Smoothing

LING 570

Fei Xia

Task #1

- You get parameters
  - $p(heads) = 0.62;\ \ p(tails) = 0.38$
- You want to produce data that conforms to this distribution
- This is ***simulation*** or ***data generation***

Task #2

- You get parameters
  - $p(heads) = 0.1;\ \ p(tails) = 0.9$
- And observations
  - HHHHTHTTHHHHH
- You need to answer: “How likely is this data according to the model?
- This is evaluating the ***likelihood*** function

Task #3

- You get observations:
  - HHTHTTHTHTHHTHTHTTTHTHT
- You need to find a set of parameters:
  - $p(heads) = x, p(tails) = 1 - x$
- This is ***parameter estimation***

Parameter estimation

- We keep talking about things like:
  - $p(x)$ as a distribution with parameters
- How do we estimate parameters?

Observations
HHTHTHTTTHTH
TTTTTTTTTTTTT
HHHTHHHTHHHH

Parameters (Heads, Tails)
(0.50, 0.50)
(0.00, 1.00)
(0.75, 0.25)

Parameter estimation techniques

- Often use Relative Frequency Estimate
  - $P(x) = \frac{c(x)}{N}$

- For certain distributions…
  - "how likely is it that I get k heads when I flip n times" (Binomial distributions)
  - "how likely is it that I get five 6s when I roll five dice" (Multinomial distributions)

- Relative Freq = Maximum Likelihood Estimate (MLE)
  - This is the set of parameters according to which the data has the max likelihood.

Maximum Likelihood has problems

- Remember:

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})}$$

- Two problems:
  - What happens if $c(w_{i-2}w_{i-1}w_i) = 0$?
    - We assign zero probability to an event…
  - Even worse, what if $c(w_{i-2}w_{i-1}) = 0$?
    - Divide by zero is undefined!

Smoothing

- Main goal: prevent zero numerators (zero probs) and zero denominators (divide by zeros)
  - Make a “sharp” distribution (where some outputs have large probabilities and others have zero probs) be “smoother”
  - The smoothest distribution is the uniform distribution

- Constraint:
  - Result should still be a distribution

$$\sum_x p(x) = 1; \quad \forall x. p(x) \geq 0$$

Smoothing

- How?
  - To shave a little bit of probability mass from the higher counts, and pile it instead on the zero counts
  - For the time being, we assume that there are no unknown words; that is, V is a closed vocabulary.

Smoothing methods

- Laplace smoothing (a.k.a. Add-one smoothing)
- Good-Turing Smoothing

- Linear interpolation (a.k.a. Jelinek-Mercer smoothing)
- Katz Backoff

- Class-based smoothing
- Absolute discounting
- Kneser-Ney smoothing

Laplace smoothing

- "Adding one smoothing": Add 1 to all frequency counts.
- Let V be the vocabulary size.

unigram: $$P_{Lap}(w_i) = \frac{1+c(w_i)}{V+N}$$

Bigram: $$P_{Lap}(w_i|w_{i-1}) = \frac{1+c(w_{i-1},w_i)}{V+c(w_{i-1})}$$

n-gram: $$P_{Lap}(w_n|w_1,...,w_{n-1}) = \frac{1+c(w_1,...,w_n)}{V+c(w_1,...,w_{n-1})}$$

Problem with Laplace smoothing

- Example: |V|=100K, a bigram “w1 w2” occurs 10 times, and the trigram ‘w1 w2 w3” occurs 9 times.
  - P_{MLE} (w3 | w1, w2) = 0.9
  - P_{Lap}(w3 | w1, w2) = (9+1)/(10+100K) = 0.0001

- Problem: give too much probability mass to unseen n-grams.

Add-one smoothing does not work well in practice.

Add-δ smoothing

$$P(w_i|w_{i-1}) = \frac{\delta + c(w_i, w_{i-1})}{\delta * V + c(w_{i-1})}$$

Need to choose δ

It works better than add-one, but still very bad.

Good-Turing smoothing

Basic ideas

- Re-estimate the frequency of zero-count N-grams with the number of N-grams that occur once.

- Let N_c be the number of n-grams that occurred c times.

- The Good-Turing estimate for any n-gram that occurs c times, we should pretend that it occurs c* times:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

$$P_{GT}(w_1, ..., w_n) = \frac{c^*(w_1,...,w_n)}{N}$$

For unseen n-grams, we assume that each of them occurs c_0^* times.

$$c_0^* = \frac{N_1}{N_0}$$ N_0 is the number of unseen ngrams

Therefore, the prob of EACH unseen ngram is:

$$P_{GT}(w_1, ..., w_n) = \frac{c^*(w_1,...,w_n)}{N} = \frac{c_0^*}{N} = \frac{N_1}{N_0 * N}$$

The total prob mass for unseen ngrams is: $\frac{N_1}{N}$

Bigram Frequencies of Frequencies and GT Re-estimates

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

N-gram counts to conditional probability

$$P_{GT}(w_i|w_1,...,w_{i-1}) = \frac{c^*(w_1,...,w_{i-1},w_i)}{c^*(w_1,...,w_{i-1})}$$

c* comes from GT estimate.

