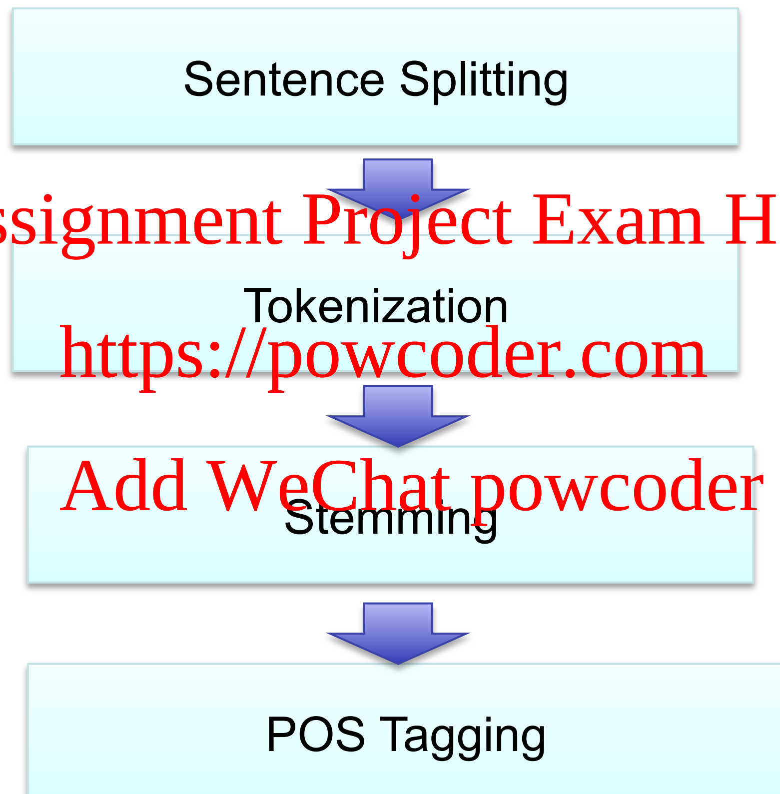
Language Modeling

Lizhen Qu

Recap



Overview of the NLP Lectures

- Introduction to natural language processing (NLP).
- Regular expressions, sentence splitting, tokenization, part-of-speech tagging.
- Language models.
- Vector semantics.
- Parsing.
- Compositional semantics.

Language Models

- Goal: assign a probability to a word sequence.
  - Speech recognition:
    - P(I ate a cherry) > P(Eye eight Jerry)
  - Spelling correction:
    - P(Australian National University) > P(Australian National Univerisity)
  - Collocation error correction:
    - P(high wind) > P(large wind)
  - Machine Translation:
    - P(The magic of Usain Bolt on show…) > P(The magic of Usain Bolt at the show …)
  - Question-answering, summarization, etc.

Probabilistic Language Modeling

- A language model computes the probability of a sequence of words.
  - A vocabulary $\mathcal{V}$.
  - $p(x_1, x_2, ..., x_l) \geq 0$
  - $\sum_{(x_1, x_2, ..., x_l) \in \mathcal{V}^*} p(x_1, x_2, ..., x_l) = 1$

- Related task: probability of an upcoming word.
  - $p(x_4 | x_1, x_2, x_3)$

- LM: Either $p(x_4 | x_1, x_2, x_3)$ or $p(x_1, x_2, ..., x_l)$.

How to Compute $p(x_1, x_2, ..., x_l)$

- Apply chain rule:

$$P(x_1, x_2, ..., x_l) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)...P(x_l|x_1, ..., x_{l-1})$$

- Compute

$$\text{P(All Blacks' hotel room bugged) = P(All) P(Blacks} \mid \text{All) P('}\mid \text{All Blacks)}$$
$$\text{... P(bugged} \mid \text{All Blacks's hotel room)}$$

Estimate the Probabilities

- P(bugged | All Blacks's hotel room) =

$$\frac{Count(\text{All Blacks's hotel room bugged})}{Count(\text{All Blacks's hotel room})}$$

- Not enough data!

Markov Assumption

- Simplification:
  - P(bugged | All Blacks's hotel room) = P(bugged | room)
  - or P(bugged | All Blacks's hotel room) = P(bugged | hotel room)

- First-order Markov assumption:

$$P(x_1, x_2, ..., x_l) = P(x_1) \prod_{i=2}^{l} P(x_i | x_1, ..., x_{i-1})$$

$$= P(x_1) \prod_{i=2}^{l} P(x_i | x_{i-1})$$

Unigram Model

- Zero-order Markov assumption.

$$P(x_1, x_2, ..., x_l) = \prod_{i}^{l} P(x_i)$$

- Examples generated from a unigram model.

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram Model

- First-order assumption.

$$P(x_1, x_2, ..., x_l) = P(x_1) \prod_{i=2}^{l} P(x_i | x_{i-1})$$

- P(I want to eat Chinese food.) = ?

- Estimate bigram probabilities from a training corpus.

Trigram Models

- Second order assumption.

$$P(x_1, x_2, ..., x_l) = P(x_1)P(x_2|x_1)\prod_{i=3}^{l} P(x_i|x_{i-1}, x_{i-2})$$

- long-distance dependencies of language.

"The iphone which I bought one week ago does not stand the cold."

