Machine Learning for natural language processing

N-grams and language models

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2016

Introduction

Goals:

- Estimate the probability that a given sequence of words occurs in a specific language.
- Model the most probable next word for a given sequence of words.

Jurafsky & Martin (2015), chapter 4, and Chen & Goodman (1999)

Table of contents

Motivation

Examples from Jurafsky & Martin (2015)

- (1) Please turn your homework ...
  What is a probable continuation? Rather *in* or *over* and not *refrigerator*.

- (2) a. all of a sudden I notice three guys standing on the sidewalk
  b. on guys all I of notice sidewalk three a sudden standing the
  Which of the two word orders is better?

Language model (LM): Probabilistic model that gives $P(w_1 \dots w_n)$ and $P(w_n | w_1 \dots w_{n-1})$

Motivation

Applications:

- Tasks in which we have to identify words in noisy, ambiguous input: **speech recognition**, **handwriting recognition**, ...
- **spelling correction**

Example

(3) a. their is only one written exam in this class
    b. there is only one written exam in this class

- **machine translation**: among a series of different word orders in the target language, one has to choose the best one.

Example

(4) a. Das Fahrrad wird er heute reparieren.
    b. The bike will he today repair
    c. The bike he will today repair.
    d. The bike he will repair today.

N-grams

Notation: $w_1^m = w_1 \ldots w_m$.
Question: How can we compute $P(w_1^m)$?

$$\begin{aligned} P(w_1^m) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_m|w_1^{m-1}) \\ &= \prod_{k=1}^{m} P(w_k|w_1^{k-1}) \end{aligned}$$

But: computing $P(w_k|w_1^{k-1})$ for a large k is not feasible.

Approximation of $P(w_k|w_1^{k-1})$: *N-grams*, i.e., look at just the $n-1$ last words, $P(w_k|w_{k-n+1}^{k-1})$.

Special cases:

- *unigrams*: $P(w_k)$
- *bigrams*: $P(w_k|w_{k-1})$
- *trigrams*: $P(w_k|w_{k-2}w_{k-1})$

N-grams

With n-grams, we get

1. Probability of a sequence of words:

$$P(w_1^l) \approx \prod_{k=1}^{l} P(w_k | w_{k-n+1}^{k-1})$$

2. Probability of a next word:

$$P(w_l | w_1^{l-1}) \approx P(w_l | w_{l-n+1}^{l-1})$$

These are strong independence assumptions called *Markov* assumptions. E.g. with bigrams

Example

$P(\text{einfach}|\text{die Klausur war nicht}) \approx P(\text{einfach}|\text{nicht})$

Maximum likelihood estimation (MLE)

Question: How do we estimate the n-gram probabilities?

Maximum likelihood estimation (MLE): Get n-gram counts from a corpus and normalize so that the values lie between 0 and 1.

$$P(w_k|w_{k-n+1}^{k-1}) = \frac{C(w_{k-n+1}^{k-1} w_k)}{C(w_{k-n+1}^{k-1})}$$

In the bigram case, this amounts to

$$P(w_k|w_{k-1}) = \frac{C(w_{k-1} w_k)}{C(w_{k-1})}$$

We augment sentences with an initial $\langle \texttt{s} \rangle$ and a final $\langle \texttt{/s} \rangle$

Maximum likelihood estimation (MLE)

Example from Jurafsky & Martin (2015)

Training data:

```
< s > I am Sam < /s >
< s > Sam I am < /s >
< s > I do not like green eggs and ham < /s >
```

Some bigram probabilities:

$$P(\texttt{I}|\texttt{< s >}) = \frac{2}{3} \qquad P(\texttt{Sam}|\texttt{< s >}) = \frac{1}{3} \qquad P(\texttt{am}|\texttt{I}) = \frac{2}{3}$$
$$P(\texttt{< /s >}|\texttt{Sam}) = \frac{1}{2} \qquad P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} \qquad P(\texttt{do}|\texttt{I}) = \frac{1}{3}$$

