Corpora and Statistical Methods – Lecture 8

Albert Gatt

Part 1

Language models continued: smoothing and backoff

Good-Turing Frequency Estimation

Good-Turing method

- Introduced by Good (1953), but partly attributed to Alan Turing
  - work carried out at Bletchley Park during WWII
- "Simple Good-Turing" method (Gale and Sampson 1995)
- Main idea:
  - re-estimate amount of probability mass assigned to low-frequency or zero n-grams based on the number of n-grams (types) with higher frequencies

Rationale

- Given:
  - sample frequency of a type (n-gram, aka bin, aka equivalence class)

- GT provides:
  - an estimate of the true population frequency of a type
  - an estimate of the total probability of unseen types in the population.

Ingredients

- the sample frequency C(x) of an n-gram *x* in a corpus of size N with vocabulary size V

- the no. of n-gram **types** with frequency C, T_c

- C*(x): the estimated true population frequency of an n-gram x with sample frequency C(x)
  - N.B. in a perfect sample, C(x) = C*(x)
  - in real life, C*(x) < C(x) (i.e. sample overestimates the true frequency)

Some background

- Suppose:
  - we had access to the true population probability of our n-grams
  - we compare each occurrence of an n-gram x to a dice-throw: either the n-gram is x or not
  - i.e. a binomial assumption

- Then, we could calculate the expected no of types with frequency C, T_c, i.e the expected frequency of frequency

$$E(T_c) = \sum_{i=1}^{T} \binom{N}{C} (p_i)^C (1 - p_i)^{(N-C)}$$

where:

T_C = no. of n-gram types with frequency f

N = total no. of n-grams

Background continued

- Given an estimate of $E(T_C)$, we could then calculate C*
- Fundamental underlying theorem:

$$C^* = (C+1)\frac{E(T_{C+1})}{E(T_C)}$$

- Note: this makes the "true" frequency C* a function of the expected number of types with frequency C+1. Like Witten-bell, it makes the adjusted count of zero-frequency events dependent on events of frequency 1.

Background continued

- We can use the above to calculate adjusted frequencies directly.
  - Often, though, we want to calculate the total "missing probability mass" for zero-count n-grams (the unseens):

$$P^*_{GT}(\text{n-grams with zero frequency}) = \frac{T_1}{N}$$

Where:

- T1 is the number of types with frequency 1
- N is the total number of items seen in training

Example of readjusted counts

- From: Jurafsky & Martin 2009
- Examples are bigram counts from two corpora.

AP Newswire			Berkeley Restaurant		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

