TLN:

Language Models: N-grams

Daniele Radicioni

outline

- probabilistic language modeling
  - from (unigrams) bigrams to n-gram models
  - counts and probabilities
- maximum likelihood estimation
- evaluation of language models
- smoothing

credits

- Jurafsky, D. (2000). Speech & language processing. Pearson Education India.
  - Chapter 3: 3.1, 3.2, 3.3, 3.4
- lecture by Mike Hammond, Linguistics 696f, Statistical Natural Language Processing,
  - Chapter 4, https://faculty.sbs.arizona.edu/hammond/archive/ling696f-sp03/snlp4.pdf
- source code from the Internet, credited within the code samples

N-gram language models

assign a probability to a sentence

- machine translation
  - P('Please complete this task') > P('Please complete this abstract')
- spell correction
  - P('Please turn your homework in') > P('Please tourne your homework in')
- speech recognition, summarization, question answering, ...

probabilistic language modeling

- models that assign probabilities to sequences of words are called language models or LMs
  - the simplest model that assigns probabilities to sentences and sequences of words is the n-gram: an n-gram is a sequence of *n* words:
    - a 2-gram (bigram) is a two-word sequence of words like "please turn";
    - a 3-gram (a trigram) is a three-word sequence of words like "please turn your", or "turn your homework".
- n-gram models can be used to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequences

estimating probabilities by counting

- one way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see
- predicting a word w given some history h is to compute $P(w|h)$
  - this would answer the question "out of all the times we saw the history h, how many times was it followed by the word w?"

$$P(\text{the}|\text{its air is so clear that}) = \frac{C(\text{its air is so clear that})}{C(\text{its air is so clear})}$$

- NO! even the web is not big enough to give us good estimates in most cases
  - even simple extensions of the example sentence may have counts of zero on the web

notation

- simplification: we represent the probability of a given random variable X_i assuming the value "the" not as $P(X_i = \text{"the"})$, but more simply as $P(the)$
- we represent a sequence of N words either as $w_1 \ldots w_n$ or $w_{1:n}$ (e.g., the expression $w_{1:n-1}$ means the string $w_1, w_2, \ldots, w_{n-1}$)
  - we indicate the joint probability of each word in a sequence having a particular value as $P(w_1, w_2, \ldots, w_n)$

chain rule of probability

- how to compute the probabilities for a whole sequence, such as $P(w_1, w_2, \ldots, w_n)$?
  - this probability can be decomposed through the chain rule:

$$P(X_1 \ldots X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_{1:2}) \ldots P(X_n \mid X_{1:n-1}) = \prod_{k=1}^{n} P(X_k \mid X_{1:k-1})$$

- applying the chain rule to words we obtain

$$P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2}) \ldots P(w_n \mid w_{1:n-1}) = \prod_{k=1}^{n} P(w_k \mid w_{1:k-1})$$

chain rule of probability

- $\prod_{k=1}^{n} P(w_k \mid w_{1:k-1}) =$

  $P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2}) \ldots P(w_n \mid w_{1:n-1})$

  - P("its water is so transparent") = P(its) × P(water | its) × P(is | its water) × P(so | its water is) × P(transparent | its water is so)
- we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities
  - however, we can't just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before

approximating the history

- intuition: the history can be approximated by just considering the last few words
  - the bigram model, for example, approximates the probability of a word given all the previous words $P(w_n \mid w_{1:n-1})$ by using only the conditional probability of the preceding word $P(w_n \mid w_{n-1})$.
  - P(transparent | its water is so) ≈ P(transparent | so)
- this is generalized by the formula

$$P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-1})$$

Markov assumption

- Markov assumption: the probability of a word depends only on the previous word
  - Markov models are a class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past
- we can generalize from the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the n-gram (which looks $n-1$ words into the past).

n-grams: general form

- the general approximation for n-grams is $P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-N+1:n-1})$, and the probability of a full word sequence can be computed as $P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$
- how does the previous formula differ from the chain rule, $P(w_{1:n}) = \prod_{k=1}^{n} P(w_k \mid w_{1:k-1})$?
- how to compute such probabilities?

