Probabilistic Language Modeling

Regina Barzilay

EECS Department

MIT

November 15, 2004

Last time

- Corpora processing
- Zipf's law
- Data sparseness

Today: Probabilistic language Modeling

Predicting String Probabilities

Which string is more likely?
(Which string is more grammatical?)

- *Grill doctoral candidates.*
- *Grill doctoral updates.*

(example from Lee 1997)

Methods for assigning probabilities to strings are called **Language Models**.

Motivation

Speech recognition, spelling correction, optical character recognition and other applications

- Let E be physical evidence, and we need to determine whether the string W is the message encoded by E
- Use Bayes Rule:

$$P(W|E) = \frac{P_{LM}(W)P(E|W)}{P(E)},$$

where $P_{LM}(W)$ is language model probability

- $P_{LM}(W)$ provides the information necessary for disambiguation
  (esp. when the physical evidence is not sufficient for disambiguation)

How to Compute it?

Naive approach:

- Use the maximum likelihood estimates (MLE): the number of times that the string occurs in the corpus S, normalized by the corpus size

$$P_{MLE}(``Grill\ doctorate\ candidates'') = \frac{count(``Grill\ doctorate\ candidates'')}{|S|}$$

- For unseen events, $P_{MLE} = 0$
  - Dreadful behavior in the presence of **Data Sparseness**

Two Famous Sentences

”It is fair to assume that neither sentence

“Colorless green ideas sleep furiously”

nor

“Furiously sleep ideas green colorless”

. . . has ever occurred . . . Hence, in any statistical model . . . these sentences will be ruled out on identical grounds as equally “remote” from English. Yet (1), though nonsensical, is grammatical, while (2) is not.” [Chomsky 1957]

The Language Modeling Problem

- Start with some vocabulary: $\nu = \{$the, a, doctorate, candidate, Professors, grill, cook, ask, $\ldots\}$
- Get a training sample of $\nu^{\star}$:
  Grill doctorate candidate.
  Cook Professors.
  Ask Professors.
  $\ldots$
- Assumption: training sample is drawn from some underlying distribution P

The Language Modeling Problem

Goal: learn a probability distribution P' "as close" to P as possible

- $\sum_{x \in \nu^\star} P'(x) = 1$
- $P'(x) \geq 0$ for all $x \in \nu^\star$

$P'(candidates) = 10^{-5}$
$P'(ask\ candidates) = 10^{-8}$

Deriving Language Model

Assign probability to a sequence $w_1 w_2 \ldots w_n$

Apply chain rule:

$$P(w_1 w_2 \ldots w_n) = P(w_1|S) * P(w_2|S, w_1) * P(w_3|S, w_1, w_2) \ldots P(E|S, w_1, w_2, \ldots, w_n)$$

- **History-based model:** we predict following things from past things
- How much context do we need to take into account?

Markov Assumption

For arbitrary long contexts $P(w_i|w_{i-n} \ldots w_{i-1})$ difficult to estimate

Markov Assumption: w_i depends only on n preceding words
Trigrams (second order):

$$P(w_i|START, w_1, w_2, \ldots, w_{i-1}) = P(w_i|w_{i-1}, w_{i-1})$$

$$P(w_1 w_2 \ldots w_n) = P(w_1|S) * P(w_2|S, w_1) * P(w_3|w_1, w_2) * \ldots P(E|w_{n-1}, w_n)$$

A Computational Model of Language

A useful conceptual and practical device: coin-flipping models

- A sentence is generated by a randomized algorithm
  - The generator can be one of several “states”
  - Flip coins to choose the next state
  - Flip other coins to decide which letter or word to output
- Shannon: “The states will correspond to the “residue of influence” from preceding letters”

Word-Based Approximations

- First-order approximation

  To him swallowed confess hear both. which. OF save on trail for are ay device and rote life have

  Every enter now severally so, let

  Hill he late speaks; or! a more to leg less first you enter

- Third-order approximation

  King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv's in;

  Will you tell me how I am?

  It cannot be but so.

Evaluating a Language Model

- We have n test string:

$$S_1, S_2, \dots, S_n$$

- Consider the probability under our model $\prod_{i=1}^{n} P(S_i)$, or log probability:

$$\log \prod_{i=1}^{n} P(S_i) = \sum_{i=1}^{n} \log P(S_i)$$

- Perplexity:

$$Perplexity = 2^{-x},$$

where $x = \frac{1}{W} \sum_{i=1}^{n} \log P(S_i)$ and W is the total number of words in the test data.

Perplexity

Perplexity is a measure of effective “branching factor”

- We have a vocabulary ν of size N, and model predicts
$$P(w) = \frac{1}{N}$$
for all the words in ν.