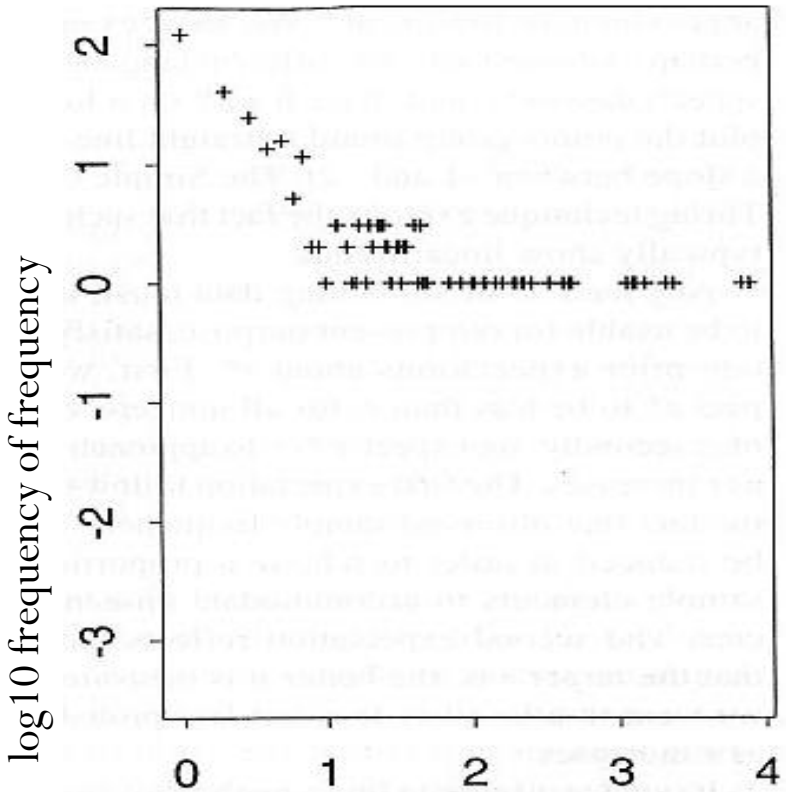
A little problem

$$C^* = (C+1)\frac{E(T_{C+1})}{E(T_C)}$$

- The GT theorem assumes that we know the expected **population** count of types!
  - We've assumed that we get this from a corpus, but this, of course, is not the case.

- Secondly, T_{C+1} will often be zero! For example, it's quite possible to find several n-grams with frequency 100, and no n-grams with frequency 101!
  - Note that this is more typical for high frequencies, than low ones.

Low frequencies and gaps



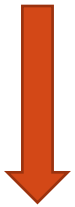
(after Gale and Sampson 1995)

- Low C: linear trend.
- Higher C: angular discontinuity.
- Frequencies in corpus display "jumps" and so do frequencies of frequencies.
- This implies the presence of gaps at higher frequencies.

Possible solution

1. Use Good-Turing for n-grams with corpus frequency less than some constant k (typically, $k = 5$).
   - Low-frequency types are numerous, so GT is reliable.
   - High-frequency types are assumed to be near the "truth".

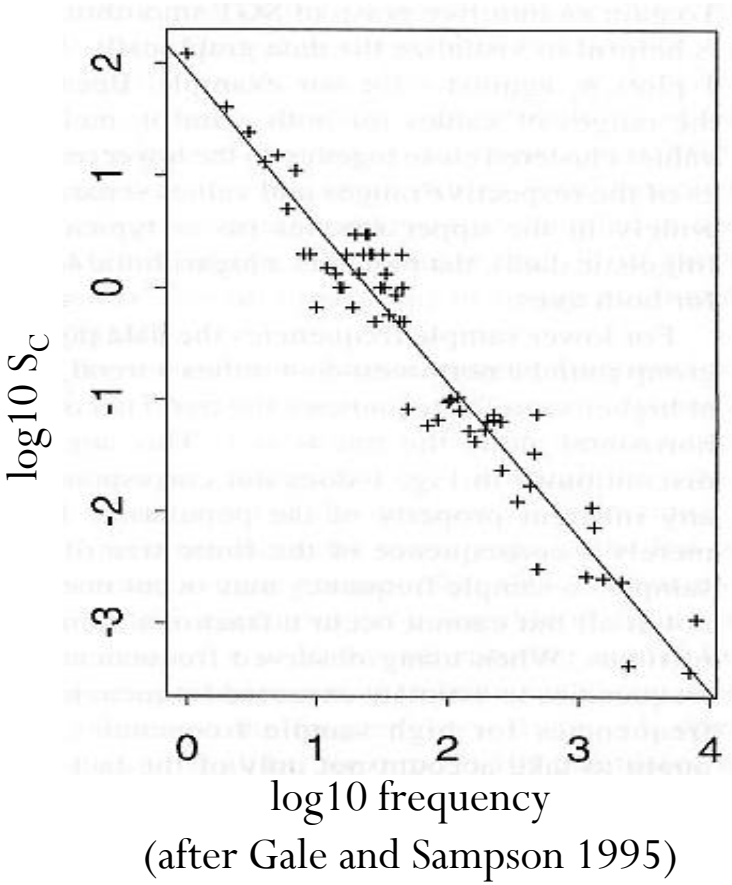
$$C^* = \begin{cases} C \text{ if } C > k \\ (C+1)\dfrac{T_{C+1}}{T_C} \text{ otherwise} \end{cases}$$