Backoff and interpolation

N-gram hierarchy

- $P_3(w3|w1,w2)$, $P_2(w3|w2)$, $P_1(w3)$

- Back off to a lower N-gram
  ➔ backoff estimation

- Mix the probability estimates from all the N-grams ➔ interpolation

Katz Backoff

$$P_{katz}(w_i \mid w_{i-1}) = \begin{cases} P_2(w_i \mid w_{i-1}) & \text{if } c(w_{i-1}, w_i) > 0 \\ \alpha(w_{i-1})\, P_1(w_i) & \text{otherwise} \end{cases}$$

$$P_{katz}(w_i \mid w_{i-2}, w_{i-1}) = \begin{cases} P_3(w_i \mid w_{i-2}, w_{i-1}) & \text{if } c(w_{i-2}, w_{i-1}, w_i) > 0 \\ \alpha(w_{i-2}, w_{i-1})\, P_{katz}(w_i \mid w_{i-1}) & \text{otherwise} \end{cases}$$

Katz backoff (cont)

- α are used to normalize probability mass so that it still sums to 1, and to "smooth" the lower order probabilities that are used.

$$\sum_{w_i} P_{katz}(w_i|w_{i-2}, w_{i-1}) = 1$$

- See J&M Sec 4.7.1 for details of how to calculate α (and M&S 6.3.2 for additional discussion)

Mixtures / interpolation

- Say I have two distributions $p_1(x)$ and $p_2(x)$
- For any λ in $[0, 1]$, $p(x) = \lambda p_1(x) + (1 - \lambda) p_2(x)$ is a distribution

- Two things to show:
  - (a) Sums to one:

$$\sum_x p(x) = \sum_x \big(\lambda p_1(x) + (1 - \lambda) p_2(x)\big)$$

$$= \lambda \sum_x p_1(x) + (1 - \lambda) \sum_x p_2(x) = \lambda \cdot 1 + (1 - \lambda) \cdot 1 = 1$$

  - (b) All values are ≥ 0

    $p_1(x) \geq 0$ and $p_2(x) \geq 0$ because they're distributions

    ➔ $\lambda p_1(x) \geq 0$ and $(1 - \lambda) p_2(x) \geq 0$ since $\lambda \geq 0$ and $(1 - \lambda) \geq 0$

    ➔ So the sum is non-negative.

Interpolation (cont)

Bigram: $P(w_i|w_{i-1}) = \lambda_2 P_2(w_i|w_{i-1}) + \lambda_1 P_1(w_i)$

$$\lambda_1 + \lambda_2 = 1 \text{ where } \lambda_i \geq 0$$

Trigram:

$$P(w_i|w_{i-1}, w_{i-2}) =$$

$$\lambda_3 P_3(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P_2(w_i|w_{i-1}) + \lambda_1 P_1(w_i)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \text{ where } \lambda_i \geq 0$$

Renormalization (**)

$$\sum_{w_i} P(w_i|w_{i-1}, w_{i-2})$$

$$= \sum_{w_i} (\lambda_3 P_3(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P_2(w_i|w_{i-1}) + \lambda_1 P_1(w_i))$$

$$= (\sum_{w_i} \lambda_3 P_3(w_i|w_{i-1}, w_{i-2})) + (\sum_{w_i} \lambda_2 P_2(w_i|w_{i-1})) + (\sum_{w_i} \lambda_1 P_1(w_i))$$

$$= \lambda_3 (\sum_{w_i} P_3(w_i|w_{i-1}, w_{i-2})) + \lambda_2 (\sum_{w_i} P_2(w_i|w_{i-1})) + \lambda_1 (\sum_{w_i} P_1(w_i))$$

$$= \lambda_3 + \lambda_2 + \lambda_1$$

$$= 1$$

Interpolation (cont)

$$P(w_i|w_{i-1}) = \lambda_2(w_{i-1})P_2(w_i|w_{i-1}) + \lambda_1(w_{i-1})P_1(w_i)$$

$$P(w_i|w_{i-2},w_{i-1}) = \lambda_3(w_{i-2},w_{i-1})P_3(w_i|w_{i-2},w_{i-1})$$

$$+\lambda_2(w_{i-2},w_{i-1})P_2(w_i|w_{i-1}) + \lambda_1(w_{i-2},w_{i-1})P_1(w_i)$$

How to choose the value for lambda?

How to set λ_i?

- Generally, here's what's done (with 3-way data split):
  - Split data into training, dev, and test
  - Train model on training set
  - Use dev to test different values and pick the ones that works best (e.g., maximize the likelihood of the dev data)
  - Test the model on the test data

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Google N-Gram Release

All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training

to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

Google Caveat

- Remember the lesson about test sets and training sets... Test sets should be similar to the training set (drawn from the same distribution) for the probabilities to be meaningful.

- So... The Google corpus is fine if your application deals with arbitrary English text on the Web.

- If not then a smaller domain specific corpus is likely to yield better results.

