Natural Language Processing (CS5803)

Lecture 5
(Language Modeling and Smoothing)

Language Modeling

- Assign probability to a text-segment/sentence
- Usage:
- **Spell correction**
  - P(Have you seen this before) vs P(Have you scene this before)
- **Speech Recognition**
  - P(The waiter came running) vs P(The water came running)
- **Evaluating generated text**
  - P(tallest person on earth) vs P(longest person on earth)
- **Matching query to documents**
  - Conditional query-likelihood model. To be discussed later
  - Which document is match according to the query's language model

Language Modeling

- How about these sentences?

- There are amazing ladies an gentlemen in this class
- I just had a delicious tea
- The tail of two cities
- Large winds tonight
- What is the mane of the person?

What probability to compute

- Sentence $W = w_1, w_2, w_3, \ldots, w_n$
- Language Model deals with finding the probability of
  - Observing the entire sentence, i.e. $P(W)$, or
  - Seeing the next word given the previous words, i.e. $P(w_n | w_1, w_2, \ldots, w_{n-1})$
- How are these two connected?
- How to compute $P(w_n | w_1, w_2, \ldots, w_{n-1})$?

How to compute the probability?

- Chain Rule
- $P(w_1, w_2, w_3, \ldots, w_n)$ $=P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1})$

- P("go to school") = P("go")*P("to"|"go")*P("school"|"go", "to")
- How to compute these probabilities?

How to compute the probability?

- Chain Rule
- $P(w_1, w_2, w_3, \ldots, w_n)$
  $= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1})$

- P("go to school") =
  P("go")*P("to"|"go")*P("school"|"go", "to")
- How to compute these probabilities?

How to compute the probability?

w	Go	to	School
P(w)	0.03	0.08	0.01

w_1, w_2	go	to	school
go	10^{-12}	0.004	0.001
to	0.001	10^(-8)	0.002
school	0.0001	0.005	10^(-6)

$(w_1, w_2), w_3$	go	to	school
go, go	10^(-10)	10^(-8)	10^(-9)
go, to	10^(-6)	10^(-6)	0.0006
go, school	10^(-6)	10^(-9)	10^(-14)
to, go	10^(-6)	10^(-5)	10^(-6)
to, to	10^(-6)	10^(-13)	10^(-7)
to, school	10^(-7)	5*10^(-6)	10^(-8)
school, go	10^(-6)	10^(-7)	10^(-7)
school, to	10^(-8)	10^(-9)	2*10^(-6)
school, school	10^(-9)	10^(-7)	10^(-7)

P("go to school") = P("go")*P("to"|"go")*P("school"|"go", "to") = ?

How to find the probabilities?

- Count-based estimates

$P(w_1 w_2)$ = (Count(Number of examples with the sequence $w_1 w_2$))/
(Count(Number of examples with w_1))

- Any issues with this scheme?

How to find the probabilities?

- MLE Estimate:
  - $P_{MLE}(w_i | w_{i-1}) = C(w_{i-1}\ w_i) / C(w_{i-1})$

- Estimate after add-1 smoothing:
  - $P_{Add-1}(w_i | w_{i-1}) = C(w_{i-1}\ w_i) + 1) / (C(w_{i-1})) + |V|)$

- Similarly, another possibility is Add-k smoothing
- How to choose k?

Other Smoothing Techniques

- **Backoff**
  - Use (n-1)-gram statistics, if n-gram statistics is not available
  - Keep doing this recursively
- **Interpolation**
  - Mix different n-gram statistics (probabilities)
  - Context based interpolation weight λs
  - λs are functions of w_{n-2} w_{n-1}

Absolute Discounting

- An experiment conducted by Church and Gale on AP newswire corpus
- Corpus segmented into two parts (seen/training and held-out) - each with 22M words
- From the seen corpus, group together bigrams with different frequencies
- For each group, count the average number of occurrences of those bigrams in the held-out corpus

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute discounting: subtract a fixed (absolute) discount $0<d<1$ from each count

Absolute Discounting

- Good estimates already have high counts, so this discount is negligible
- For smaller counts, anyways the stats are unreliable. Discounting does not matter
- Different ways for fixing d
- One way is to set d = $n_1/(n_1+2n_2)$

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\,v)} + \lambda(w_{i-1})P(w_i)$$

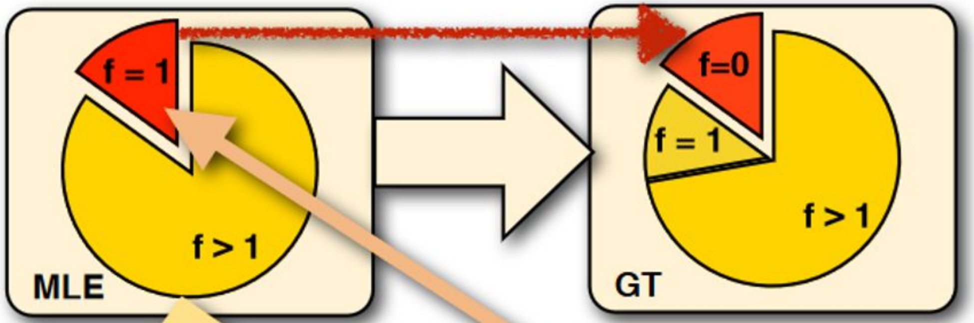
Kneser-Ney discounting

- Similar to absolute discounting
- Only difference in how unigram probability P(w) is estimated
- P(w) is treated as $P_{CONTINUATION}(w)$
- Count the number of contexts (v) in which the unigram (w) occurs
    - Number of different words that precede w in the collection