2. To avoid gaps (where $T_{c+1} = 0$), empirically estimate a function S(C) that acts as a proxy for $E(T_C)$

$$C^* = (C+1)\frac{S(C+1)}{S(C)}$$

Proxy function for gaps



(after Gale and Sampson 1995)

- For any sample *C*, let:

$$S_C = \frac{2 \times T_f}{C'' - C'}$$

- where:
  - *C*" is the next highest non-zero frequency
  - C' is the previous non-zero frequency

Gale and Sampson's combined proposal

- For low frequencies (< k), use standard equation, assuming $E(T_C) = T_C$

$$C^* = (C+1)\frac{T_{C+1}}{T_C}$$

  - If we have gaps (i.e. $T_C = 0$), we use our proxy function for T_C. Obtained through linear regression to fit the log-log curve

$$C^* = (C+1)\frac{S(C+1)}{S(C)}$$

- And for high frequencies, we can assume that C* = C
- Finally, estimate probability of n-gram:

$$P^*_{GT}(w_1 \ldots w_n) = \frac{C^*(w_1 \ldots w_n)}{N}$$

GT Estimation: Final step

- GT gives approximations to probabilities.
  - Re-estimated probabilities of n-grams won't sum to 1
  - necessary to re-normalise
- Gale/Sampson 1995:

$$p_{GT_{normalised}} = \begin{cases} \frac{N_1}{N} & \text{if } C = 0 \\ \left(1 - \frac{N_1}{N}\right)\frac{C^*}{N} & \text{otherwise} \end{cases}$$

A final word on GT smoothing

- In practice, GT is very seldom used on its own.

- Most frequently, we use GT with **backoff**, about which, more later…

Held-out estimation & cross-validation

Held-out estimation: General idea

- "hold back" some training data
- create our language model
- compare, for each n-gram ($w_1 \ldots w_n$):
  - C_t: estimated frequency of the n-gram based on training data
  - C_h: frequency of the n-gram in the held-out data

Held-out estimation

- Define Tot_C as:
  - total no. of times that *n-grams* with frequency C in the training corpus actually occurred in the held-out data

$$Tot_C = \sum_{\{w_1 \ldots w_n : C_t(w_1 \ldots w_n) = C\}} C_h(w_1 \ldots w_n)$$

- Re-estimate the probability:

