University of Malta
BSc IT (Hons)Year IV

CSA4050: Advanced Topics Natural Language Processing

Lecture N-Grams

Statistical Approaches to NLP

- Sentence and Word Prediction
- N-Gram models
- Bigram Calculations

Dept Computer Science and AI 2004/05
Lecturer: Michael Rosner

Grammaticality versus Sentence Probability

The Linguist Says

"It must be recognised that the notion of "probability of a sentence" is an entirely useless one, under any known interpretation of this term".
Noam Chomsky 1969

The Statistician Says

"Anytime a linguist leaves the group the recognition rate goes up".
Fred Jelenek (1988), Head of the then IBM speech group, in an address to a speech recognition workshop.

Word Prediction is Useful

1. Automatic Speech Recognition (ASR)
2. Handwriting Recognition (OCR)
3. Augmentative Communication
4. Spell Checking

In 1 and 2, input is noisy and does not give enough information to completely specify the intended word, some kind of top-down knowledge about words and their combinations seems necessary.

In 3, the ability to predict the next word can improve the efficiency of devices designed to help users to specify the next sentence.

In 4 errors can be often detected by probabilistic models of syntax particularly when the misspelt strings are actually valid words.

Probability of a Word Sequence

- Model we shall discuss concerns words but is equally valid for other kinds of sequence.

- Problem: how to compute the probability of a complete string of words $P(w_1 \ldots w_n)$.

  Below we use the abbreviation w_1^n $(\equiv w_1 \ldots w_n)$.

- The chain rule for probabilities gives us

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

- The problem is that in general we do not know how to compute $P(w_k \mid w_1^{k-1})$ i.e. the probability of a word given a long sequence of preceding words.

Markov Models: Bigram

- We cannot obtain $P(w_k \mid w_1^{k-1})$ by counting: most corpora are too small to contain useful information about such sequences.
- The obvious solution is to use the following approximation:

  $P(w_k \mid w_1^{k-1}) \approx P(w_k \mid w_{k-1})$

  That is, we *approximate* the probability of a word given all the previous words as the probability of the word given just the previous word.
- The assumption that $P(w_n)$ depends only on $P(w_{n-1})$ is called a *Markov* Assumption.
- Markov models are the class of probabilistic models which assume that we can predict the probability of some future event by looking a finite distance into the past.
- A bigram looks one word into the past.

Bigram Probability of a Sequence

- The bigram probability of a sting is the product of the probabilities of all the bigrams that make it up. So for example,

$$P(abcd) = P(a \mid b) * P(b \mid c) * P(c \mid d)$$

- Hence the bigram probability of a string w_1^n is

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

N-Gram Approximation

The general equation for N-gram approximation of the conditional probability of the next word in a sequence is:

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

- **Bigram Approximation, N=2**

  $P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1}^{n-1})$
  $= P(w_n \mid w_{n-1})$

- **Trigram Approximation N=3**

  $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-2}^{k-1})$

Bigram Example: Berkeley Restaurant Project Jurafsky(1994)

Example Utterances

I'm looking for Cantonese food
I'd like to eat dinner someplace nearby
I'm looking for a good place to eat breakfast
When is Caffe Venezia open during the day?

Sample of Bigram Probabilities

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat breakfast	.03
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

Calculating Probability of a Sentence

P(I want to eat British Food) =
P(I $|< s >$)P(want | I)P(to | want)P(eat | to)P(British | eat)P(food | British)

$= .25 * .32 * .65 * .26 * .002 * .60$
$= .000016$

- The longer the string, the smaller the probability
- Risk of numerical underflow
- Many standard programs for computing N-gram store bigram probabilities as logarithms (base 2).

Computing Bigram Probabilities

N-gram models can be trained by counting and normalising

We take some training corpus, and from this corpus take the count of a particular bigram, and divide this count by the number sum of bigrams that share the same first word:

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

We can simplify this since the sum of all the bigram counts that start with a given word w_{n-1} is the same as the unigram count for that word

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

Bigram Probabilities

Unigram Counts

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

Bigram Counts

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.02	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Higher Order N-grams

- Increasing accuracy of N-gram models as we increase the value of N.
- Very strong dependency of the model on the training corpus.

  The cardinal sin in Statistical NLP is to test on your training data

- Sparse data: One major problem with standard N-gram models is that they must be trained from some finite corpus from which certain perfectly legitimate N-grams are missing. Hence the training matrix is sparse.

  Jelinek reports that using a 1.5 million word corpus and applying the resulting trigram model to a 300K word text, 25% of the trigram types in the second text did not appear.

Accuracy

1-gram	Every enter now severally so, let Hill he late speaks; or! a more to leg less first you enter. No coherent relation between words; no sentence-final punctuation.
2-gram	What means, sir. I confess she? then all sorts, he is trim, captain. Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. Strong local coherence between word-to-word coherence (esp. punctuation).
3-gram	Sweet prince, Falstaff shall die. Harry of Monmouth's grave. Indeed the duke; and had a very good grave
4-gram	It cannot be but so. They say all lovers swear more performance than they are wont to keep obliged faith unforfeited!

