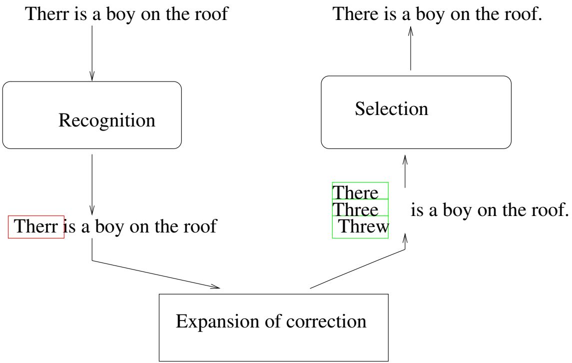
Lecture 3: Markov Models as Language Model

Ivan Titov
ILLC, Universiteit van Amsterdam

April 9, 2014

Recap: Spelling correction



Context-Dependent models

- Typos in text: `Three is a boy on the roof`
- Context of words : *a word is known by the company it keeps*
  Consider whole sentences/ parts of sentences.

How did we approach the problem?

The Noisy-Channel metaphor at sentence level

Sentence level O and C

$$O = o_1, \ldots, o_n \qquad C = w_1, \ldots, w_n$$

Two Models: Task Model and Language Model

$$\arg\max_{w_1^n \in \Gamma} P(w_1^n \mid o_1^n) \;=\; \arg\max_{w_1^n \in \Gamma} P(o_1^n \mid w_1^n)\;\; P(w_1^n)$$

$P(o_1^n \mid w_1^n)$ *Task* Model

- How likely is o_1^n as a result of typos in w_1^n?
- *What plays a role: knowledge of keyboard, knowledge of ins/del/sub/tran!*

$P(w_1^n)$ *Language* Model

- How likely is it that w_1^n is a sentence in the language?
- *Here, knowledge of the language (word order,syntax, semantics can be included..)*

Division of labor: language model and task model

Independence Assumptions (Task model)

Task Model We make the assumption that misspelling in a word is independent of misspellings in other words (a reasonable assumption!)

$$P(o_1^n \mid w_1^n) \approx \prod_{i=1}^{n} P(o_i \mid w_i)$$

This is the *word-level* model.
Single-point transforms (ins / del / sub / tran)

Task model

- Create a training corpus: find mis-spelt text, correct it and keep track of corrections.
- Estimate $P(o_i \mid c_i)$
- How to estimate the probability for every error type?
  - **Example**: what's the probability of spelling w = "problem" as t="oroblem"?
  - "p" occurs 14568 times. It is mis-spelt as "o" 17 times.
  - The probability is

$$\frac{sub(t_n, c_n)}{count(c_n)} = \frac{sub(o, p)}{count(p)} = \frac{17}{14568}$$

  - Similarly, for other 3 misspelling types

Two Models: Task Model and Language Model

$$\arg\max_{w_1^n \in \Gamma} P(w_1^n \mid o_1^n) = \arg\max_{w_1^n \in \Gamma} P(o_1^n \mid w_1^n) \; P(w_1^n)$$

$P(o_1^n \mid w_1^n)$ *Task* Model

- Defined!

$P(w_1^n)$ *Language* Model

- How? We will talk about it today

The Construction of Language Models

Probability of a sentence $w_1 ... w_n$ (Joint probability)

$$
\begin{aligned}
P(w_1, \ldots, w_n) &= P(w_n|w_1 \ldots w_{n-1})\; P(w_1 \ldots w_{n-1}) \\
&= P(w_n|w_1 \ldots w_{n-1})\; P(w_{n-1}|w_1 \ldots w_{n-2})\; P(w_1 \ldots w_{n-2}) \\
&= P(w_1)\; \prod_{i=2}^{n} P(w_i|w_1, \ldots, w_{i-1}) \text{ — from Chain Rule}
\end{aligned}
$$

Example: *a look back at history*

$P(a\ look\ back\ at\ history) =$
$P(a)P(look|a)P(back \mid a\ look)P(at \mid a\ look\ back)P(history \mid a\ look\ back\ at)$

Estimation from Corpora

We want a model of sentence probability $P(w_1, \ldots, w_n)$ for all word sequences $w_1, \ldots, w_n$ over the vocabulary V:

$$P(w_1, \ldots, w_n) = P(w_1) \prod_{i=2}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

Tasks to do:

- Estimate $P(w_1)$
- Estimate probabilities $P(w_i | w_1, \ldots, w_{i-1})$ for all $w_1, \ldots, w_i$!