$$P_h(w_1 \ldots w_n) = \frac{Tot_C}{T_C N} \text{ if } C(w_1 \ldots w_n) = C$$

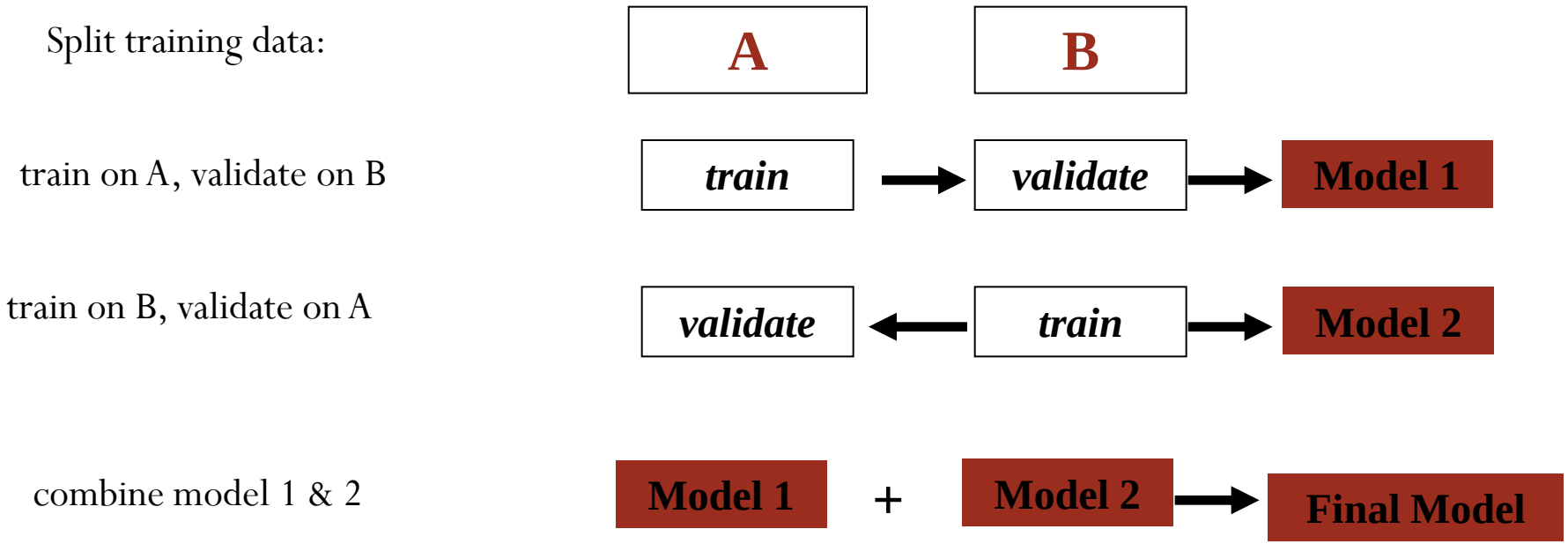
Cross-validation

- Problem with held-out estimation:
  - our training set is smaller

- Way around this:
  - divide training data into training + validation data (roughly equal sizes)
  - use each half first as training then as validation (i.e. train twice)
  - take a mean

Cross-Validation
(a.k.a. deleted estimation)

- Use training and validation data



Cross-Validation

$$P_h = \frac{Tot_C^{AB}}{T_C^A N} \qquad P_h = \frac{Tot_C^{BA}}{T_C^B N}$$

Combined estimate (arithmetic mean):

$$P_{ho} = \frac{Tot_C^{AB} + Tot_C^{BA}}{N(T_C^A + T_C^B)}$$

Combining estimators: backoff and interpolation

The rationale

- We would like to balance between reliability and discrimination:
  - use trigram where useful
  - otherwise back off to bigram, unigram

- How can you develop a model to utilize different length *n-grams* as appropriate?

Interpolation vs. Backoff

- Interpolation: compute probability of an n-gram as a function of:
  - The n-gram itself
  - All lower-order n-grams
  - Probabilities are linearly interpolated.
  - Lower-order n-grams are always used.

- Backoff:
  - If n-gram exists in model, use that
  - Else fall back to lower order n-grams

Simple interpolation: trigram example

- Combine all estimates, weighted by a factor.

$$\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P(w_n \mid w_{n-1}w_{n-2}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n)$$

- All parameters sum to 1: $\sum_i \lambda_i = 1$

- NB: we have different interpolation parameters for the various n-gram sizes.

More sophisticated version

- Suppose we have the trigrams:
  - (*the dog barked*)
  - (*the puppy barked*)
- Suppose (*the* dog) occurs several times in our corpus, but not (*the puppy*)
- In our interpolation, we might want to weight trigrams of the form (*the dog* _) more than (*the puppy* _) (because the former is composed of a more reliable bigram)
- Rather than using the same parameter for all trigrams, we could condition on the initial bigram.

Sophisticated interpolation: trigram example

- Combine all estimates, weighted by factors that depend on the context.

$$\hat{P}(w_3 \mid w_1 w_2) = \lambda_1(w_1 w_2) P(w_3 \mid w_1 w_2) + \lambda_2(w_1 w_2) P(w_3 \mid w_2) + \lambda_3(w_1 w_2) P(w_3)$$

Where do parameters come from?

- Typically:
  - We estimate counts from training data.
  - We estimate parameters from held-out data.
  - The lambdas are chosen so that they maximise the likelihood on the held-out data.