- What about Perplexity?

$$Perplexity = 2^{-x},\ where\ x = \log\frac{1}{N}$$

$$Perplexity = N$$

Perplexity

- estimate of human performance (Shannon, 1951)
  - Shannon game — humans guess next letter in text
  - PP=142(1.3 bits/letter), uncased, open vocabulary
- estimate of trigram language model (Brown et al. 1992)
  - PP=790(1.75 bits/letter), cased, open vocabulary

Maximum Likelihood Estimate

MLE makes training data as probable as possible

$$P_{ML}(w_i|w_{i-1}, w_{i-2}) = \frac{Count(w_{i-2}, w_{i-1}, w_i)}{Count(w_{i-2}, w_{i-1})}$$

For vocabulary of size N, we will have N^3 parameters in the model

For $N = 1,000$, we have to estimate $1,000^3 = 10^9$ parameters

Problem: how to deal with unseen words?

Sparsity

The aggregate probability of unseen events constitutes a large fraction of the test data

Brown et al (1992): considered a 350 million word corpus of English, 14% of trigrams are unseen

Today

How to estimate probability of unseen elements?

- Discounting
  - Laplace
  - Good-Turing
- Linear Interpolation
- Katz Back-Off

Add-One (Laplace) Smoothing

Simplest discounting technique:

$$P(w_i|w_{i-1}) = \frac{C(w1, w2) + 1}{c(w1) + |\nu|},$$

where $|\nu|$ is a vocabulary size

- Bayesian estimator assuming a uniform unit prior on events
- Problem: Too much probability mass to unseen events

Example

Assume $|\nu| = 10,000$, and $|S| = 1,000,000$

$$P_{MLE}(ball|kick\ a) = \frac{Count(kick\ a\ ball)}{Count(kick\ a)} = \frac{9}{10} = 0.9$$

$$P_{+1}(ball|kick\ a) = \frac{Count(kick\ a\ ball) + 1}{Count(kick\ a) + |\nu|} = \frac{9+1}{10 + 10,000} = 9{*}10^{-4}$$

Weaknesses of Laplace

- For Sparse distribution, Laplace's Law gives too much of the probability space to unseen events
- Worst at predicting the actual probabilities of bigrams than other methods

More reasonable to use add-ϵ smoothing (Lidstone's Law)

Good-Turing Discounting

How likely are you to see a new word type in the future? Use things you've seen once to estimate the probability of unseen things

- n_r — number of elements with r frequency and $r > 0$
- n_0 — size of the total lexicon minus the size of observed lexicon
- Modified count for elements with frequency r

$$r^\star = (r+1) * \frac{n_{r+1}}{n_r}$$

Good-Turing Discounting: Intuition

Goal: estimate how often word with r counts in training data occurs in test set of equal size.
We use deleted estimation:

- delete one word at a time
- if "test" word occurs $r + 1$ times in complete data set:
  - it occurs r times in "training" set
  - add one count to words with r counts
- total count placed to bucket for r-count words is: $n_{r+1} * (r + 1)$
- (avg-count of r count words) = $\frac{n_{r+1} * (r+1)}{n_r}$

Good-Turing Discounting (cont.)

In Good-Turing, the total probability assigned to all the unobserved events is equal to n_1/N, where N is the size of the training set. It is the same as a relative frequency formula would assign to singleton events.

Example: Good-Turing

Training sample of 22,000,000 (Church&Gale'1991)

r	N_r	heldout	$r\star$
0	74,671,100,000	0.00027	0.00027
1	2,018,046	0.448	0.446
2	449,721	1.25	1.26
3	188,933	2.24	2.24
4	105,668	3.23	3.24
5	68,379	4.21	4.22
6	48,190	5.23	5.19

The Bias-Variance Trade-Off

- (Unsmoothed) trigram estimate

$$P_{ML}(w_i|w_{i-2}, w_{i-1}) = \frac{Count(w_{i-2}w_{i-1}w_i)}{Count(w_{i-2}, w_{i-1})}$$

- (Unsmoothed) bigram estimate

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

- (Unsmoothed) unigram estimate

$$P_{ML}(w_i) = \frac{Count(w_i)}{\sum_j Count(w_j)}$$

How close are these different estimates to the "true" probability $P(w_i|w_{i-2}, w_{i-1})$?

Interpolation

- One way of solving the sparseness in a trigram model is to mix that model with bigram and unigram models that suffer less from data sparseness
- The weights can be set using the Expectation-Maximization Algorithm or another numerical optimization technique

Linear Interpolation

$$\begin{aligned} P'(w_i|w_{i-2}, w_{i-1}) = \\ &\lambda_1 * P_{ML}(w_i|w_{i-2}, w_{i-1}) \\ &+\lambda_2 * P_{ML}(w_i|w_{i-1}) \\ &+\lambda_3 * P_{ML}(w_i) \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i.