The trigram and quadrigram sentences look a lot like Shakespeare; too much so. Once the generator has chosen "it cannot be but", there is only one continuation, given the training data.

Training set is far too small for the task at hand. Corpus = 1M.

Dealing with Sparse Data

- Abstraction over the data to increase the size of equivalence classes.
  - Tagged Data
  - Instead, n-grams are calculated only for the K most common words. The rest are considered as ”out of vocabulary” (OOV) and mapped to a single token (e.g. <UNK>).
- Smoothing Techniques

Add-One Smoothing for Unigrams

Unsmoothed MLE of the unigram probability

$P(w_x) = \frac{\text{count of } w_x}{\text{corpus size}} = \frac{c(w_x)}{\sum_i c(w_i)} = \frac{c(w_x)}{N}$

where N is the total number of tokens. The various smoothing techniques will rely upon an the notion of an *adjusted count* c^*. The adjusted count for add-one smoothing can be defined by:

- adding one to the count and then
- multiplying by a normalisation factor $\frac{N}{N+V}$:

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

Here V is the *vocabulary size*, i.e. the number of word types. Since we are adding 1 to the count of each word type, the total number of tokens N, must be increased by the number of types, V.

Turning Adjusted Counts into Probabilities

The adjusted count c_i^* can then be turned into an *adjusted probability* p_i^* by normalisation (this time by N - i.e. total number of *actual* tokens).

$$p_i^* = \frac{c_i^*}{N} = \frac{N(c_i+1)}{N(N+V)} = \frac{c_i+1}{N+V}$$

The effect is to give a little bit of the probability space to unseen events.

Now we turn to the case of bigrams.

Unsmoothed versus Add-One Adjusted Bigram Counts

Here are the original counts:

ORIG.	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

So we add one to all of them:

ADD-1	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	12	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	19	1	17	1	1	1	1
lunch	5	1	1	1	1	2	1

Smoothed Bigram Probabilities

Recall that normal bigram probabilities are computed by normalising counts by the unigram count:

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

The adjusted probabilities $p*(w_n \mid w_{n-1})$ are obtained by

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

(It is as though we have added all possible word pairs to the corpus)

Here, V is 1616 - the total number of word types in the Berkeley corpus. The effect is marked.

word	count	count +1616
I	3437	5053
want	1215	2931
eat	938	2554
lunch	459	2075

Unsmoothed v. Smoothed Bigram Probabilities

ORIG.	I	want	to	eat	Chinese	food	lunch
I	.0023	$\boxed{.32}$	0	.0038	0	0	0
want	.0025	0	$\boxed{.65}$	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.02	.0021	.055
Chinese	.0094	0	0	0	0	$\boxed{.56}$	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Here are the smoothed probabillities.

ADD1	I	want	to	eat	Chinese	food	lunch
I	.0018	$\boxed{.22}$	.00020	.0028	.0002	.0002	.0002
want	.0014	.00035	$\boxed{.28}$	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	$\boxed{.066}$	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.0003
lunch	.0024	.00048	.00048	.00048	.00048	.0096	.0004

Although difficult to read, we can discern that some of the changes are disproportionate.

Add-one Smoothed Bigram Counts

To make things easier to read we can can reconstruct the count matrix

ORIG.	I	want	to	eat	Chinese	food	lunch
I	8	$\boxed{1087}$	0	13	0	0	0
want	3	0	$\boxed{786}$	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	$\boxed{120}$	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

ADD1	I	want	to	eat	Chinese	food	lunch
I	6	$\boxed{740}$	.68	10	.68	.68	.68
want	2	.42	$\boxed{331}$	.42	3	4	3
to	3	.69	8	594	3	.69	9
eat	.37	.37	1	.37	7.4	1	20
Chinese	.36	.12	.12	.12	.12	$\boxed{15}$	.24
food	10	.48	9	.48	.48	.48	.48
lunch	1.1	.22	.22	.22	.22	.44	.22

- Add-one smoothing has made a very big change to the counts.
- Note that the adjusted count for “Chinese food” went from 120 to 15.

Add-One Smoothing - Remarks

- The main disadvantage of add-one smoothing is that the estimates are dependent on the size of the vocabulary.
- The larger the number of unseen elements, the more probability mass is “stolen” from the seen ones.
- So for sparse data sets over large vocabularies, the method is unsatisfactory. In the case at hand:
  - too much probability mass is assigned to previously unseen bigrams, and
  - too little probability is assigned to those bigrams having non-zero counts.

Gale and Church (1994): What's wrong with adding one?

- A study by Gale and Church (1994) concludes that
  - add-one is worse at predicting zero-count bigrams than other methods.
  - Variances of the counts produced by the add-one method are actually worse than than those for the unsmoothed method.