- Often, the expectation maximisation (EM) algorithm is used to discover the right values to plug into the equations.
- (more on this later)

Backoff

- Recall that backoff models only use lower order n-grams when the higher order one is unavailable.

- Best known model by Katz (1987).
  - Uses backoff with smoothed probabilities
  - Smoothed probabilities obtained using Good-Turing estimation.

Backoff: trigram example

- Backoff estimate:

$$P_{katz}(w_3 \mid w_1 w_2) = \begin{cases} P^*(w_3 \mid w_1 w_2) & \text{if } C(w_1 w_2 w_3) > 0 \\ \alpha(w_1 w_2) P_{katz}(w_3 \mid w_2) & \text{if } C(w_1 w_2) > 0 \\ \alpha(w_2) P^*(w_3) & \text{otherwise} \end{cases}$$

- That is:
  - If the trigram has count > 0, we use the smoothed (P*) estimate
  - If not, we recursively back off to lower orders, interpolating with a paramater (alpha)

Backoff vs. Simple smoothing

- With Good-Turing smoothing, we typically end up with the "leftover" probability mass that is distributed equally among the unseens.
  - So GF tells us how much leftover probability there is.

- Backoff gives us a better way of distributing this mass among unseen trigrams, by relying on the counts of their component bigrams and unigrams.
  - So backoff tells us how to divide that leftover probability.

Why we need those alphas

- If we rely on true probabilities, then for a given word and a given n-gram window, the probability of the word sums to 1:

$$\sum_{i,j} P(w_x \mid w_i w_j) = 1$$

- But if we back off to lower-order model when the trigram probability is 0, we're adding extra probability mass, and the sum will now exceed 1.
- We therefore need:
  - *P** to discount the original MLE estimate (P)
  - *Alphas* to ensure that the probability from the lower-order n-grams sums up to exactly the amount we discounted in P*.

Computing the alphas -- I

- Recall: we have $C(w_1 w_2 w_3) = 0$
- Let ß($w_1 w_2$) represent the amount of probability left over when we discount (seen) trigrams containing w_3

$$\beta(w_1 w_2) = 1 - \sum_{w_3 : C(w_1 w_2 w_3) > 0} P^*(w_3 \mid w_1 w_2)$$

The sum of probabilities P for seen trigrams involving w_3 (preceded by any two tokens) is 1. The smoothed probabilities P* sum to less than 1. We're taking the remainder.

Computing the alphas -- II

- We now compute alpha:

$$\alpha(w_1 w_2) = \frac{\beta(w_1 w_2)}{\sum_{w_3 : C(w_1 w_2 w_3)=0} P_{katz}(w_3 \mid w_2)}$$

The denominator sums over all *unseen* trigrams involving our bigram. We distribute the remaining mass ß($w_1 w_2$) overall all those trigrams.

What about unseen bigrams?

- So what happens if even $(w_1 w_2)$ in $(w_1 w_2 w_3)$ has count zero?

$$P_{katz}(w_3 \mid w_1 w_2) = P_{katz}(w_3 \mid w_2) \; if \; C(w_1 w_2) = 0$$

  - I.e. we fall to an even lower order. Moreover:

$$P^*(w_3 \mid w_1 w_2) = 0 \; if \; C(w_1 w_2) = 0$$

  - And:

$$\beta(w_1 w_2) = 1 \; if \; C(w_1 w_2) = 0$$

Problems with Backing-Off

- Suppose (w_2 w_3) is common but trigram (w_1 w_2 w_3) is unseen

- This may be a *meaningful* gap, rather than a gap due to chance and scarce data
  - i.e., a "grammatical null"

- May not want to back-off to lower-order probability
  - in this case, p = 0 is accurate!

References

- Gale, W.A., and Sampson, G. (1995). Good-Turing frequency estimation without tears. *Journal of Quantitative Linguistics,* 2: 217-237