Maximum likelihood estimation (MLE)

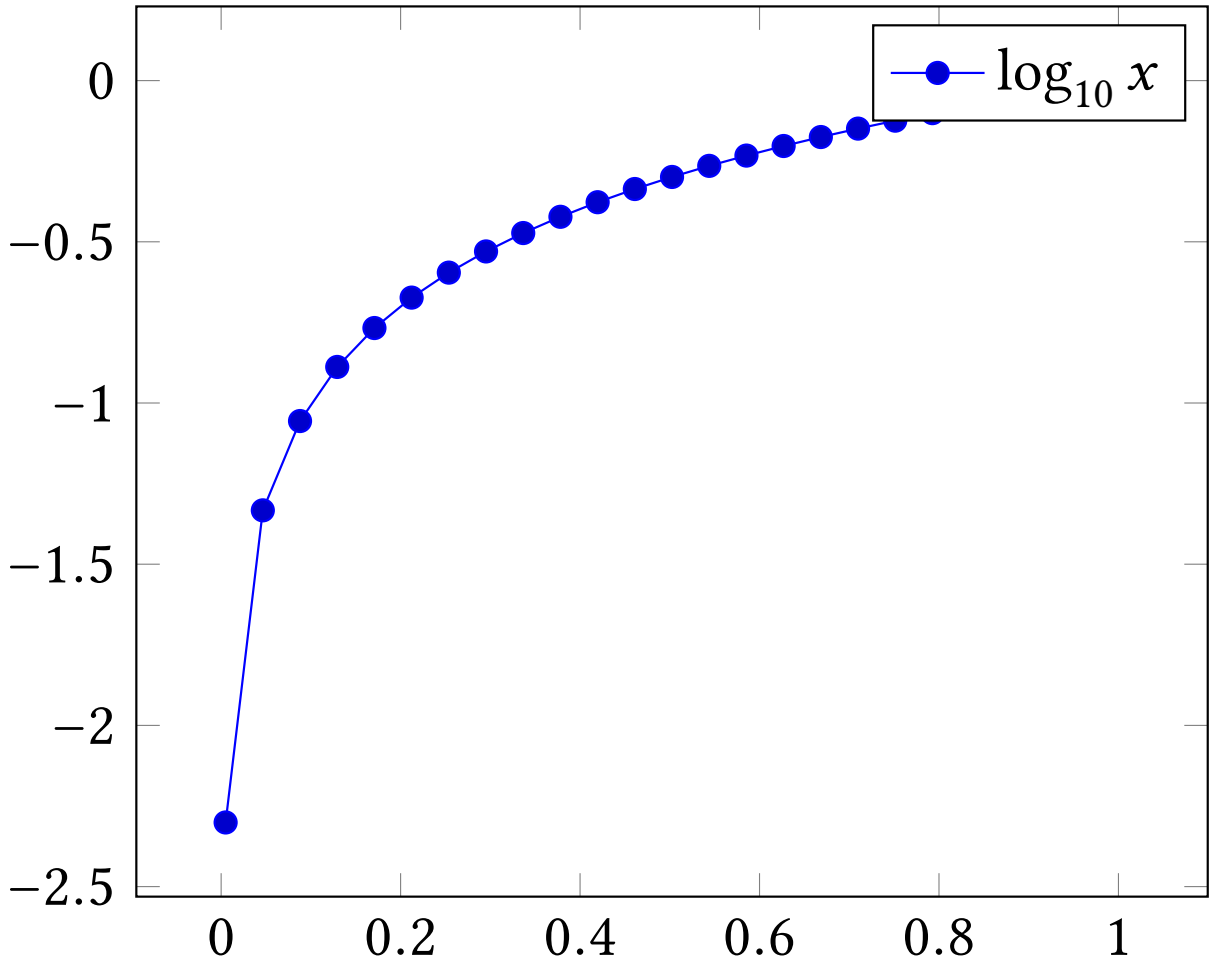
Practical issues:

- In practice, n is mostly between 3 and 5, i.e., we use trigrams, 4-grams or 5-grams.
- LM probabilities are always represented as *log probabilities*. Advantage: Adding replaces multiplying and numerical overflow is avoided.

$$p_1 \cdot p_2 \cdot \ldots p_l = \exp(\log p_1 + \log p_2 + \cdots + \log p_l)$$

Maximum likelihood estimation (MLE)

Reminder: $\log 1 = 0, \log 0 = -\infty$



Evaluating language models

The data is usually separated into

- a training set (80% of the data),
- a test set (10% of the data),
- and sometimes a development set (10% of the data).

The model is estimated from the training set. The higher the probability of the test set, the better the model.
Instead of measuring the probability of the test set, LMs are usually evaluated with respect to the *perplexity* of the test set. The higher the probability, the lower the perplexity.

Evaluating language models

The perplexity of a test set $W = w_1 w_2 \ldots w_N$ is defined as

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

With our n-gram model, we get then for the perplexity:

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{k=1}^{N} P(w_k | w_{k-n+1}^{k-1})}}$$

Evaluating language models

A different way to think about perplexity: it measures the *weighted average branching factor* of a language.

Example

$L = \{a, b, c, d\}^*$. Frequencies are such that $P(a) = P(b) = P(c) = P(d) = \frac{1}{4}$ (independent from the context).
For any $w \in L$, given this model, we obtain

$$PP(w) = \sqrt[|w|]{\frac{1}{\prod_{k=1}^{|w|} \frac{1}{4}}} = \sqrt[|w|]{\frac{1}{\frac{1}{4}^{|w|}}} = \sqrt[|w|]{4^{|w|}} = 4$$

The perplexity of any $w \in L$ under this model is 4.

Evaluating language models

Example

$L = \{a, b, c, d\}^*$. Words in the language contain three times as many as as they contain bs, cs or ds. $P(a) = \frac{1}{2}$ and $P(b) = P(c) = P(d) = \frac{1}{6}$. For any $w \in L$ with these frequencies and with $|w| = 6n$:

$$PP(w) = \sqrt[6n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 2 \cdot 2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[6n]{2^{6n} \cdot \sqrt{3}^{6n}} = 2\sqrt{3} = 3.46$$

Assume that we use the same model but test it on a W with equal numbers of as, bs, cs and ds, $|W| = 4n$. Then we get

$$PP(W) = \sqrt[4n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[4n]{2^{4n} \cdot 3^{3n}} = 2\sqrt[4n]{3^{\frac{3}{4}4n}} = 2\sqrt[4]{27} = 4.56$$

Unknown words

Problem: New text can contain

- unknown words; or
- unseen n-grams.

In these cases, with the algorithm seen so far, we would assign probability 0 to the entire text. (And we would not be able to compute perplexity at all.)

Example from (Jurafsky & Martin, 2015)

Words following the bigram *denied the* in WSJ Treebank 3 with counts:

denied the allegations	5
denied the speculation	2
denied the rumors	1
denied the report	1

If the test set contains *denied the offer* or *denied the loan*, the model would estimate its probability as 0.

Unknown words

Unknown or *out of vocabulary* words:

- Add a pseudo-word ⟨UNK⟩to your vocabulary.
- Two ways to train the probabilities concerning ⟨UNK⟩:
  1. Choose a vocabulary V fixed in advance. Any word $w \notin V$ in the training set is converted to ⟨UNK⟩. Then estimate probabilities for ⟨UNK⟩as for all other words.
  2. Replace the first occurrence of every word w in the training set with ⟨UNK⟩. Then estimate probabilities for ⟨UNK⟩as for all other words.

Smoothing

Unseen n-grams: To avoid probabilities 0, we do *smoothing*: Take off some probability mass from the events seen in training and assign it to unseen events.

Laplace Smoothing (or *add-one* smoothing):

- Add 1 to the count of all n-grams in the training set before normalizing into probabilities.
- Not so much used for n-grams but for other tasks, for instance text classification.

- For unigrams, if N is the size of the training set and $|V|$ the size of the vocabulary, we replace $P(w) = \frac{C(w)}{N}$ with $P_{Laplace}(w) = \frac{C(w)+1}{N+|V|}$.
- For bigrams, we replace $P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$ with $P_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+|V|}$.

Smoothing

Smoothing methods for n-grams that use the $(n-1)$-grams, $(n-2)$-grams etc.:

- *Backoff*: use the trigram if it has been seen, otherwise fall back to the bigram and, if this has not been seen either, to the unigram.
- *Interpolation*: Use always a weighted combination of the trigram, bigram and unigram probabilities.

Linear interpolation:

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

with $\sum_i \lambda_i = 1$.

Smoothing

More sophisticated: each λ is computed conditioned on the context.

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

In both cases,

- the probabilities are first estimated from the training corpus,
- and the λ parameters are then estimated from separate held-out data.
- They are estimated such that they maximize the likelihood of the held-out data. This can be done for example using the *EM algorithm* (to be introduced later in this course).

Smoothing

Most commonly used N-gram smoothing method: *Kneser-Ney* algorithm Kneser & Ney (1995); Chen & Goodman (1999).

Idea: discount the count of an n-gram by the average discount we see in a held-out corpus.

Table from Jurafsky & Martin (2015)

Average bigram counts in held-out corpus of 22 million words for all bigrams in 22 million words training data of AP newswire (Church & Gale, 1991):

count training	average count held-out	count training	average count held-out
0	0.0000270	5	4.21
1	0.448	6	5.23
2	1.25	7	6.21
3	2.24	8	7.21
4	3.23	9	8.26

Smoothing

Absolute discounting:

$$P_{AbsoluteDiscounting}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

Possible discount $d = 0.75$.

Further refinement: replace $P(w_i)$ with the probability that we see w_i as a continuation. Underlying intuition: If w_i has appeared in many contexts, it is a more likely continutation in a new context.

$$P_{continuation}(w_i) = \frac{|\{w \,|\, C(ww_i) > 0\}|}{|\{w_{j-1}w_j \,|\, C(w_{j-1}w_j) > 0\}|} = \frac{|\{w \,|\, C(ww_i) > 0\}|}{\sum_{w_j} |\{w_{j-1} \,|\, C(w_{j-1}w_j) > 0\}|}$$

Smoothing

Interpolated Kneser-Ney smoothing:

$$P_{KN}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)$$

where $\lambda(w_{i-1})$ is a normalizing constant:

$$\lambda(w_{i-1}) = \frac{d}{C(w_{i-1})}|\{w|\, C(w_{i-1}w) > 0\}|$$

Here,

- $\frac{d}{C(w_{i-1})}$ is the normalized discount, and
- $|\{w|\, C(w_{i-1}w) > 0\}|$ is the number of word types that can follow w_{i-1}, i.e., the number of times we applied the normalized discount.

Smoothing

Finally, we obtain as a generalization (Chen & Goodman, 1999):

Interpolated Kneser-Ney

For $n > 1$:

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

and the recursion terminates with

$$P_{KN}(w_i) = \frac{|\{w|\,C(ww_i) > 0\}|}{|\{w_{j-1}w_j\,|\,C(w_{j-1}w_j) > 0\}|}$$

References

Chen, Stanley F. & Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language* 13. 359–394.

Church, K. W. & W. A. Gale. 1991. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language* 5. 19–54.

Jurafsky, Daniel & James H. Martin. 2015. Speech and language processing. an introduction to natural language processing, computational linguistics, and speech recognition. Draft of the 3rd edition.

Kneser, R. & H. Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing*, vol. 1, 181–184. Detroit, MI.