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|}$$

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\text{CONTINUATION}}(w_i)$$

Good Turing Smoothing



$$
\begin{array}{llcccc}
 & P(seen) & + & P(unseen) & = & 1 \\
\textbf{MLE} & \frac{N}{N} & + & 0 & = & 1 \\
\textbf{Good Turing} & \frac{2 \cdot N_2 + \ldots + m \cdot N_m}{\sum_{i=1}^{m} i \cdot N_i} & + & \frac{1 \cdot N_1}{\sum_{i=1}^{m} i \cdot N_i} & = & \frac{\sum_{i=1}^{m} i \cdot N_i}{\sum_{i=1}^{m} i \cdot N_i}
\end{array}
$$

N_c: number of *event types* that occur c times *(can be counted)*

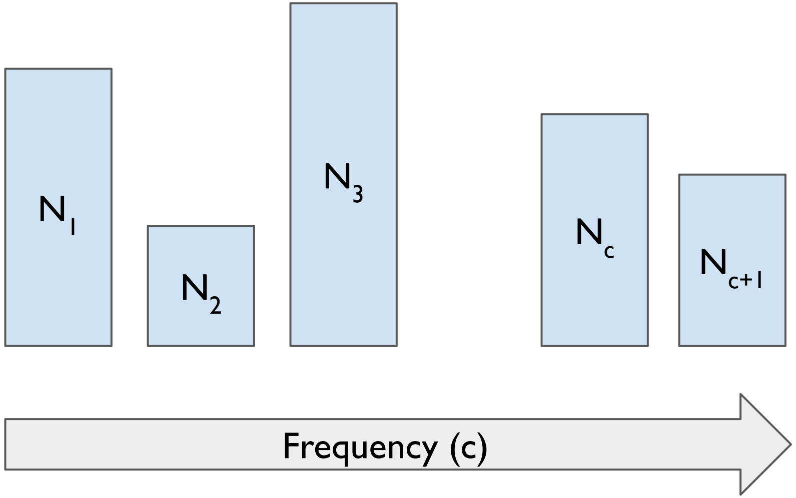
N_1: number of *event types that occur once*

$N = 1N_1 + \ldots + mN_m$: total number of observed event tokens

Idea behind Good Turing Smoothing

- Need to assign probability to unseen words
- **Equivalent:** Need to reserve probability to words with frequency 0
- Bin the words according to frequencies

- Transfer mass from one bin (c+1) to the previous one (c)
- Recalculate effective counts
- Re-estimate probabilities



Good Turing Smoothing

The probability mass of all words that appear k–1 times becomes:

$$\sum_{w:C(w)=k-1} P_{GT}(w) = \sum_{w':C(w')=k} P_{MLE}(w') = \sum_{w':C(w')=k} \frac{k}{N}$$

$$= \frac{k \cdot N_k}{N}$$

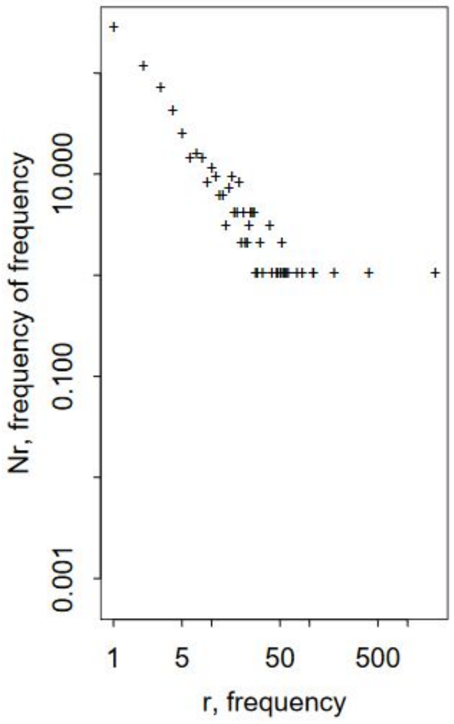
There are $N_{k\text{-}1}$ words w that occur k–1 times in the training data.

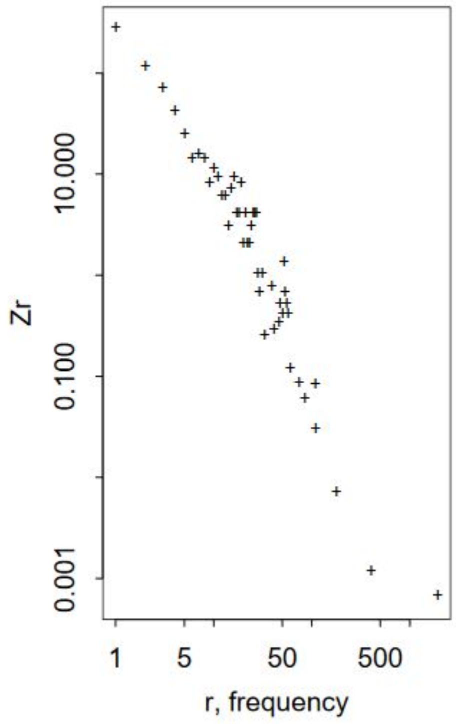
Good-Turing replaces the original count c_{k-1} of w with a new count c^*_{k-1}:

$$c^*_{k-1} = \frac{k \cdot N_k}{N_{k-1}}$$

Simple Good Turing Estimate

frequency r	frequency of frequency N_r
1	268
2	112
3	70
4	41
5	24
6	14
7	15
400	1
1918	1





Simple Good Turing Estimate

- Many gaps (0 words with frequency r) as r increases
- GT estimate becomes useless
- From N_r, get Z_r
  - $Z_r = 2N_r/(t-q)$
- Use the log(Z_r)-vs-r or log(Z_r)-vs-r curve to get N_r intermediate values for r
- log(Nr) = a + b*log(r)
  - Use the fitted value (SGT) for large r
  - GT value for small r