Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Backoff and Interpolation

Sometimes it helps to use **less** context

Condition on less context for contexts you haven't learned much about

Backoff: (When we shift to smaller gram for better evidence).

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

Interpolation: (In case when "trigram" is not useful then we use mix) mix unigram, bigram, trigram

Interpolation works better

The practices suggests that interpolation is better

Linear Interpolation

λ sum is to check that probability is 1

1. Simple interpolation

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

$$\sum_i \lambda_i = 1$$

2. Lambdas conditional on context:

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

To make is more complex we make the lambda depended on the previous words, so that we can train the data with more context decisioning

Where the λ come from?

Use a **held-out** corpus

Training Data	Held-Out Data	Test Data

Held-out Data : To set and check meta and/or other parameters.

Choose λs to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

How to set the λs?

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Find set of probabilities such that log probability that occurred in data is the highest.

Smoothing

Till now what we have done;

The words that were not seen before, (i.e. zero according to Corpus)
we assigned some value which is called Smoothing.

But what we have done with the actual word?

Unknown words: Open versus closed vocabulary tasks

If we know all the words in advanced
- Vocabulary V is fixed
- Closed vocabulary task

Specially in the closed set or defined set.

Often we don't know this
- **Out Of Vocabulary** = OOV words
- Open vocabulary task

Words that are not seen in the corpus

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

Therefore, an UNK is created. To use the UNK create a fixed lexicon, then get all the unseen word and change them to UNK.

Huge web-scale n-grams

How to deal with, e.g., Google N-gram corpus

1. Pruning
   - Only store N-grams with count > threshold.
     - Remove singletons of higher-order n-grams
   - Entropy-based pruning

Efficiency

2. Efficient data structures like tries
3. Bloom filters: approximate language models
4. Store words as indexes, not strings

   Use Huffman coding to fit large numbers of words into two bytes
5. Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for Web-scale N-grams

"Stupid backoff" (Brants et al. 2007)

No discounting, just use relative frequencies

S is used for Stupid but not instead of P Probabilities because the weight or the fixed value should be selected against certain logical reasoning.

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4\,S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

Trigram

Bigram

Certain fixed value to avoid zero.

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Unigram

N-gram Smoothing Summary

Add-1 smoothing:

OK for text categorization, not for language modeling

The most commonly used method:

Extended Interpolated Kneser-Ney

For very large N-grams like the Web:

Stupid backoff

Advanced Language Modeling (recently used)

1. Discriminative models:
   choose n-gram weights to improve a task, not to fit the training set
2. Parsing-based models
3. Caching Models
   Recently used words are more likely to appear

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2} w_{i-1}) + (1-\lambda)\frac{c(w \in history)}{|history|}$$

These perform very poorly for speech recognition (why?)

Language Modeling

Advanced:

Kneser-Ney Smoothing

Absolute discounting: just subtract a little from each count

Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros

How much to subtract ?

Church and Gale (1991)'s clever idea

Divide up 22 million words of AP Newswire

- Training and held-out set
- for each bigram in the training set
- see the actual count in the held-out set!

It sure looks like c* = (c - 0.75)

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

Bigram count in training	Bigram count in heldout set
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 Interpolation

Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

Will be discussed

unigram

(Maybe keeping a couple extra values of d for counts 1 and 2)

But should we really just use the regular unigram P(w)?

Actually this is the problem that should we keep the unigram P(w) or we may change the unigram P(w).

Kneser-Ney Smoothing I

Therefore, its good to use the Kneser Ney Smoothing but better to use probabilities of lower-order unigrams.

Kneser-Ney Smoothing I

Better estimate for probabilities of lower-order unigrams!

- Shannon game: I can't see without my reading ___Francisco___ ?

"Francisco" is more common than "glasses"

"Francisco" is the unigram mostly used (or more common) in corpus as the "glasses".

… but "Francisco" always follows "San"

word "Francisco" is more frequent with "San Francisco", therefore, it seems wrong.

The unigram is useful exactly when we haven't seen this bigram!

Instead of P(w): "How likely is w"

$P_{continuation}$(w): "How likely is w to appear as a novel continuation?"

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing II

How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing III

Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

This denominator is same to the previous because the Bigram is predicting a word next to previous two words.
Same the sum of word in each to next of two words i.e. w'_{i+1}, w'

Therefore, a frequent word (Francisco) occurring in only one context (San) will have a low continuation probability.

Kneser-Ney Smoothing IV

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

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Kneser-Ney Smoothing Algorithm

It is used to find the Lower and Higher count probabilities of sequential words (i.e., sentence) in NLP, which is further used in various other techniques.

It is very basic algorithm used in Machine Learning, Machine Translation and Deep Learning.

Q&A

That’s all for today’s Lecture