Chapter 3 N-gram Language Models

9/6/20 5:40 PM

-- Models that assign probabilities to sequences of words are called language models or LMs
-- We'll see how to use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequences.

N-gram Language Model (short: n-gram)

- Use n-1 words from the past to approximate probability of nth word
- E.g. bi-gram: approximate probability of proceeding word given one previous word

Probability of a sentence from bi-gram

- Figure 3.1:
  - Column: 1st word of bi-gram
  - Row: 2nd word of bi-gram
  - Numbers: count of frequency
-

	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

- Figure 3.2:
  - Normalized probability (frequency divided by total freq of row)
-

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

- Probability of a sentence:
  -
$$\begin{aligned} P(\texttt{<s> i want english food </s>}) &= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\ &\quad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= .000031 \end{aligned}$$

Evaluating Language Model

- Extrinsic evaluation: end-to-end; embed model in application and measure completely
- Intrinsic evaluation:
  - Training set, development set (validation), test set
  - Better model: fit the test set better (prediction is more accurate in details for test set)

Perplexity (PP)

-
$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

- Intuition: "weighted average branching factor of a language"
  - If each word order appears equally, the perplexity is maximized (multiplication of probabilities is smallest possible) -- This case, harder to predict -> larger perplexity
  - the more information the n-gram gives us about the word sequence, the lower the perplexity
  - Example: different trainings on same corpus
  -

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Generalization

- A case for bigram generalization:
  - Generate a random bigram that starts with <s> (according to bigram probability)
  - Suppose the next word is *w*, then generate a random bigram starting with *w*
  - Continue this generation, until we get bigram ending with </s>
- Training and test corpus should be from same genre of text

Unknown words (2 ways to train <UNK>)

- 1st: convert to closed vocabulary by choosing fixed vocabulary in advance
  - Choose vocabulary list
  - Convert in training set any OOV (out of vocabulary) words to <UNK>
  - Estimate probability of <UNK> just like estimating for other words
- 2nd: No fixed vocab list but create on implicitly
  - Convert in training set words appear under certain frequencies to <UNK>

Smoothing

Motivation: words appear in training and test, but context never shows up in training. - Zero probability
Smoothing makes zero probabilities not exact zero, but small probability

Laplace smoothing

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

C_i: count of word w_i; N: number of tokens (include repetitions), V: number of vocabulary (tokens without repetitions)

Add-k smoothing

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

Usually k is a small number like 0.01, 0.05, 0.1,. Etc

Back-off and Interpolation

Sometimes less context is better, helping to generalize more for contexts that the model hasn't learned much about.

- Back-off: start with trigram is context is sufficient, otherwise reduce to bigram, then unigram
- Interpolation: always mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram and unigram counts
  - E.g. Linear interpolation:

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

    - Lambda's are weights, which should add up to 1
  - A more sophisticated version:

$$\begin{aligned} \hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1})P(w_n) \end{aligned}$$

    - Lambda is conditioned on the context