bigrams

L, a language with 3 words

- let us consider a language L with only three words, $\{a, b, c\}$
  - each word has equal likelihood to occur, which is then $.33$ $\times_{0.\bar{1}}$
  - likelihood of sequences of length two is illustrated in the table

xy	P(x)*P(y)
aa	0.1
ab	0.1
ac	0.1
ba	0.1
bb	0.1
bc	0.1
ca	0.1
cb	0.1
cc	0.1
TOTAL	1

another L

- let us consider a language with only three words, $\{a, b, c\}$
  - if we modify word probabilities, e.g. P(a)=.5; P(b)=.25; P(c)=.25
  - likelihood of sequences of length two changes as illustrated in the table
- this distribution is a language model (L), where higher-probability emissions are better exemplars of L than lower-probability texts
  - e.g., *aa* is a better exemplar than *bc* or *cc*.

xy	P(x)*P(y)
aa	0.25
ab	0.125
ac	0.125
ba	0.125
bb	0.0625
bc	0.0625
ca	0.125
cb	0.0625
cc	0.0625
TOTAL	1

bigrams

- in a bigram model the history is restricted to the immediately preceding word:

  $$P(w_1 w_2 \ldots w_i) = P(w_1) \times P(w_2 \mid w_1) \times \ldots \times P(w_i \mid w_{i-1})$$

- where the conditional probability is computed as the fraction between the number of cases where both words occur divided by the number of cases where only the first one was found: $P(w_i \mid w_{i-1}) = \frac{|w_{i-1} w_i|}{|w_{i-1}|}$

bigrams

Peter Piper picked a peck of pickled pepper.

Where's the pickled pepper that Peter Piper picked?

- 16 words in these sentences. 'Peter' occurs twice, so P(Peter) = 2/16 = .125
- word frequencies are reported in table

word	frequency
Peter	2
Piper	2
picked	2
a	1
peck	1
of	1
pickled	1
pepper	1
Where's	1
the	1
that	1

bigrams

Peter Piper picked a peck of pickled pepper.

Where's the pickled pepper that Peter Piper picked?

- the conditional probability of 'Piper' given 'Peter' is computed as

$$P(\text{Piper}\,|\,\text{Peter}) = \frac{|\,\text{Peter Piper}\,|}{|\,\text{Peter}\,|} = \frac{2}{2} = 1$$

- bigram frequencies are reported in table

bigrams	big. freq.
picked a	1
pepper that	1
peck of	1
a peck	1
pickled pepper	2
Where s	1
Piper picked	2
the pickled	1
Peter Piper	2
of pickled	1
pepper Where	1
that Peter	1
s the	1

bigrams

Peter Piper picked a peck of pickled pepper.

Where's the pickled pepper that Peter Piper picked?

- the conditional probability of 'Peter Peter' is computed as

$$P(\text{Peter}\,|\,\text{Peter}) = \frac{|\,\text{Peter Peter}\,|}{|\,\text{Peter}\,|} = \frac{0}{2} = 0$$

- bigram frequencies are reported in table

bigrams	big. freq.
picked a	1
pepper that	1
peck of	1
a peck	1
pickled pepper	2
Where s	1
Piper picked	2
the pickled	1
Peter Piper	2
of pickled	1
pepper Where	1
that Peter	1
s the	1

bigrams

- we also need to consider the edges of sentences
  - it is necessary to estimate the probability of starting and final words
- we add two markers to delimit sentences (we also refer to these as to 'paddings')
  - <s> Peter Piper picked a peck of pickled pepper. </s>
  - <s> Where's the pickled pepper that Peter Piper picked? </s>
  - in this setting, $P(w_1 w_2 \ldots w_n)$ is computed as $P(w_1 \mid < \text{s} >) \times P(w_2 \mid w_1) \times \ldots \times P(< /\text{s} > \mid w_n)$

some little counts

<s> Peter Piper picked a peck of pickled pepper. </s>

<s> Where's the pickled pepper that Peter Piper picked? </s>

- let us compute P(Peter Piper picked)

$= P(\text{Peter} \mid < \text{s} >) \times P(\text{Piper} \mid \text{Peter}) \times P(\text{picked} \mid \text{Piper}) \times P(</\text{s}> \mid \text{picked})$

