Parameter estimation

- Maximum likelihood (ML) estimates do very badly when faced with sparse data:

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(c|\theta)$$

- Maximum a posteriori (MAP) estimates can improve on that by using prior knowledge:

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(\theta) P(c|\theta)$$

- In general, specifying a prior distribution is hard

- Conjugate priors, MCMC estimation

Parameter estimation

- Uniform prior $\rightarrow$ Laplace's Law

$$P(x) = \frac{C(x) + 1}{N + B}$$

- Dirichlet prior $\rightarrow$ Lidstone's Law

$$P(x) = \frac{C(x) + B\lambda}{N + \lambda}$$

- When data is sparse, the choice of a prior determines the solution (nuisance parameters)

Cross-validation methods

- Divide training data into two parts *a* and *b*
- Suppose an *n*-gram *w* occurs *r* times in part *a*
- Let N_r^a be the total number of *n*-grams that occurred *r* times in part *a*, and T_r^b be the total number of times an *n*-gram which occurred *r* times in part *a* occurs in part *b*
- Held out estimation:

$$P(w) = \frac{T_r^b}{N_r^a N}$$

- Deleted interpolation:

$$P(w) = \frac{T_r^b + T_r^a}{N(N_r^a + N_r^b)}$$

Good-Turing theorem

- Another approach to the problem was taken by Alan Turing and I.J. Good at Bletchley Park (Good 1953)
- Resurrected and simplified by William Gale (AT&T) and colleagues in 1991
- Used in population biology to estimate the number of animals you didn’t see
- Applied to language modeling by IBM speech group
- Related to leave one cross validation

Good-Turing theorem

- Suppose you have a bunch of n-grams $\alpha_1, \ldots, \alpha_s$, which appear with true probability $p_1, \ldots, p_s$. Take an arbitrary n-gram α_i, which occurs r times in a training corpus.

- Find p_i:

$$E[p_i | C(\alpha_i) = r] = \sum_j P(i = j | C(\alpha_i) = r) p_j$$

where:

$$\begin{aligned} P(i = j | C(\alpha_i) = r) &= \frac{P(C(\alpha_j) = r)}{\sum_k P(C(\alpha_k) = r)} \\ &= \frac{\binom{N}{r} p_j^r (1 - p_j)^{N-r}}{\sum_k \binom{N}{r} p_k^r (1 - p_k)^{N-r}} \\ &= \frac{p_j^r (1 - p_j)^{N-r}}{\sum_k p_k^r (1 - p_k)^{N-r}} \end{aligned}$$

Good-Turing theorem

- Substituting, we get:

$$E[p_i|C(\alpha_i) = r] = \sum_j p_j \frac{p_j^r (1-p_j)^{N-r}}{\sum_k p_k^r (1-p_k)^{N-r}}$$

$$= \frac{\sum_j p_j^{r+1} (1-p_j)^{N-r}}{\sum_k p_k^r (1-p_k)^{N-r}}$$

- Now consider $E[N_r]$, the expected number of n-grams which occur exactly N_r times in N words:

$$E[N_r] = \sum_j P(C(\alpha_j) = r)$$

$$= \sum_j \binom{N}{r} p_j^r (1-p_j)^{N-r}$$

Good-Turing theorem

- Substituting again, we get:

$$
\begin{aligned}
E[p_i | C(\alpha_i) = r] &= \frac{E[N_{r+1}] / \binom{N+1}{r+1}}{E[N_r] / \binom{N}{r}} \\
&= \frac{\binom{N}{r}}{\binom{N+1}{r+1}} \frac{E[N_{r+1}]}{E[N_r]} \\
&= \left(\frac{r+1}{N+1} \right) \left(\frac{E[N_{r+1}]}{E[N_r]} \right)
\end{aligned}
$$

- If that's the expected value of p_i, then expected frequency r^* is:

$$
\begin{aligned}
r^* &= N E[p_i | C(\alpha_i) = r] \\
&= \left(\frac{r+1}{1+1/N} \right) \left(\frac{E[N_{r+1}]}{E[N_r]} \right)
\end{aligned}
$$