Estimation from Corpora II

Relative Frequency from a corpus

$$P(w_i \mid w_1, \ldots, w_{i-1}) = \frac{Count(w_1, \ldots, w_{i-1}, w_i)}{\sum_{w \in V} Count(w_1, \ldots, w_{i-1},\ w)}$$

where N is number of all sequences of length i in corpus.

Why is this not a good idea?

Suppose $|V| = 1000$, sentences are ≈ 10 words long:
1000^{10} possible sequences (probability values to estimate): no corpus is large enough!

What to do in order to estimate these probabilities?

- Bucketing histories: Markov models
- Smoothing techniques against sparse-data

Markov Assumption and N-grams

Limited history: There is a fixed finite k such that for all w_1^{i+1}:

$$P(w_{i+1}|w_1, \ldots, w_i) \approx P(w_{i+1}|w_{i-k}, \ldots, w_i)$$

For $k \geq 0$

$$P(w_i \mid w_1, \ldots w_{i-k} \ldots, w_{i-1}) \approx P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

How to estimate the probabilities?

Estimation: k^{th}-order Markov Model

$$P(w_i \mid w_{i-k}, \ldots, w_{i-1}) = \frac{Count(w_{i-k}, \ldots, w_{i-1}, w_i)}{\sum_{w \in V} Count(w_{i-k}, \ldots, w_{i-1},\ w)}$$

Estimation (from a corpus)

$$P(w_i \mid w_{i-k}, \ldots, w_{i-1}) = \frac{Count(w_{i-k}, \ldots, w_{i-1}, w_i)}{\sum_{w \in V} Count(w_{i-k}, \ldots, w_{i-1},\ w)}$$

Addition of START and STOP

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{i=n+1} P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

where $w_j =< s >$ (START) for $j \leq 0$ and $w_{n+1} =< /s >$ (STOP)

$$\sum_{w \in V \cup \{STOP\}} Count(w_{i-k}, \ldots, w_{i-1},\ w) = Count(w_{i-k}, \ldots, w_{i-1})$$

Smoothing

An example corpus:

1. the cat saw the mouse.
2. the cat heard a mouse.
3. the mouse heard.
4. a mouse saw.
5. a cat saw.

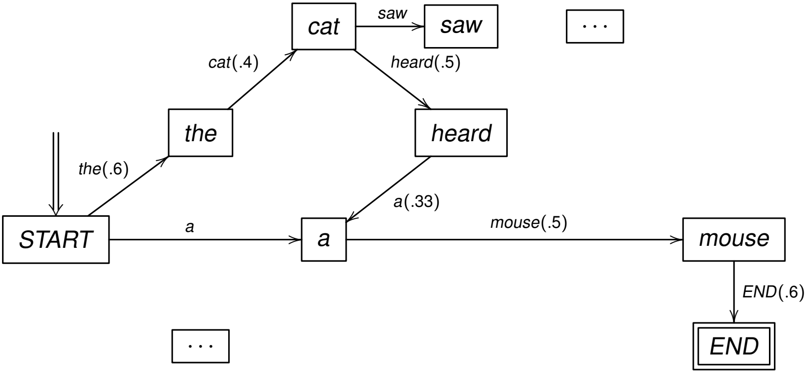
(Langley & Stromsten, 2000)

Bigram model

bigram	count	unigram	count	bigram r.f.
START the	3	START	5	.6
the cat	2	the	5	.4
cat saw	2	cat	4	.5
saw the	1	saw	3	.33
the mouse	2	the	5	.4
mouse END	3	mouse	5	.6
cat heard	2	cat	4	.5
heard a	1	heard	3	.33
a mouse	2	a	4	.5

