Australian National University

COMP4650/6490 Document Analysis

Language Modelling & Smoothing

ANU School of Computing

Outline

- Language Models
- Smoothing
- Evaluation of Language Models

Outline

- Language Models
- Smoothing
- Evaluation of Language Models

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, summarisation, etc

Language Models

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

- Related task: probability of an upcoming word
  - $P(x_4 \mid x_1, x_2, x_3)$

- Language model (LM):
  Either $P(x_1, x_2, \dots, x_l)$ or $P(x_l \mid x_1, x_2, \dots, x_{l-1})$

How to Compute $P(x_1, x_2, \ldots, x_l)$

- Apply the chain rule of probability

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

- Example

P(John Smith’s hotel room bugged)

= P(John) P(Smith’s | John) P(hotel | John Smith’s)

… P(bugged | John Smith’s hotel room)

Estimate the Probabilities

- Maximum likelihood estimation (MLE):

  P(bugged | John Smith's hotel room) =

  $$\frac{count(\text{John Smith's hotel room bugged})}{count(\text{John Smith's hotel room})}$$

- P(bugged | John Smith's hotel room) will most likely result in 0, as such specific long sequences are not likely to appear in the training set
- So, we need to simplify the calculations

Markov Assumption

- Simplification:

  $P(\text{bugged} \mid \text{John Smith's hotel room}) \approx P(\text{bugged} \mid \text{room})$
  OR
  $P(\text{bugged} \mid \text{John Smith's hotel room}) \approx P(\text{bugged} \mid \text{hotel room})$

- First-order Markov assumption:

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

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

Unigram Model

- Zero-order Markov assumption

$$P(x_1, x_2, \ldots, x_l) = \prod_{i=1}^{l} P(x_i)$$

- 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

- Not very good!

Bigram Model

- First-order Markov assumption

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

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

- Estimate bigram probabilities from a training corpus

Bigram Counts from Berkeley Restaurant Corpus

x_i

x_{i-1}

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Table contains $count(x_{i-1}, x_i)$, only a subset of the full table of counts shown

Compute Bigram Probabilities

	i	want	to	eat	chinese	food	lunch	spend
$count(x_{i-1})$	2533	927	2417	746	158	1093	341	278

- Maximum likelihood estimation:

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

where special symbols <s> and </s> are added at the beginning/ end of a sentence in the corpus (for a bigram LM).

- Bigram probabilities, e.g. P(want | I) = 827 / 2533 ≈ 0.33

- Log probabilities (logarithm helps with numerical stability)

  logP(<s> *I want to eat Chinese food* </s>)
  = logP(I | <s>) + logP(want | I) + logP(to | want) + logP(eat | to) +
  logP(Chinese | eat) + logP(food | Chinese) + logP(</s> | food)

Trigram Models

- Second order Markov assumption

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

- 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 …

Sequence Generation

- Given 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
  - …
- Sampling:
  - Ensures you don’t just get the same sentence all the time!
  - Can be done by generating a random number in [0,1] based on a uniform distribution
  - Words that are a better fit (in the n-gram context) are more likely to be selected

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 food*) = ?
  when count(*Chinese food*) = 0

- In practice, the test corpus is often different than the training corpus

- Unknown words

- Unseen n-grams

Outline

- Language Models
- Smoothing
- Evaluation of Language Models

Smoothing

- To prevent a language model from assigning zero probability to unseen events (e.g. unknown words, unseen contexts), i.e. not to overfit the training corpus
- Smoothing: adjusting low probabilities (such as zero probabilities) upwards, and high probabilities downwards
- Typically by adjusting the counts in MLE to produce more accurate probabilities
- Works very well, also used to improve NN approaches

Interpolation

- Key idea: mix lower order n-gram probabilities
- λs are hyper-parameters
- For **bigram** models:

$$\hat{P}(x_i \mid x_{i-1}) = \lambda P(x_i \mid x_{i-1}) + (1 - \lambda) P(x_i), \quad \lambda \in [0,1]$$

- For **trigram** models:

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

$$\sum_{i=1}^{3} \lambda_i = 1, \quad \lambda_i \in [0,1]$$

How to Set the λs?

- Choose λ_i that maximise the probability of held-out data

Training data	Held-out data	Test data

- Typically, Expectation Maximisation (EM) is used
- One crude approach (Collins et al. Course notes 2013)

$$\lambda_3 = \frac{count(x_{i-2}, x_{i-1})}{count(x_{i-2}, x_{i-1}) + \gamma}, \quad \lambda_2 = (1 - \lambda_3)\frac{count(x_{i-1})}{count(x_{i-1}) + \gamma},$$

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

  - Ensures λ_i is larger when count is larger
  - Different λs for each n-gram
  - Only one parameter to estimate (i.e. γ)

Absolute Discounting

- Aims to deal with sequences that occur infrequently
- The discount d reserves some probability mass for the unseen n-grams

$$P_{\text{AbsDiscount}}(x_i \mid x_{i-1}) = \frac{\max(count(x_{i-1}, x_i) - d,\ 0)}{count(x_{i-1})} + \lambda(x_{i-1})\, P(x_i)$$

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

- How much to discount? Estimate using a held-out corpus. Typically, $d = 0.75$ is used.
- Is $P_{\text{AbsDiscount}}(x_i \mid x_{i-1})$ a true probability distribution?