- We can extend to 4-grams , 5-grams …

Restaurant Corpus

Bigrams :

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	0
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

Unigrams :

I	want	to	eat	Chinese	food	lunch
3437	809	1265	3256	938	213	459

Total : 11024

Compute Bigram Probabilities

- Maximum likelihood estimation:

$$P(x_i|x_{i-1}) = \frac{count(x_{i-1}, x_i)}{count(x_{i-1})}$$

- Bigram probabilities:
  - P(want|I) = 1087 / 3437 = 0.32

- Log probabilities:
  - logP(I want to eat Chinese food) = logP(I) + logP(want | I) + logP(to|want) + logP(eat|to) + logP(Chinese|eat) + logP(food|Chinese)

Sequence Generation

- Compute conditional probabilities.
  - P(want | I) = 0.32
  - P(to| I) = 0
  - P(eat|I) = 0.004
  - P(Chinese|I) = 0
  - P (I | I) = 0.002
  - …

- Generate a random number in [0,1].

- See which region it falls into.

Approximating Shakespeare

- Generate sentences from a unigram model.
  - Every enter now severally so, let
  - Hill he late speaks; or! a more to leg less first you enter

- from a bigram model
  - What means, sir. I confess she? then all sorts, he is trim, captain.
  - Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry.

- from a trigram model.
  - Sweet prince, Falstaff shall die.
  - This shall forbid it should be branded, if renown made it empty.

The Perils of Overfitting

- P(I want to eat Chinese lunch.) = ? when count(Chinese lunch) = 0.
- In real life, the test corpus is often different than the training corpus.
- Unknown words!

Generalization by avoiding zeros!

Interpolation

- Key idea: mix of lower-order n-gram probabilities.

- For bigram model:

$$\hat{P}(x_i|x_{i-1}) = \lambda_1 P(x_i|x_{i-1}) + \lambda_2 P(x_i)$$

$$\lambda_1 \geq 0, \lambda_2 \geq 0 \qquad\qquad \lambda_1 + \lambda_2 = 1$$

- For trigram model:

$$\hat{P}(x_i|x_{i-1}, x_{i-2}) = \lambda_1 P(x_i|x_{i-1}, x_{i-2}) + \lambda_2 P(x_i|x_{i-1}) + \lambda_3 P(x_i)$$

$$\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0 \qquad\qquad \sum_i \lambda_i = 1$$

How to Set the Lambdas?

- Estimate λ_i on the held-out data.

Training data | held-out data | test data

- One simple estimation (Collins et al.).

$$\lambda_1 = \frac{count(x_{i-1}, x_{i-2})}{count(x_{i-1}, x_{i-2}) + \gamma}$$

$$\lambda_2 = (1 - \lambda_1) \times \frac{count(x_{i-1})}{count(x_{i-1}) + \gamma}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

Absolute Discounting Interpolation

- Absolute discounting.

$$P_{\text{AbsDiscount}}(x_i|x_{i-1}) = \frac{count(x_i, x_{i-1}) - d}{count(x_{i-1})} + \lambda(x_{i-1})P(x_{i-1})$$

- Often use d = 0.75 in practice.

- Is it sufficient to use $P(x_{i-1})$?

Knerser-Ney Smoothing (i)

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading*___________?
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"

- $P_{\text{continuation}}(x)$: How likely is a word to appear as a novel continuation?
  - For each word x, count the number of bigram types it completes.

$$P_{\text{continuation}}(x_i) \propto |\{x_{i-1} | count(x_{i-1}, x_i) > 0\}|$$

Knerser-Ney Smoothing (ii)

- Example:

$$|\{x_{i-1}|count(x_{i-1}, \text{Francisco}) > 0\}| << \\ |\{x_{i-1}|count(x_{i-1}, \text{glasses}) > 0\}|$$

- Normalized by the total number of bigram types.

$$|\{(x_{j-1}, x_j)|count(x_{j-1}, x_j) > 0\}|$$

$$P_{\text{continuation}}(x_i) = \frac{|\{x_{i-1}|count(x_{i-1}, x_i) > 0\}|}{|\{(x_{j-1}, x_j)|count(x_{j-1}, x_j) > 0\}|}$$

Kneser-Ney Smoothing (iv)

- definition for Bigrams:

$$P_{\mathrm{KN}}(x_i|x_{i-1}) = \frac{\max(count(x_{i-1}, x_i) - d, 0)}{count(x_{i-1})} + \lambda(x_{i-1})P_{\mathrm{continuation}}(x_i)$$

where

$$\lambda(x_{i-1}) = \frac{d}{count(x_{i-1})}|\{x|count(x_{i-1}, x) > 0\}|$$

https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf

Evaluation of Language Models

- Extrinsic evaluation:
  - Put each model in a task
    - Spelling correction, machine translation etc.
  - Time consuming.
  - Task dependent.

- Intrinsic evaluation:
  - perplexity.
  - Useful in pilot experiments.

$$
\begin{aligned}
P(X) &= P(x_1 x_2 \ldots x_L)^{-\frac{1}{L}} \\
&= \sqrt[L]{\frac{1}{P(x_1 x_2 \ldots x_L)}} \\
&= \sqrt[L]{\frac{1}{\prod P(x_i \mid x_{i-1})}}
\end{aligned}
$$

	Unigrams	Bigram	Trigram
Perplexity	962	170	109

Google N-Gram Release

AUG
3

All Our N-gram are Belong to You

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,

That's why we decided 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.

- 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

http://ngrams.googlelabs.com/

Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)
- No discounting, just use relative frequencies

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

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

Tools

- SRILM
  - http://www.speech.sri.com/projects/srilm/
- Berkeley LM
  - https://code.google.com/archive/p/berkeleylm/
- KenLM
  - https://kheafield.com/code/kenlm/
- Available LM models
  - http://www.keithv.com/software/csr/