…

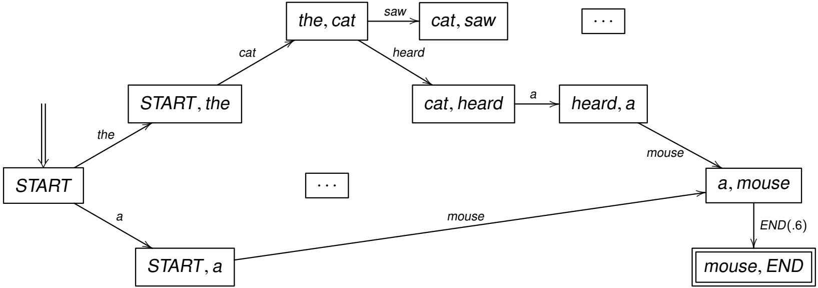
Bigram derivations



Likelihood:

$$P_{bigram}(\text{“ the cat heard a mouse ”}) = .6 \times .4 \times .5 \times .33 \times .5 \times .6 = 0.12$$

Trigram derivations



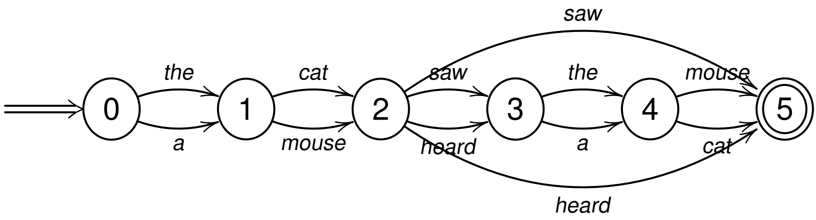
Relation between ngrams and finite-state automata

An ngram model with Markov order $m = n - 1$ is equivalent to an automaton, with for every ngram $g =< w_{i-m}, w_{i-(m-1)}, \ldots, w_i >$

- states defined by all histories $< w_{i-m}, \ldots, w_{i-1} >$
- transition probabilities
  $P(< w_{i-(m-1)}, \ldots, w_i > | < w_{i-m}, \ldots, w_{i-1} >) = P(g)$

Finite-state machines

- More general than ngram models.
- States no longer restricted to histories (sequences of *observable* words, but maybe from any finite set of arbitrary states.)
- Probabilistic FSMs equivalent to Hidden Markov Models.

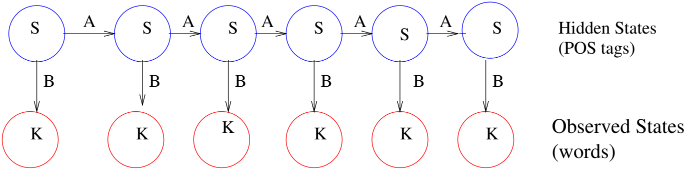


Hidden Markov Models: Concepts

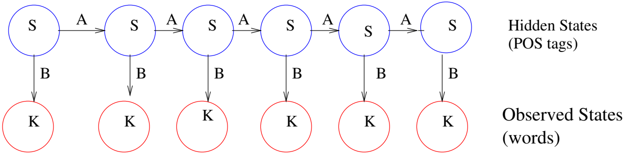
What is an HMM?

Graphical Representation

- Circles indicate states; special state called Start State.
- Arrows indicate probabilistic dependencies between states.
- Top circles are **Hidden** States; bottom circles are **Observed** states.



Hidden Markov Models- 2



Transition prob. For every transition $S_j \rightarrow S_k$, we have a transition Probability $P(S_k|S_j)$ (implementing n-gram probability).

Emission prob: for every state s_k and every word w_i, we have an emission probability $P(w_i \mid s_k)$ (implementing $P(w_i \mid t_i)$).

Constraints: $\forall s_j : \ \sum_{s_i \in S} P(s_i \mid s_j) = 1 \qquad \forall s_k : \ \sum_w P(w \mid s_k) = 1$

We will get back to them during the next lecture!

Smoothing techniques applied to ngram statistics

Today: Smoothing for *n*-grams

- N-gram statistics and Sparse-data problems
- The need for smoothing techniques
- A sketch of smoothing approaches:
  1. Add λ method
  2. Good-Turing method
  3. Discounting: Katz' backoff
  4. Interpolation: Jelinek-Marcer

Reference: Joshua Goodman and Stanly Chen. *An empirical study of smoothing techniques for language modeling*. Tech. report TR-10-98, Harvard University, August 1998.
`http://research.microsoft.com/~joshuago/`

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) = 0$, why?