Absolute Discounting

For all n-grams that had count c in the training corpus, calculating the average count $\hat{c}$ of all those n-grams in the held-out corpus, and this can be used to estimate d

Bigram count in training set	Bigram count in heldout set
0	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

AP newswire texts.
22 million words for training
22 million words for held-out

We tend to systematically overestimate n-gram counts

Smoothing

Kneser-Ney Smoothing

- Augments absolute discounting
- Key idea (assumption):
  - If a **word** appears after a small number of contexts, it should be less likely to appear in a **novel context**
  - If there are only *a* few words that come after a **context**, then a **novel word** in that context should be less likely
- Consider the word: *Francisco*
  - Might be common because: *San Francisco* is a common term
  - But *Francisco* occurs in very few contexts

Kneser-Ney Smoothing

- For task: I can't see without my reading _____?
  - The word *glasses* seems more likely here than *Francisco*
  - However, a standard unigram model will assign *Francisco* a higher probability than *glasses* because *Francisco* is more common (since *San Francisco* is a very frequent term)
  - We want our unigram model to prefer *glasses*, because *Francisco* is frequent mainly because *San Francisco* is frequent
- $P_{\text{continuation}}(x_i)$ instead of $P(x_i)$
  - Words that have appeared in more contexts are more likely to appear in new context, e.g. *glasses*

Kneser-Ney Smoothing

- $P_{\text{continuation}}(x_i)$
  - How likely is word x_i to continue a new context?
  - Proportional to the number of different words it follows:

    $P_{\text{continuation}}(x_i) \propto |\{x : count(x, x_i) > 0\}|$
  - And

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

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

$$P_{\text{continuation}}(x_i) = \frac{|\{x : count(x, x_i) > 0\}|}{\sum_{x'} |\{x : count(x, x') > 0\}|}$$

Kneser-Ney Smoothing

- Absolute discounting

$$P_{\text{AbsDiscount}}(x_i \mid x_{i-1}) = \frac{\max(count(x_{i-1}, x_i) - d,\ 0)}{count(x_{i-1})} + \lambda(x_{i-1})\, P(x_i)$$

- Kneser-Ney smoothing

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

$\lambda(x_{i-1})$ is defined the same as in absolute discounting

- Is $P_{\text{KN}}(x_i \mid x_{i-1})$ a true probability distribution?

Smoothing

Smoothing for Web-scale N-grams

- *Stupid Backoff* (Brants *et al*. 2007): No discounting, simply backoff to a lower order n-gram (with fixed weight λ) if a higher order n-gram has 0 count
- Does not give a probability distribution

$$S(x_i \mid x_{i-n+1:i-1}) = \begin{cases} \dfrac{count(x_{i-n+1:i})}{count(x_{i-n+1:i-1})} & \text{if } count(w_{i-n+1:i}) > 0 \\ \lambda S(x_i \mid x_{i-n+2:i-1}) & \text{otherwise} \end{cases}$$

$$S(x) = \frac{count(x)}{N}$$

- N is the size of the training corpus in words
- $\lambda = 0.4$ worked well

Smoothing for Web-scale N-grams

- *Stupid Backoff*: A simplistic type of smoothing
- Inexpensive to train on large data sets, and approaches the quality of Kneser-Ney smoothing as the amount of training data increases
- Try to use higher-order n-gram's, otherwise drop to shorter sequence, hence backoff!
  - Every time you 'backoff', e.g. reduce from trigram to bigram because you've encountered a 0 probability, you multiply by 0.4
  - i.e. use trigram if good evidence available, otherwise bigram, otherwise unigram!
- S = scores, not probabilities!

Outline

- Language Models
- Smoothing
- Evaluation of Language Models

Evaluation of LMs

- Extrinsic evaluation:
  - Put each model in a task, e.g. spelling correction, machine translation etc.
  - Usually time consuming
- Intrinsic evaluation:
  - Measures the quality of a language model independent of any application
  - Needs a held-out dataset $x_{1:L}$
  - Perplexity

$$PP(x_{1:L}) = P(x_{1:L})^{-\frac{1}{L}}$$

$$= \sqrt[L]{\frac{1}{P(x_{1:L})}}$$

Evaluation of LMs

Perplexity

- The inverse probability of held-out data, normalised by the number of words
- The lower the better
- Works well when test and training data are similar
- Pre-processing and vocabulary matter (only comparable if two LMs use identical vocabulary)
- Improvement in perplexity does not guarantee increased performance of an NLP task

	Unigrams	Bigram	Trigram
Perplexity	962	170	109

Perplexity of a 1.5M-word WSJ test set

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 Language Models (and training recipe)
  - http://www.keithv.com/software/csr/

References

- Chapter 3. Speech and Language Processing (3rd ed. draft)
- Brants et al. Large language models in machine translation. EMNLP/CoNLL. 2007.
- Course notes for NLP. Michael Collins. http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf
- NLP Lunch Tutorial: Smoothing. Bill MacCartney. https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf