Introduction to Computational Linguistics: N-gram language models

Sharon Goldwater
(with some slides from Philipp Koehn)

9 July 2015

School of informatics

Language models

- **Language models** tell us $P(\vec{w}) = P(w_1 \dots w_n)$: *How likely to occur is this string of words?*

  Roughly: *Is this string of words a "good" one in my language?*

- Sentence processing:
  - Can we define a model that predicts human grammaticality judgments? or processing times? or errors?

- Phonology:
  - Model words w consisting of phonemes c_i, so $P(w) = P(c_1 \dots c_n)$.
  - Can we define a model that predicts "goodness" of non-words (e.g., plick vs psick vs pnick)?

Estimating a language model

- We want to know $P(\vec{w}) = P(w_1 \dots w_n)$ for big n (e.g., sentence).

- What will not work: try to directly estimate probability of each full sentence (e.g., using MLE).
  - **Sparse** data: lots of sentences will never have been seen before (MLE=0).
  - Storage: cannot store probabilities for all possible sentences.

A first attempt to solve the problem

Perhaps the simplest way to model sentence probabilities: a **unigram** model.

- Generative process: choose each word in the sentence independently.

- Resulting model: $$\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$$

- Not a *good* model, but still a model.

- Of course, $P(w_i)$ also needs to be estimated!

MLE for unigrams

- How to estimate $P(w)$, e.g., $P(\text{the})$?

- Remember that MLE is just relative frequencies:

$$P_{\text{ML}}(w) = \frac{C(w)}{N}$$

  - $C(w)$ is the token count of w in a large corpus
  - $N = \sum_{x'} C(x')$ is the total number of word tokens in the corpus.

Unigram models in practice

- Seems like a pretty bad model of language: probability of word obviously *does* depend on context.

- Yet unigram (or **bag-of-words**) models are surprisingly useful for some applications.
  - Can model "aboutness": topic of a document, semantic usage of a word
  - Applications: lexical semantics (disambiguation), information retrieval, text classification. (See, e.g., J&M 20.2, 23.1)
  - But, we will focus on models that capture at least some syntactic information.

General n-gram language models

$$\begin{aligned} P(\vec{w}) &= P(w_1 \dots w_n) & (1)\\ &= P(w_n|w_{n-1}, w_{n-2}, \dots w_1)P(w_{n-1}|w_{n-2}, \dots w_1) \dots P(w_1) & (2)\\ &\approx P(w_n|w_{n-1}, w_{n-2})P(w_{n-1}|w_{n-2}, w_{n-3}) \dots P(w_1) & (3) \end{aligned}$$

- (1) By definition

- (2) Using chain rule

- (3) Makes a conditional independence assumption
  - **Markov** assumption: only a finite history matters (w_i is cond. indep. of $w_1 \dots w_{i-3}$ given w_{i-1}, w_{i-2}). Here, two word history = **trigram** model.

Estimating N-Gram Probabilities

- Maximum likelihood (relative frequency) estimation for bigrams:

$$P_{\text{ML}}(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

- Or trigrams:

$$P_{\text{ML}}(w_3|w_1, w_2) = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)}$$

- Collect counts over a large text corpus
  - Millions to billions of words are easy to get
  - (trillions of English words available on the web)

Derivation of MLE formulas

- Defn of conditional probability: $P(B|A) = \frac{P(A,B)}{P(A)}$
- Let $A =$ "w_1 is first item in bigram", $B =$ "w_2 is second item in bigram".

$$\begin{aligned} P_{\mathrm{ML}}(w_2|w_1) &= \frac{P_{\mathrm{ML}}(w_1, w_2)}{P_{\mathrm{ML}}(w_1, \cdot)} \\ &= \frac{C(w_1, w_2)/(N-1)}{C(w_1, \cdot)/(N-1)} \\ &= \frac{C(w_1, w_2)}{C(w_1, \cdot)} \end{aligned}$$

- Note subtlety in edge case.

Example: 3-Gram

- Counts for trigrams in Europarl corpus, and estimated word probabilities

the green (total: 1748)			the red (total: 225)			the blue (total: 54)		
word	c.	prob.	word	c.	prob.	word	c.	prob.
paper	801	0.458	cross	123	0.547	box	16	0.296
group	640	0.367	tape	31	0.138	.	6	0.111
light	110	0.063	army	9	0.040	flag	6	0.111
party	27	0.015	card	7	0.031	,	3	0.056
ecu	21	0.012	,	5	0.022	angel	3	0.056

- 225 trigrams in the Europarl corpus start with the red
- 123 of them end with cross

→ maximum likelihood probability is $\frac{123}{225} = 0.547$.

How good is the LM?

- A good model M assigns a text of real English $\vec{w}$ a high probability.
- Can be measured with **cross entropy**:

$$H_M(w_1 \dots w_n) = -\frac{1}{n} \log P_M(w_1 \dots w_n)$$

  - Avg neg log probability our model assigns to each word we saw
- Or, **perplexity**:

$$\mathrm{PP}_M(\vec{w}) = 2^{H_M(\vec{w})}$$

Example: 3-Gram

prediction	P_{ML}	$-\log_2 P_{\mathrm{ML}}$
P_{ML}(i|</s><s>)	0.109	3.197
P_{ML}(would|<s>i)	0.144	2.791
P_{ML}(like|i would)	0.489	1.031
P_{ML}(to|would like)	0.905	0.144
P_{ML}(commend|like to)	0.002	8.794
P_{ML}(the|to commend)	0.472	1.084
P_{ML}(rapporteur|commend the)	0.147	2.763
P_{ML}(on|the rapporteur)	0.056	4.150
P_{ML}(his|rapporteur on)	0.194	2.367
P_{ML}(work|on his)	0.089	3.498
P_{ML}(.|his work)	0.290	1.785
P_{ML}(</s>|work .)	0.99999	0.000014
	average	2.634