Data: text does not contain bigrams $\langle START, Jerry \rangle$ and $\langle Jerry, sees \rangle$

Zeros: are called "unseen events",

Question: the above sentence should have a non-zero probability, how do we estimate its probability?

The sparse-data problem

- As n increases, the number of n-grams greater : chance that all 2-grams are present in training corpus is small.
- The probability is smaller for 3-grams, 4 grams ,...!!
- Which *n* yields *n*-grams that are suitable for modeling language? n = 1, 2, 3

Why are zero's a problem for language models?

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)
  - For e.g., in case of *n*-gram models, sparse-data problem is worse as *n* increases.

So: we will need a general solution for this.

N-gram counts and sparsity

A 2^{nd}-order Markov model (trigrams) of $P(w_1, \ldots w_m)$ can be modelled using two tables (actually one):

Trigram	**count**
$< s >$ The boy	Count($< s >$ The boy)
The boy went	Count(The boy went)
boy went home	Count(boy went home)
$\vdots$	$\vdots$

Bigram	**counts**
$< s >$ The	Count($< s >$ The)
The boy	Count(The boy)
boy went	Count(boy went)
$\vdots$	$\vdots$

Expect many zero's in the table

More data: Does that solve the problem completely?

There will always be events that are missing.

Zipf's law: an empirical observation about text, species etc.

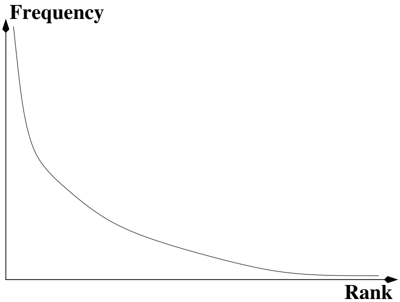
$freq(e)$: the frequency of e in naturally occuring data
$rank(e)$: the rank of e in the list ordered by freq.

There is a constant K such that for all e

$$freq(e) \times rank(e) = K$$

Example: An event of the $100,000^{th}$ rank occurs $10,000$ times less often than an event of the 10^{th} rank, i.e.
$freq(e_{100000}) = \frac{1}{10,000} \times freq(e_{10})$

Zipf's law



- In a given corpus, a few very frequent words, but very many infrequent words.
- Having a resonably sized corpus is important, but we always need smoothing.
- Smoothing technique used has a large effect on the performance of any NLP system.

Smoothing techniques for Markov Models

General Idea

Find a way to fill the gaps in counts of events in corpus C.

- Take care not to change original distribution too much.
- Fill the gaps only as much as needed: as corpus grows larger, there will be less gaps to fill.

Smoothing techniques for Markov Models

General Idea

Adding λ: Assume all events (bigrams) occur λ times more than they occur actually

Discount and redistribute: Reserve probability mass from seen events in order to give to unseen events (discounting).

- How to discount mass in a proper way? (how much is enough)
- How to redistribute mass? (define neighbors)
- How can we combine different model estimates and benefit from the complementary strengths of different models (Interpolation)?

Smoothing is a subject that offers a wide variety of techniques

Naive smoothing: Adding λ method (1)

Events: Set E of possible events, e.g. bigrams over V: $E = (V \times V)$

Data: $e_1, \ldots e_N$ (data size is N events)

Counting: Event (bigram) e occurs $C(e)$ times in Data

Relative Frequency estimate: $P_{rf}(e) = \frac{C(e)}{N}$

Add $0 < \lambda \leq 1$: for all $e \in E$ (bigrams) change $C(e)$ and $P_{rf}(e)$

$$\hat{C}(e) = C(e) + \lambda$$

$$P_{\lambda}(e) = \frac{C(e) + \lambda}{N + \lambda|E|}$$

Add λ method (2)

Example: Bigram Model

$$P_\lambda(w_i|w_{i-1}) = \frac{\lambda + c(w_{i-1}w_i)}{\sum_w(\lambda + c(w_{i-1}, w))}$$

Advantages: very simple and easy to apply

Disadvantges: Method performs poorly (see Chen & Goodman):

- All unseen events receive same probability! Is that OK?
- All events upgraded by λ! Is that OK?