Good-Turing theorem

- When two independent marginally binomial samples $B_1(N;p_1,\ldots,p_s)$ and $B_2(N;p_1,\ldots,p_s)$ are drawn, the expected frequency r^* in the sample B_2 of types occurring r times in B_1 is:

$$r^* = \left(\frac{r+1}{1+1/N}\right)\left(\frac{E[N_{r+1}|B(N+1;p_1,\ldots,p_s)]}{E[N_r|B(N;p_1,\ldots,p_s)]}\right)$$

  and when N is sufficiently large:

$$r^* \approx (r+1)\,\frac{E[N_{r+1}|B(N;p_1,\ldots,p_s)]}{E[N_r|B(N;p_1,\ldots,p_s)]}$$

  which can be approximated as:

$$r^* \approx (r+1)\,\frac{N_{r+1}}{N_r}$$

Good-Turing smoothing

- We'd like to estimate the adjusted counts using:

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

  but this will be wildly inaccurate for larger values of r

- We can fit a function S to N_r and use those values (Simple Good-Turing Smoothing):

$$r^* = (r+1)\frac{S(N_{r+1})}{S(N_r)}$$

- The total probability assigned to unseen objects is N_1/N (for the Brown corpus example, $335,105/450,852 = 0.74$)

Good-Turing smoothing

- Example from AP newswire corpus:

r	N_r	r^*
0	160,519,590,316	0.0000128
1	2,053,146	0.446
2	458,136	1.26
3	191,809	2.24
4	107,522	3.25
5	69,883	. . .

Good-Turing smoothing

- Example from AP newswire corpus:
- In R:

```
> n <- c(160519590316,2053146,458136,191809,107522,69883)
> r <- 0:4
> (r+1)*(n[r+2]/n[r+1])
[1] 1.279063e-05 4.462771e-01 1.256018e+00 2.242272e+00
[5] 3.249707e+00
```

Smoothing methods

$r = f_{MLE}$	f_{emp}	f_{Lap}	f_{del}	f_{SGT}
0	0.000027	0.000137	0.000037	0.000027
1	0.448	0.000274	0.396	0.446
2	1.25	0.000411	1.24	1.26
3	2.24	0.000548	2.23	2.24
4	3.23	0.000685	3.22	3.24
5	4.21	0.000822	4.22	4.22
6	5.23	0.000959	5.20	5.19
7	6.21	0.00109	6.21	6.21

Combining estimators

- A complementary approach is to use ML estimators when we think they'll be accurate, and something else when we are suspicious of them
- For example, We could use MLE estimates for large r and GT smoothing for small r
- Another possibility is to combine different kinds of history equivalence classes:

$$P(w_3|w_1,w_2) = \lambda_1 P_1(w_3) + \lambda_2 P_2(w_3|w_2) + \lambda_3 P_3(w_3|w_1,w_2)$$

  where the weights λ_i must be non-negative and sum to one

- This is linear interpolation or a mixture model that combines unigram, bigram, and trigram models (*deleted interpolation*)

Backoff models

- Deleted interpolation always uses some information from each model
- Katz's (1987) backoff model uses the best model available:

$$P(w_3|w_1,w_2) = \begin{cases} (1-d(w_1,w_2,w_3))P_3(w_3|w_1,w_2) & \text{if } C(w_1,w_2,w_3) \geq 1 \\ \alpha(w_1,w_2,w_3)P(w_3|w_2) & \text{otherwise} \end{cases}$$

- The discount $d(w_1,w_2,w_3)$ and normalizing factor $\alpha(w_1,w_2,w_3)$ are required to make sure this is a proper distribution
- Discounts can be calculated using Good-Turing frequency estimates, or by other methods

Smoothing

- Many other methods, some ad hoc but useful ($0 \rightarrow \epsilon$)
- Most depend on knowing the number of bins (SGT is a partial exception), and can't distinguish between accidental and 'structural' zeros
- Chen and Goodman (1996) carried out a large-scale empirical comparison of smoothing methods, and found that Simple Good-Turing smoothing worked best with lots of data, otherwise modified Kneser-Ney backoff smoothing