Human Language Technology: Applications to Information Access

Lesson 7a: Language Modeling

November 10, 2016

EPFL Doctoral Course EE-724
Andrei Popescu-Belis
Idiap Research Institute, Martigny

Language modeling (LM)

- Objective
  - compute the probability of any sequence of words
  - or: given a word sequence, predict the most likely next word
  - with:
    - “most likely” given a certain use of language in a domain
    - “probability”: over what space?

- Used in statistical MT, as well as ASR, OCR, spell checkers, handwriting recognition, rule-based MT, authorship, etc.
  - main use of LM: rank candidates of a process based on the likelihood of the sequence of their words

Techniques for LM

- Any method or knowledge that improves modeling
- Traditionally: n-grams (as in this course)
  - modeling a sequence in a Markovian way and using counts to estimate probabilities
  - do not capture the deep properties of language, but appear to work well
- Improvements:
  - use more linguistic information
    - syntactically-based LMs, topic-based LMs
  - alternative sequence modeling
    - neural network based LMs

Plan of the lesson

- Definition of n-gram based LMs
- Learning a language model
  - counts of n-grams
  - smoothing (discounting)
  - interpolation and back-off
- Testing a language model
  - perplexity measures
  - application to tasks such as MT

- Practical work: use in MT system
  - build and query a language model with KenLM

Markov hypothesis

- Goal of LMs: compute the probability of a sequence, or the probability of next word

  $P(w_1, w_2, \ldots w_n)$ or $P(w_n|w_1, w_2, \ldots, w_{n-1})$

  The two formulations are equivalent because
  $P(w_n|w_1, \ldots, w_{n-1}) = P(w_1, \ldots w_n) \,/\, P(w_1, \ldots w_{n-1})$

- Markov chain approximation

  $P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx P(w_n|w_{n-m}, \ldots, w_{n-1})$

  – often $m = 2$ (trigrams), or $m = 1$ (bigrams), or even $m = 0$ (unigrams, no history considered) = the **order** of the LM

  – e.g. trigrams: $P(w_1, w_2, \ldots w_n) \approx \prod_{k=1..n} P(w_k|w_{k-2}, w_{k-1})$

    - with some conventions for the initial words: $P(w_1|w_{-1}, w_0) = P(w_1|\text{<s>})$

Estimating n-gram probabilities

- Simplest idea: maximum likelihood estimate
  - the model is built so that it maximizes the likelihood of what is observed ("what you see is the most likely")

- E.g., for a trigram model built from a (large) text

  $P(w_n | w_{n-2}, w_{n-1}) = \text{count}(w_{n-2}, w_{n-1}, w_n) / \sum_w \text{count}(w_{n-2}, w_{n-1}, w)$

- Problems
  - n-grams not appearing in the corpus get zero probability
  - any string that contains them will have zero probability
    - but in reality not seeing an n-gram in the corpus does not mean it is impossible

Smoothing

- Add some *fictitious* counts, i.e. some mass to all the probabilities, to avoid unseen n-grams having 0 probability

- Simplest smoothing: 'Laplace' or 'add-one'

$$P(w_n | w_{n-2}, w_{n-1}) = (\text{count}(w_{n-2}, w_{n-1}, w_n) + 1) / (\sum_w \text{count}(w_{n-2}, w_{n-1}, w) + \textit{VocabularySize})$$

  where *VocabularySize* is the number of possible trigrams

- Actually, 1 is too much, in practice, for unseen trigrams
  (especially if the number of possible 3-grams is much larger than the corpus)
  - replace it with a much smaller α and adjust denominator
    - α can also be estimated by looking at some held-out data

Other smoothing methods

- Deleted estimation
  - separately adjust counts for unigrams, bigrams, etc., by looking at a held-out corpus of similar size
- Good-Turing
  - adjust probabilities of n-grams based on the number of occurrences of the n-grams of various orders
    - same adjustment probability for n-grams of same order
- Witten-Bell
- Kneser-Ney (modified or not) = state of the art

Other solutions for modeling unseen n-grams

- Back-off
  - if count is zero, combine lower-order n-grams
- Interpolation
  - compute counts on training data for all orders below the intended one
  - combine language models with different orders:

$$P_{\text{INT}}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n \mid w_{n-2}, w_{n-1})$$

    - with $\lambda_1+\lambda_2+\lambda_3 = 1$ (weights set by looking at held-out set)

  - zero counts for an unseen trigram do not necessarily lead to zero interpolated probability
  - can be combined with smoothing

Evaluating LMs

1. Indirect: how much do they help a task? Or, which type of LM most improves the task?
   - complex assessment, depending on the task
2. Direct: perplexity measure
   - given a new observed sentence, compute $P_{LM}(w_1, \ldots, w_n)$: the higher, the better!
   - the cross-entropy over a sentence is defined as:

$$H(P_{LM}) = - \log (P_{LM}(w_1, \ldots, w_n)) / n$$
$$= - \sum_{i=1..n} \log (P_{LM}(w_i | w_1, \ldots, w_{i-1})) / n$$

   - the perplexity of a LM for a sentence or text is $2^{H(P_{LM})}$: the lower, the better!

Do not confuse…

- *Evaluating* a LM
  - given observed text (well-formed English), compare two LMs to see which one gives lower perplexity to the text
  - good LMs should show low perplexities when seeing new but well-formed texts

- *Evaluating* a sentence against an LM = querying the LM
  - find the perplexity of the sentence
  - between two sentences, the one with the lowest perplexity is the most likely to be in “good English”, at least according to the LM

References

- Philipp Koehn, *Statistical Machine Translation*, Cambridge University Press, 2010 – Chapter 7, “Language Modeling”

- Christopher Manning and Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999 – Chapter 13, “Statistical Inference: n-gram Models over Sparse Data”

- Daniel Jurafsky and James H. Martin, *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Second Edition, Prentice-Hall, 2008 – Chapter 4, “N-grams”

- Implemented LM tools for building & querying LMs
  - SRILM, IRSTLM, KenLM, etc.

Possible tasks with KenLM (better follow TP-MT-instructions)

1. Query the model
   - use “mosesdecoder/bin/query” command and the “sample-models/lm/europarl.srilm.gz” language model downloaded last time
   - input: `query [-s] [-n] [--help] lmfile [< input]`
   - output:
     - for each word: index, n-gram, log probability
     - for each sentence: log probability
     - for a text (set of sentences): total perplexity
   - **tasks to do**
     - try various examples of “good” and “bad” English sentences and compare their probabilities
     - try in command line or with an input file; tokenize, lowercase; compare only comparable things
2. Build a new model (in English, but, e.g., on a different domain)
   - use “mosesdecoder/bin/lmplz”
   - documentation at https://kheafield.com/code/kenlm/estimation/
   - needs tokenized and truecased (or lowercased) data, see Moses manual
     - idea: re-use the English side of the parallel corpus from previous TP
   - if large, the model can be binarized with “mosesdecoder/bin/build_binary”
   - **task**: build a new model (e.g. to be used with your Moses MT system)
3. Compare perplexities of various texts against the two LMs
   - send to APB by email some meaningful comparisons and observations