Good-Turing method

Intuition Use the counts of things you have seen *once* to estimate the count of things not seen.
i.e. use n-grams with frequency 1 to re-estimate the frequency of zero counts.

Suppose we have data with total count of events being N:

Standard notation:

r	=	frequency of event e
n_r	=	number of events e with frequency r
		$n_r = \lvert\{e \in E \mid Count(e) = r\}\rvert$
N_r	=	total frequency of events occuring exactly r times
		$N_r = r \times n_r$

Observation: $N = \sum_{r=1}^{\infty} N_r$ $\qquad N_0 = 0$

What we want: To recalculate the frequency r of an event (r^*)

Binomial distribution

- You have a biased coin:
  - The probability of the head (H) is p
  - The probability of the tail (T) is $(1-p)$
- What's the probability of observing *k* heads in *n* tosses $P(k|n)$?

$\mathrm{n} = 2$

H T	$p \times (1-p)$
T H	$p \times (1-p)$
H H	p^2
T T	$(1-p)^2$

$$p(k=0|n=2) = (1-p)^2$$
$$p(k=1|n=2) = 2p(1-p)$$
$$p(k=2|n=2) = p^2$$

Binomial coefficient

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- How do we generalize it?

$$p(k|n) = p^k \times (1-p)^{n-k} \times \ [\text{number of sequences with k heads}]$$

Binomial ngram model

- We have a corpus of *N* ngrams
- Imagine you have ngrams $(\alpha_1, \ldots, \alpha_s)$ with (unknown) probabilities $(p_1, \ldots, p_s)$
- What would be the probability that ngram α_i appears r times?

$$\binom{N}{r} p_i^r (1-p_i)^{(N-r)}$$

- What would be the expected number of ngrams appearing r times?

$$E_N[n_r] = \sum_{i=1}^{s} \binom{N}{r} p_i^r (1-p_i)^{(N-r)}$$

$$E_{N+1}[n_{r+1}] = \sum_{i=1}^{s} \binom{N+1}{r+1} p_i^{r+1} (1-p_i)^{(N-r)} = \sum_{i=1}^{s} \frac{N+1}{r+1} \binom{N}{r} p_i^{r+1} (1-p_i)^{(N-r)}$$

Expectations

- Distribution q and a statistics x

$$q_i = (\quad 1/3, \quad 1/3, \quad 1/3 \quad)$$
$$x_i = (\quad 1, \quad 1, \quad 10 \quad)$$

- Expectation for x under the distribution q

$$E_{[q]}(x) = \sum_i q_i \times x_i \;= \frac{1}{3} \times 1 + \frac{1}{3} \times 1 + \frac{1}{3} \times 10 = 4$$

The same as average for uniform distributions

Expectations

- Distribution q and a statistics x

$$q_i = (\quad 1/20, \quad 1/20, \quad 9/10 \quad)$$
$$x_i = (\quad 1, \quad 1, \quad 10 \quad)$$

- Expectation for x under the distribution q

$$E_{[q]}(x) = \sum_i q_i \times x_i = \frac{1}{20} \times 1 + \frac{1}{20} \times 1 + \frac{9}{10} \times 10 = 9.1$$

- Think of this as of a weighted average, where weights are the specified distribution

Binomial ngram model

- Imagine you have ngrams $(\alpha_1, \ldots, \alpha_s)$ with (unknown) probabilities $(p_1, \ldots, p_s)$
- What is the expected true probability for a ngram α which appears *r* times in a corpus of *N* ngrams?

That's what we need!

$$E_{P(\alpha=\alpha_j | C(\alpha)=r)}(p)$$

Expectation of probability!

Under the distribution of α over $(\alpha_1, \ldots, \alpha_s)$

$$E_{P_{unif}(\alpha=\alpha_j)}(p) = \sum_{j=1}^{s} P_{unif}(\alpha = \alpha_j) \times p_j = \sum_{j=1}^{s} \frac{1}{s} p_j = \frac{\sum_{j=1}^{s} p_j}{s}$$

But that's not what we need!

Binomial ngram model

- Imagine you have ngrams $(\alpha_1, \ldots, \alpha_s)$ with (unknown) probabilities $(p_1, \ldots, p_s)$
- What is the expected true probability for a ngram α which appears *r* times in a corpus of *N* ngrams?

That's what we need!

$$E_{P(\alpha=\alpha_j|C(\alpha)=r)}(p)$$

Expectation of probability!

Under the distribution of α over $(\alpha_1, \ldots, \alpha_s)$

How do we compute this??

$$E_{P(\alpha=\alpha_j|C(\alpha)=r)}(p) = \sum_{j=1}^{s} P(\alpha = \alpha_j|C(\alpha) = r) \times p_j$$

- We know how to compute this:

$$P(\alpha = \alpha_j, C(\alpha) = r) = \binom{N}{r} p_j^r (1 - p_j)^{(N-r)}$$

Recall: $P(x|y) = \frac{P(x,y)}{P(y)} = \frac{P(x,y)}{\sum_{x'} P(x',y)}$

$$P(\alpha = \alpha_j|C(\alpha) = r) = \frac{P(\alpha = \alpha_j, C(\alpha) = r)}{\sum_{j'} P(\alpha = \alpha_{j'}, C(\alpha) = r)} = \frac{p_j^r (1 - p_j)^{(N-r)}}{\sum_{j'} p_{j'}^r (1 - p_{j'})^{(N-r)}}$$

Binomial ngram model

- Imagine you have ngrams $(\alpha_1, \ldots, \alpha_s)$ with (unknown) probabilities $(p_1, \ldots, p_s)$
- What is the expected true probability for a ngram α which appears *r* times in a corpus of *N* ngrams?

$$p_{GT}(\alpha) = \quad E_{P(\alpha=\alpha_j|C(\alpha)=r)}(p) = \sum_{j=1}^{s} P(\alpha = \alpha_j | C(\alpha) = r) \times p_j =$$

$$P(\alpha = \alpha_j | C(\alpha) = r) = \frac{p_j^r (1-p_j)^{(N-r)}}{\sum_{j'} p_{j'}^r (1-p_{j'})^{(N-r)}}$$

$$= \sum_{j=1}^{s} \frac{p_j^r (1-p_j)^{(N-r)}}{\sum_{j'} p_{j'}^r (1-p_{j'})^{(N-r)}} p_j \quad = \frac{\sum_{j=1}^{s} p_j^{r+1} (1-p_j)^{(N-r)}}{\sum_{j'} p_{j'}^r (1-p_{j'})^{(N-r)}}$$

Recall: $E_N[n_r] = \sum_{i=1}^{s} \binom{N}{r} p_i^r (1-p_i)^{(N-r)}$

$$E_{N+1}[n_{r+1}] = \sum_{i=1}^{s} \frac{N+1}{r+1} \binom{N}{r} p_i^{r+1} (1-p_i)^{(N-r)}$$

The number of ngrams appearing (r+1) times

Good-Turing estimate: $p_{GT}(\alpha) = \frac{r+1}{N+1} \frac{E_{N+1}[n_{r+1}]}{E_N[n_r]} \approx \frac{r+1}{N+1} \frac{n_{r+1}}{n_r} \approx \frac{r+1}{N} \frac{n_{r+1}}{n_r}$

Good-Turing Estimates

The Good-Turing probability estimate for events with frequency r:

$$P_{GT}(\alpha) \approx \frac{r+1}{N}\frac{n_{r+1}}{n_r} = \frac{1}{N} \times (r+1) \times \frac{n_{r+1}}{n_r}$$

We can think of this, as assigning frequency of r^* to events appearing r times:

$$r^* = (r+1) \times \frac{n_{r+1}}{n_r}$$

n_r :	number of events with freq. r
n_{r+1} :	number of events with freq. $r+1$

Properties of Good-Turing

Preservation: Total number of counts is preserved:

$$N = \sum_{r=1}^{\infty} r n_r = \sum_{r=0}^{\infty} (r+1) n_{r+1} = \sum_{r=0}^{\infty} n_r r^*$$

Discounting: Total freq. for non-zero events is discounted

$$N_0 = n_0 \times 0^* = n_0 \times (1 \times \frac{n_1}{n_0}) = n_1$$

Zero freq. events

$$P_0 = \frac{r^*}{N} = \frac{0*}{N} = \frac{n_1}{N}$$

Zero events: No explicit method for redistributing N_0 among zero events!

Redistribute the reserved mass (N_0) uniformly among zero events?