Linear Interpolation

This estimate defines distribution:

$$\sum_{w \in \nu} P'(w_i|w_{i-2}, w_{i-1})$$

$$\begin{aligned}
&= \sum_{w \in \nu} [\lambda_1 * P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 * P_{ML}(w_i|w_{i-1}) + \\
&\quad \lambda_3 * P_{ML}(w_i)] \\
&= \lambda_1 * \sum_{w \in \nu} P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 * \sum_{w \in \nu} P_{ML}(w_i|w_{i-1}) + \\
&\quad \lambda_3 * \sum_{w \in \nu} P_{ML}(w_i) \\
&= \lambda_1 + \lambda_2 + \lambda_3 = 1
\end{aligned}$$

Parameter Estimation

- Hold out part of training set as “validation” data
- Define $Count_2(w_1, w_2, w_3)$ to be the number of times the trigram (w_1, w_2, w_3) is seen in validation set
- Choose λ_i to maximize:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3 \in \nu} Count_2(w_1, w_2, w_3) \log P'(w_3 | w_1, w_2)$$

such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and λ_i for all i.

An Iterative Method

Initialization: Pick arbitrary/random values for $\lambda_1, \lambda_2, \lambda_3$

Step 1: Calculate the following quantities:

$$c_1 = \sum_{w_1, w_2, w_3 \in \nu} \frac{Count_2(w_1, w_2, w_3)\lambda_1 * P_{ML}(w_3|w_2, w_1)}{\lambda_1 * P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 * P_{ML}(w_i|w_{i-1}) + \lambda_3 * P_{ML}(w_i)}$$

$$c_2 = \sum_{w_1, w_2, w_3 \in \nu} \frac{Count_2(w_1, w_2, w_3)\lambda_2 * P_{ML}(w_3|w_2)}{\lambda_1 * P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 * P_{ML}(w_i|w_{i-1}) + \lambda_3 * P_{ML}(w_i)}$$

$$c_3 = \sum_{w_1, w_2, w_3 \in \nu} \frac{Count_2(w_1, w_2, w_3)\lambda_3 * P_{ML}(w_3)}{\lambda_1 * P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 * P_{ML}(w_i|w_{i-1}) + \lambda_3 * P_{ML}(w_i)}$$

Step 2: Reestimate: $\lambda_i = \frac{c_i}{c_1+c_2+c_3}$

Step 3: If λ_i have not converged, go to Step 1.

Allowing the λ's to vary

- Partition histories (for instance, based on frequencies)

$$\Phi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & if\ Count(w_{i-2}, w_{i-1}) = 0 \\ 2 & if\ 1 \leq Count(w_{i-2}, w_{i-1}) \leq 3 \\ 5 & Otherwise \end{cases}$$

- Condition λ's on partitions

$$\begin{aligned} P'(w_i|w_{i-2}, w_{i-1}) = & \\ & \lambda_1^{\Phi(w_{i-2},w_{i-1})} * P_{ML}(w_i|w_{i-2}, w_{i-1}) \\ & +\lambda_2^{\Phi(w_{i-2},w_{i-1})} * P_{ML}(w_i|w_{i-1}) \\ & +\lambda_3^{\Phi(w_{i-2},w_{i-1})} * P_{ML}(w_i) \end{aligned}$$

where $\lambda_1^{\Phi(w_{i-2},w_{i-1})} + \lambda_2^{\Phi(w_{i-2},w_{i-1})} + \lambda_3^{\Phi(w_{i-2},w_{i-1})} = 1$, and $\lambda_i^{\Phi(w_{i-2},w_{i-1})} \geq 0$ for all i.

Katz Back-Off Models (Bigrams)

- Define two sets

$$A(w_{i-1}) = \{w : Count(w_{i-1}, w) > 0\}$$

$$B(w_{i-1}) = \{w : Count(w_{i-1}, w) = 0\}$$

- A bigram model

$$P_K(w_i|w_{i-1}) = \begin{cases} \frac{Count^\star(w_{i-1}, w)}{Count(w_{i-1})} > 0 & if\ w_i \in A(w_{i-1}) \\ \alpha(w_{i-1}) \frac{P_{ML}(w_i)}{\sum_{w \in B(w_{i-1})} P_{ML}(w)} & if\ w_i \in B(w_{i-1}) \end{cases}$$

$$\alpha(w_{i-1}) = 1 - \sum_{w \in A(w_{i-1})} \frac{Count^\star(w_{i_1}, w)}{Count(w_{i-1})}$$

$Count^\star$ definitions

- Katz uses Good-Turing method for $Count(x) < 5$, and $Count^\star(x) = Count(x)$ for $Count(x) \geq 5$
- "Kneser-Ney" method:

$$Count^\star(x) = Count(x) - D,\ whereD = \frac{n_1}{n_1 + n_2}$$

n_1 is a number of elements with frequency 1
n_2 is a number of elements with frequency 2

Weaknesses of n-gram Models

Any ideas?

- Short-range
- Mid-range
- Long-range

More Refined Models

- Class-based models
- Structural models
- Topical and long-range models

Summary

- Start with a vocabulary
- Select type of model
- Estimate Parameters

Consider using CMU-Cambridge language modeling toolkit:

`http://mi.eng.cam.ac.uk/~prc14/toolkit.html`