Comparison 1–4-Gram

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Unseen N-Grams

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus

Unseen N-Grams

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus

  → $P(\text{consuming}|\text{shall commence}) = 0$
- Any sentence with shall commence consuming will be assigned probability 0

The guests shall commence consuming supper
Green inked shall commence consuming garden the

The problem with MLE

- MLE estimates probabilities that make the observed data maximally probable
- by assuming anything unseen cannot happen
- It **over-fits** the training data
- **Smoothing** methods reassign some probability mass from observed to unobserved events

Add-One Smoothing

- For all possible bigrams, add one more count.

$$P_{\mathrm{ML}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

Add-One Smoothing

- For all possible bigrams, add one more count.

$$P_{\mathrm{ML}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

- NO! Summing over possible values of w_i (for vocabulary V) must equal 1:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = 1$$

- True for P_{ML} but we increased the numerator; must change denominator too.

Add-One Smoothing: normalization

- We want:

$$\sum_{w_i \in V} \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + x} = 1$$

- Solve for x:

$$\begin{aligned} \sum_{w_i \in V} (C(w_{i-1}, w_i) + 1) &= C(w_{i-1}) + x \\ \sum_{w_i \in V} C(w_{i-1}, w_i) + \sum_{w_i \in V} 1 &= C(w_{i-1}) + x \\ C(w_{i-1}) + v &= C(w_{i-1}) + x \end{aligned}$$

- So, $P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)+1}{C(w_{i-1})+v}$ where v = vocabulary size.

Add-One Smoothing: effects

- Large vobulary size means v is often much larger than $C(w_{i-1})$, overpowers actual counts.
- Ex: in Europarl, $v = 86,700$ word types (30m tokens, max $C(w_{i-1})$ = 2m).
- Compute some example probabilities:

$C(w_{i-1}) = 10,000$		
$C(w_{i-1}, w_i)$	$P_{\mathrm{ML}} =$	$P_{+1} \approx$
100	1/100	1/970
10	1/1k	1/10k
1	1/10k	1/48k
0	0	1/97k

$C(w_{i-1}) = 100$		
$C(w_{i-1}, w_i)$	$P_{\mathrm{ML}} =$	$P_{+1} \approx$
100	1	1/870
10	1/10	1/9k
1	1/100	1/43k
0	0	1/87k

The problem with Add-One smoothing

- All smoothing methods "steal from the rich to give to the poor"
- Add-one smoothing steals way too much
- ML estimates for frequent events are quite accurate, don't want smoothing to change these much.

Add-α Smoothing

- Add $\alpha < 1$ to each count

$$P_{+\alpha}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha v}$$

- Simplifying notation: c is n-gram count, n is history count

$$P_{+\alpha} = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?

Optimizing α

- Divide corpus into **training** set (80-90%), **held-out** (or **development**) set (5-10%), and **test** set (5-10%)
- Train model (estimate probabilities) on training set with different values of α
- Choose the value of α that minimizes perplexity on development set
- Report final results on test set

A general methodology

- Training/dev/test split is used across machine learning/NLP, and often also appropriate for CL (esp cognitive modeling).
- Development set used for evaluating different models, debugging, optimizing/fitting parameters (like α)
- Test set performance measures how well model *generalizes* once final model and parameters are chosen. (Ideally: once per paper)
- Avoids overfitting to the training set and even to the test set

Adjusted Counts

- Previously, we estimated probabilities based on actual counts

$$P_{\text{ML}} = \frac{c}{n}$$

- Then, we changed the formula to estimate smoothed probabilities

$$P_{+\alpha} = \frac{c+\alpha}{n+\alpha v}$$

- Another view: we adjusted the counts c

$$P_{+\alpha} = \frac{c^*}{n} \quad \Rightarrow \quad c^* = n\, P_{+\alpha} = (c+\alpha)\,\frac{n}{n+\alpha v}$$

Good-Turing Smoothing

- Adjust actual counts c to expected counts c^* with formula

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

  – N_c number of n-grams that occur exactly c times in corpus
  – N_0 total number of unseen n-grams

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
c	N_c	c^*	t_c
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

t_c are average counts of n-grams in test set that occurred c times in corpus

Good-Turing justification: 0-count items

- Estimate the probability that the next observation is previously unseen (i.e., will have count 1 once we see it)

$$P(\text{unseen}) = \frac{N_1}{n}$$

This part uses MLE!

- Divide that probability equally amongst all unseen events

$$P_{\text{GT}} = \frac{1}{N_0}\frac{N_1}{n} \quad \Rightarrow \quad c^* = \frac{N_1}{N_0}$$

Good-Turing justification: 1-count items

- Estimate the probability that the next observation was seen once before (i.e., will have count 2 once we see it)

$$P(\text{once before}) = \frac{2N_2}{n}$$

- Divide that probability equally amongst all 1-count events

$$P_{\text{GT}} = \frac{1}{N_1}\frac{2N_2}{n} \quad \Rightarrow \quad c^* = \frac{2N_2}{N_1}$$

- Same thing for higher count items

Problems with Good-Turing

- Assumes we know the vocabulary size (no unseen words) [but see J&M 4.3.2]
- Doesn't allow "holes" in the counts (if $N_i > 0$, $N_{i-1} > 0$) [but see J&M 4.5.3]
- Applies discounts even to high-frequency items [but see J&M 4.5.3]
- Divides shifted probability mass evenly between all items of same frequency.