Summary

- Laplace smoothing: δ
- Good-Turing Smoothing: gt_min[i], gt_max[i].
- Linear interpolation: $\lambda_i(w_{i-2},w_{i-1})$
- Katz Backoff: $\alpha(w_{i-2},w_{i-1})$

Additional slides

Laplace as a mixture

- Say we have K outcomes and $N = \sum_x c(x)$ total observations. Laplace says:

$$p(x) = \frac{c(x)+1}{N+K} = \frac{c(x)}{N+K} + \frac{1}{N+K}$$

$$= \frac{N}{N+K}\frac{c(x)}{N} + \frac{K}{N+K}\frac{1}{K} = \frac{N}{N+K}\frac{c(x)}{N} + \left(1 - \frac{N}{N+K}\right)\frac{1}{K}$$

$$= \frac{N}{N+K}\, p_{MLE}(x) + \left(1 - \frac{N}{N+K}\right)\mathrm{uni}(x)$$

Laplace is a mixture between MLE and uniform! Mixture weight is determined by N and K

Issues in Good-Turing estimation

- If N_{c+1} is zero, how to estimate c^*?
  - Smooth N_c by some functions:
    Ex: $\log(N_c) = a + b \log(c)$
  - Large counts are assumed to be reliable ➔ gt_max[ ]
    Ex: $c^* = c$ for $c >$ gt_max
- May also want to treat n-grams with low counts (especially 1) as zeroes ➔ gt_min[ ].
- Need to renormalize all the estimate to ensure that the probs add to one.
- Good-Turing is often not used by itself; it is used in combination with the backoff and interpolation algorithms.

One way to implement Good-Turing

- Let N be the number of trigram tokens in the training corpus, and min3 and max3 be the min and max cutoffs for trigrams.

- From the trigram counts

  calculate N_0, N_1, …, N_{max3+1}, and N

  calculate a function f(c) , for c=0, 1, …, max3.

$$f(c) = (c+1)\frac{N_{c+1}}{N_c}$$

- Define c^* = c if c > max3
  = f(c) otherwise

- Do the same for bigram counts and unigram counts.

Good-Turing implementation (cont)

- Estimate trigram conditional prob:

$$P_{GT}(w_3|w_1,w_2) = \frac{c^*(w1,w2,w3)}{c^*(w1,w2)}$$

- For an unseen trigram, the joint prob is:

$$P_{GT}(w1,w2,w3) = \frac{c_0^*}{N} = \frac{N_1}{N_0 * N}$$

Do the same for unigram and bigram models

Another example for Good-Turing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- How likely is *octopus*? Since c(*octopus*) = 1. The GT estimate is 1*.
- To compute 1*, we need n_1=3 and n_2=1.

$$1^* = 2 * \frac{1}{3} = \frac{2}{3}$$

- What happens when N_c = 0?

Absolute discounting

c (MLE)	0	1	2	3	4	5	6	7	8	9
c^* (GT)	0.0000270	0.446	1.26	2.24	3.24	4.22	5.19	6.21	7.24	8.25

$$P_{abs}(y \mid x) = \begin{cases} \frac{c(xy)-D}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x) P_{abs}(y) & \text{otherwise} \end{cases}$$

What is the value for D?

How to set α(x)?

Intuition for Kneser-Ney smoothing

- I cannot find my reading __

- P(Francisco | reading) > P(glasses | reading)
  - *Francisco* is common, so interpolation gives P(Francisco | reading) a high value

  - But *Francisco* occurs in few contexts (only after *San*), whereas *glasses* occurs in many contexts.

  - Hence weight the interpolation based on number of contexts for the word using discounting

➔Words that have appeared in more contexts are more likely to appear in some new context as well.

Kneser-Ney smoothing (cont)

Continuation probability:

$$P_{continuation}(w_i) = \frac{|\{w|c(w,w_i)>0\}|}{\sum_{w'}|\{w|c(w,w')>0\}|}$$

Backoff:

$$P_{KN}(w_i|w_{i-1}) = \begin{cases} \frac{c(w_{i-1},w_i)-D}{c(w_{i-1})} & c(w_{i-1},w_i) > 0 \\ \alpha(w_{i-1})P_{cont}(w_i) & otherwise \end{cases}$$

Interpolation:

$$P_{KN}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i) - D}{c(w_{i-1})} + \beta(w_{i-1})P_{cont}(w_i)$$

Summary

- Laplace smoothing (a.k.a. Add-one smoothing)
- Good-Turing Smoothing
- Linear interpolation: $\lambda(w_{i-2},w_{i-1})$, $\lambda(w_{i-1})$
- Katz Backoff: $\alpha(w_{i-2},w_{i-1})$, $\alpha(w_{i-1})$
- Absolute discounting: D, $\alpha(w_{i-2},w_{i-1})$, $\alpha(w_{i-1})$
- Kneser-Ney smoothing: D, $\alpha(w_{i-2},w_{i-1})$, $\alpha(w_{i-1})$
- Class-based smoothing: clusters