$= .5 \times 1 \times 1 \times .5 = .25$

some little counts

<s> Peter Piper picked a peck of pickled pepper. </s>

<s> Where's the pickled pepper that Peter Piper picked? </s>

- let us compute P(Peter Piper picked)

$$= P(\text{Peter} \mid < \text{s} >) \times P(\text{Piper} \mid \text{Peter}) \times P(\text{picked} \mid \text{Piper}) \times P(\text{</s>} \mid \text{picked})$$

$$= .5 \times 1 \times 1 \times .5 = .25$$

some little counts

<s> Peter Piper picked a peck of pickled pepper. </s>

<s> Where's the pickled pepper that Peter Piper picked? </s>

- let us compute P(Peter Piper picked)

$$= P(\text{Peter} \mid < \text{s} >) \times P(\text{Piper} \mid \text{Peter}) \times P(\text{picked} \mid \text{Piper}) \times P(\text{</s>} \mid \text{picked})$$

$$= .5 \times 1 \times 1 \times .5 = .25$$

some little counts

<s> Peter Piper picked a peck of pickled pepper. </s>

<s> Where's the pickled pepper that Peter Piper picked? </s>

- let us compute P(Peter Piper picked)

$$= P(\text{Peter} \mid <\text{s}>) \times P(\text{Piper} \mid \text{Peter}) \times P(\text{picked} \mid \text{Piper}) \times P(</\text{s}> \mid \text{picked})$$

$$= .5 \times 1 \times 1 \times .5 = .25$$

some little counts

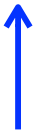
<s> Peter Piper picked a peck of pickled pepper. </s>

<s> Where's the pickled pepper that Peter Piper picked? </s>

- let us compute P(Peter Piper picked)

$$= P(\text{Peter} \mid < \text{s} >) \times P(\text{Piper} \mid \text{Peter}) \times P(\text{picked} \mid \text{Piper}) \times P(\text{</s>} \mid \text{picked})$$

$$= .5 \times 1 \times 1 \times .5 = .25$$



sequence generation

- only few specific sequences can be built based on the language model acquired, without reducing the probabilities to zero
  - those sequences that occur in the training set (i.e., the tongue twister)
  - sequences of unbounded length can be generated through the language model, although no guarantee can be provided that generated sentences are grammatical...

maximum likelihood estimation

maximum likelihood estimation

- to estimate probabilities we can use maximum likelihood estimation, MLE
  - get counts from a corpus, and then normalize counts, so that these lie in $[0,1]$
  - for example, to compute a particular bigram probability of a word y given a previous word x, we take the count of the bigram $C(xy)$ and normalize by the sum of all the bigrams that share the same first word x: $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$, which can be simplified as $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$

maximum likelihood estimation

- to estimate probabilities we can use maximum likelihood estimation, MLE
  - get counts from a corpus, and then normalize counts, so that these lie in $[0,1]$
  - for example, to compute a particular bigram probability of a word y given a previous word x, we take the count of the bigram $C(xy)$ and normalize by the sum of all the bigrams that share the same first word x: $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$, which can be simplified as $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$

the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

relative frequency. this formula estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix.

```
<s>good morning everybody</s>
<s>morning good everybody</s>
<s>every morning a gazelle</s>
```

P(morning | good) = 1/2	P(good | <s>) = 1/3
P(everybody | morning) = 1/3	P(</s> | gazelle) = 1/1
P(good | everybody) = 0/2	P(a | morning) = 1/3

MLE

- maximum likelihood estimation (MLE): relative frequencies are a means to estimate probabilities
  - if we observed that the word 'morning' occurs 400 times in a corpus of 1 million words, the probability for a randomly selected word from that corpus to be 'morning' is 400/1000000 = .0004
  - this figure may change if we consider another corpus; however, it still is the most likely for the considered corpus

	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

bigram counts form a corpus of >9000 sentences

borrowed from cited Jurafsky, (2000)

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

	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

bigram counts form a corpus of >9000 sentences

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

unigram counts form same corpus

borrowed from cited Jurafsky, (2000)

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

borrowed from cited Jurafsky, (2000)

	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

bigram counts form a corpus of >9000 sentences

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

unigram counts form same corpus

	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

bigram probabilities after normalization

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

	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

bigram counts form a corpus of >9000 sentences

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

unigram counts form same corpus

	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								
eat								
chi								
food								
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

bigram probabilities after normalization

$$P(\text{to} \mid \text{want}) = \frac{C(\text{want to})}{C(\text{want})} = \frac{608}{927} = 0.6559$$

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

borrowed from cited Jurafsky, (2000)

