$

Welche vereinfachenden Annahmen müssen Sie dabei machen? (2 Punkte)

$$\begin{aligned}\hat{s} &= \arg \max_s p(s|C) \\ &= \arg \max_s \frac{p(s, C)}{p(C)} \\ &= \arg \max_s p(s, C)\end{aligned}$$

$$p(s, C) = p(s, w_1 \dots w_n) = p(s) \prod_{i=1}^{n+1} p(w_i|s, w_1 \dots w_{i-1})$$

Zur Vereinfachung des Modells wird angenommen, dass die Kontextwörter statistisch unabhängig sind, wenn die Bedeutung  $s$  gegeben ist:

$$p(s, C) = p(s) \prod_{i=1}^{n+1} p(w_i|s)$$

Damit erhalten wir den **Naïve Bayes-Klassifikator**:

$$\hat{s} = \arg \max_s p(s) \prod_{i=1}^{n+1} p(w_i|s)$$

**Aufgabe 8)** Ein Naive-Bayes-Modell soll benutzt werden, um einen Zeitungsartikel einem Themengebiet (Politik, Wirtschaft etc.) zuzuweisen. Wie trainieren Sie das Modell? Welche Art von Daten benötigen Sie dazu? Wie berechnen Sie die wahrscheinlichste Kategorie eines Artikels (mit Formel)?

Anmerkung: Wir haben diese Anwendung nicht in der Vorlesung besprochen. Sie müssen hier Ihr Wissen auf eine neue Anwendung übertragen. (4 Punkte)

$$\begin{aligned}\hat{s} &= \arg \max_s p(s|C) \\ &= \arg \max_s \frac{p(s, C)}{p(C)} \\ &= \arg \max_s p(s, C)\end{aligned}$$

- **Daten:** Wir brauchen einen Textkorpus von Zeitungsartikeln, annotiert mit Themengebiet. Jedes Wort im Artikel soll auch mit POS annotiert wird.
- **Training:**
  - Das Modell hat folgende Formel  $\hat{s} = \arg \max_s p(s) \prod_{i=1}^{n+1} p(w_i|s)$
  - Read the corpus, extract  $f(\text{topic})$ ,  $f(\text{word}, \text{topic})$ . **word** denotes a content word in the articles.
  - We can use POS-tag to determine which word is a content word. E.g. take only verb, noun, adjective.
  - Use these frequencies to estimate the model parameters  $p(\text{topic})$  and  $p(\text{word} | \text{topic})$
- **Application**
  - read input text
  - compute  $p(\text{topic} | w_1, w_2, w_3, \dots) = p(\text{topic}) p(w_1|\text{topic}) p(w_2|\text{topic}), p(w_3|\text{topic}), \dots$  for every topic candidate using the parameters we stored in the model.
    - $w_1, w_2, w_3, \dots$  are the content words in the text.
  - return the topic that has the highest probability.

Übung: Explain how to use Naive Bayes Model for word sense disambiguation

## Übung: Explain how to use Naive Bayes Model for word sense disambiguation

- **Data:** Text corpus where all ambiguous words are annotated with their senses and each word is annotated with POS

$$\hat{s} = \arg \max_s p(s|C)$$

- **Training:**

- The model

$$\hat{s} = \arg \max_s p(s) \prod_{i=1}^{n+1} p(w_i|s)$$

- Read the corpus, extract  $f(\text{sense})$ ,  $f(\text{word}, \text{sense})$ .
  - „word“ is a content word (context) that is located near the ambiguous word. For example, starting with the ambiguous word, we take 50 previous and 50 next content words as context.

- note: „word“ does not include the ambiguous word.
  - We can use POS-tag to determine which word is a content word. E.g. take only verb, noun, adjective.
  - Use these frequencies to estimate the model parameters  $p(s)$  and  $p(w|s)$

- **Application**

- read an input text and the given ambiguous word.
  - compute  $p(\text{sense} | w_1, w_2, w_3, \dots) = p(\text{sense}) p(w_1|\text{sense}) p(w_2|\text{sense}), p(w_3|\text{sense}), \dots$  for every possible sense using the parameters we stored in the model.
    - $w_1, w_2, w_3, \dots$  are the context words in the text
  - return the sense that has the highest probability.

$$= \arg \max_s \frac{p(s, C)}{p(C)}$$

$$= \arg \max_s p(s, C)$$