	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

bigram counts form a corpus of >9000 sentences

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

unigram counts form same corpus

	i	want	to	eat	chinese	food	lunch	spend
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

$$P(\text{food} \mid \text{chinese}) = \frac{C(\text{chinese food})}{C(\text{chinese})} = \frac{82}{158} = 0.5189$$

bigram probabilities after normalization

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

borrowed from cited Jurafsky, (2000)

computing P of an entire sentence

	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

- let assume that P(i | <s>) = .25, and compute the probability of the sentence "i want to eat chinese food":
  - P(i want to eat chinese food) = .25 × .33 × .66 × .28 × .021 × .52 = 0.00016648632

syntactic knowledge

- bigram probabilities grasp syntactic regularities, such as after 'eat' we are likely to have a noun or an adjective; or that 'to' is mostly followed by a verb
- even further knowledge, such as that this corpus reflects higher probabilities for Chinese food vs. French food

from 2- to n-grams

- trigram models condition on the previous two words; 4-grams models condition on the previous three words; ...
  - for larger n-grams it is necessary to assume extra context to the left and right of the sentence (e.g., at the beginning two pseudo-words may be employed for the first trigram).

log probabilities

- probabilities are by definition less than or equal to 1, so that the more probabilities we multiply together, the smaller the product becomes
  - multiplying enough n-grams together would result in numerical underflow (i.e., the result is smaller in magnitude than the smallest value representable)
  - adding is faster than multiplying...
- we thus employ log probabilities
  - multiplying in linear space is equivalent to adding in log space, so that $p_1 \times p_2 \times p_3 \times p_4 = \log p_1 + \log p_2 + \log p_3 + \log p_4$

evaluation of LMs

extrinsic vs. intrinsic evaluation

- extrinsic evaluation: the first way to assess LMs is to use them in an application, and to measure whether and how the application improves
  - e.g., in speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and then comparing the accuracies
- intrinsic evaluation: metric measuring the quality of a model independent from a given application
  - the probabilities of an n-gram model are acquired from the corpus it is trained on, which we call the training set
  - the quality of an n-gram model can be measured through its performance on some unseen data, which we call the test set

intrinsic evaluation

- the evaluation procedure implies partitioning the data into training and test sets, train the parameters of both models on the training set, and then compare how well the two trained models fit the test set
  - fits best the test set whichever model assigns to the test set higher probability (that is 'predicts' the test set)

perplexity

- let us consider a word sequence of k elements, $W = \{w_1, w_2, \ldots, w_k\}$; since we are interested in evaluating the model on unseen data, the test sequence W must be new, and not be part of the training set.

- given the language model LM, we can compute the probability of the sentence W , that is LM(W).

perplexity

- the higher the probability, the better the model
  - average log probability computed based on the model is defined as $\frac{1}{k}\log\prod_{i=1}^{k}\text{LM}(W) = \frac{1}{k}\sum_{i=1}^{k}\log\text{LM}(W)$, which is the log probability of the whole test sequence W, divided by the number of tokens in the sequence
- the perplexity of sequence W given the language model LM is computed as $\text{PPL(LM,W)} = \exp\{-\frac{1}{k}\sum_{i=1}^{k}\log\text{LM}(w_i|w_{1:i-1})\}$
  - it is now clear why low PPL values (corresponding to high probability values) indicate that the word sequence fits well to the model or, equivalently, that the model is able to predict that sequence.

zeros

generative features

- probabilities encode specific facts about a given training corpus
  - n-grams do a better and better job of modeling the training corpus as we increase the value of N
- let assume we are able to generate random sentences from different n-gram models
  - let us consider the case of unigrams
  - imagine all the words of the English language covering the probability space between 0 and 1, each word covering an interval proportional to its frequency
  - we choose a random value between 0 and 1 and print the word whose interval includes this chosen value
  - we continue choosing random numbers and generating words until we randomly generate the sentence-final token </s>

unknown words

- we may incur a bigram whose probability is zero
  - e.g., in the training set we have 'denied the reports', 'denied the claims', 'denied the request' while in the test set we find 'denied the loan'...
- and words that simply do not appear in the training set?
  - closed vocabulary assumption: test set can only contain words from a given lexicon, such that no unknown words may be found
  - such an assumption may be reasonable in some domains, such as speech recognition where the pronunciation dictionary is fixed in advance

open vocabulary

- in this case we may need to deal with words which were never seen before, unknown words, or out of vocabulary (OOV) words
  - the percentage of OOV words that appear in the test set is called the OOV rate
- an open vocabulary system is one in which we model these potential unknown words in the test set by adding a pseudo-word called <UNK>

how to sort up UNK words (1)

- two main strategies exist to train the probabilities of the unknown word model <UNK>
  - the first one is to turn the problem back into a closed vocabulary one by choosing a fixed vocabulary in advance:
  - this can be done by choosing a vocabulary; converting words (in the training set) that are not in the vocabulary into a <UNK> token; estimate the probabilities for <UNK> like for any other word in the training set

how to sort up UNK words (2)

- as an alternative, when the vocabulary is not given beforehand, is to create a close vocabulary implicitly, replacing words in the training data by <UNK> based on their frequency
  - e.g., replace by <UNK> all word that occur less than n times in the training set

smoothing

motivation for smoothing

- what happens with words that are in our vocabulary (they are not unknown words) but appear in a test set in an unseen context?
  - for example, a word that in test set occurs after a word they never appeared after in training
- we want to prevent the language model from assigning zero probability to such previously unseen events:
  - we need to reduce a bit the probability mass from some more frequent events to give it to the events never seen.

Laplace smoothing

- the simplest way to do smoothing is to add one to all the bigram counts, before we normalize them into probabilities
  - all counts that were zero will now have a count of 1, the counts of 1 will be 2, and so on
- let us consider Laplace smoothing for unigrams
  - the maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N: $P(w_i) = \frac{c_i}{N}$
  - Laplace smoothing adds one to each count. since we have V words in the vocabulary and each one was incremented, the denominator is also adjusted to account for the additional V extra observations: $P_{Lap}(w_i) = \frac{c_i + 1}{N + V}$

Laplace smoothing

- instead of changing both the numerator and denominator, we can also describe how a smoothing algorithm affects the numerator, by defining an adjusted count c^*
- then the adjusted count needs to be multiplied by a normalization factor $\frac{N}{N+V}$: that is $c_i^* = (c_i + 1)\frac{N}{N+V}$
  - the adjusted count is then turned into a probability P_i^* by normalizing by N
- smoothing can be seen as discounting (that is, lowering), some non-zero counts in order to get the probability mass that will be assigned to the zero counts; in this view a relative discount d_c is the ratio between the discounted counts and the original counts, $d_c = \frac{c^*}{c}$

borrowed from cited Jurafsky, (2000)

	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

bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

add-one smoothed bigram counts

normalization

- the normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

- for the add-one smoothed bigram counts we augment the unigram count by the number of total word types in the vocabulary V: $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$

- e.g., in the next example the unigram counts will be augmented by $V = 1446$

	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

bigram probabilities after normalization

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

add-one smoothed bigram probabilities

borrowed from cited Jurafsky, (2000)

	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

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

	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								
eat								
chi								
food								
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

$$P(\text{to} \mid \text{want}) = \frac{C(\text{want to})}{C(\text{want})} = \frac{608}{927} = 0.6559$$

simple normalization

unigram counts form same corpus

bigram probabilities after normalization

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

borrowed from cited Jurafsky, (2000)

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to								055
eat								00046
chines								00062
food								00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

$$P(\text{to} \mid \text{want}) = \frac{C(\text{want to})}{C(\text{want})} = \frac{609}{927 + 1446} = 0.2566$$

add-one smoothed bigram probabilities

unigram counts

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

borrowed from cited Jurafsky, (2000)

backoff

- intuition: using less context may be helpful to generalize for contexts under-represented in the training set
- if we are trying to compute $P(w_n | w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can estimate its probability by using the bigram probability $P(w_n | w_{n-1})$
  - further, if we don't have counts to compute $P(w_n | w_{n-1})$, we can look to the unigram $P(w_n)$
  - in backoff one considers a lower-order n-gram if no evidence was collected for a higher-order n-gram

interpolation

- in interpolation, we mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram, and unigram counts.
  - in this setting the trigram probability $P(w_n \mid w_{n-2}w_{n-1})$ is estimated weighting uni-, bi-, trigram probabilities by λ:

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

  - where the λs must sum to 1, that is $\sum_i \lambda_i = 1$

LAB: twitting like (a) Trump

consegna

- dato il Trump Twitter Archive (~290 tweet attribuiti all'ex Presidente US, disponibile fra i materiali della lezione)
  - acquisire due language models (uno a bi-grammi e uno a tri-grammi) su questo set di testi;
  - utilizzare i due modelli per produrre tweet