Smoothing and interpolation

Micha Elsner

January 9, 2014
Estimation for the bigram model needs *smoothing* (regularization)
  - ...cope with sparse data
  - Simple add-$\lambda$ smoother

\[
\hat{P}(w_{i+1} = b | w_i = a) = \frac{\#(a \cdot b) + \lambda}{\#(a \cdot) + \lambda |V|}
\]
Log-likelihood curve for a bigram model

This isn’t quite the one I’m outlining now—I used my code for Assignment 1, which is a bit more complex

Finding optimal $\lambda$: line search
Line search
Mathematical optimization

Optimization
Mathematical/computer science techniques for solving \textit{max} or \textit{min} problems

- A large library of techniques
- We’ll cover some basic ones

\textit{Objective}: what you are maximizing/minimizing

Key insight: the \textit{objective} is separate from the optimization method!

- Be clear about \textit{what you want}
- Then think about \textit{how to find it}

Two methods so far: ratio-of-counts (a \textit{closed form estimate})
one-dimensional line search (an \textit{iterative} method)
Unknown words

Issue: how big is vocab set $V$ anyway?

- Common to take set of words in training data as vocabulary
- Plus “unknown word” symbol $<UNK>$
  - This symbol is assigned to all words not appearing in train
- Of course it has no training counts...

- Train: The cat is on the mat.
- Test: The dog is on the sofa.
- $\rightarrow$ Transformed test: The $<UNK>$ is on the $<UNK>$
More complicated UNK and vocabulary systems

- Possible to “spell out” unknown words letter by letter
  - The d.o.g is on the s.o.f.a
  - The letter probabilities can be fit with more held-out data
- Can have multiple kinds of \textlt{UNK}\rt based on morphophonology or syntactic category
  - For instance, what about numbers?
  - What are some other examples?
More complicated UNK and vocabulary systems

- Possible to “spell out” unknown words letter by letter
  - The d.o.g is on the s.o.f.a
  - The letter probabilities can be fit with more held-out data
- Can have multiple kinds of \(<\text{UNK}\)> based on morphophonology or syntactic category
  - For instance, what about numbers?
  - What are some other examples?
- One simple way to do this (Stanford/Berkeley): orthographic shape
- Where are capital letters, punctuation, numbers?
  - cassowary = “c”
  - Stanford = “Cc”
  - “3G-related” = “1C-c”
In order to use these more sophisticated UNK methods, it helps to learn some parameters for UNK

- Already discussed using held-out data
- Alternately, we can use rare items from the data we have
- Transform very rare words into \(<\text{UNK}>\)
  - We can also keep them as themselves... just count them twice?
- For instance, all words that appear only once in training
- These words *could* easily be unseen...
  - So they’re probably similar to words that are actually unseen
Good-Turing estimates

From work at Bletchley in WWII
Basic idea: estimate number of unseen items using trend from:

▸ 1-count objects, 2-count items, 3-count items... etc
▸ For instance, if there are 100 2-count items
▸ And 200 1-count items
▸ How many 0-count items should we expect?
▸ Not a fully-specified algorithm (must decide how to model the trend)

Transforming rare words into UNK makes the assumption that there are as many 0-count words as 1-count words...

▸ This is unsophisticated, but easy to implement
Problems with the add-\(\lambda\) smoother

- The add-\(\lambda\) smoother can behave oddly
- What follows a rare context word: eg *depart*, *deepen*?
This is a problem...

- Our own intuitions about rare words—can usually make some guesses
- Makes it impossible to build high-order n-gram models
  - Data sparsity gets worse as grams get longer...
  - More and more data in the long tail
  - Count for “to eat a fish” ≤ “eat a fish” ≤ “a fish”
  - More complex model overfits more...
A better idea

- Fall back on unigram estimates
- “the” is really common, so perhaps “depart the” and “deepen the” also common?
- Could explicitly switch models (“backoff”) but common to merge estimates together

\[
\hat{P}(w_{i+1} = b|w_i = a) = \frac{\#(a \ b) + \beta P_{\text{unigram}}(w_{i+1})}{\#(a \bullet) + \beta}
\]

- \(\beta\) is total number of pseudocounts (ie, \(\lambda|V|\))
- \(P_{\text{unigram}}\) determines how we split up the pseudocounts
  - Split up *unfairly* so more probable items get more of \(\beta\)
- If \(P_{\text{unigram}}\) uniform, we recover our \(\lambda\) scheme
Interpolation

- Estimate multiple distributions
- Combine so that final estimate is in between

Notice that for $0 \leq \lambda \leq 1$ and probabilities $P_1, P_2$, the following is a probability (linear interpolation):

$$P(x) = \lambda P_1(x) + (1 - \lambda)P_2(x)$$

$$\hat{P}(w_{i+1} = b|w_i = a) = \frac{\#(a \ b) + \beta P_{\text{unigram}}(w_{i+1})}{\#(a \bullet) + \beta}$$

This one is sometimes called *Dirichlet* smoothing
- Is a particular kind of interpolation
- Name comes from a derivation via Bayesian statistics
Estimates for unigrams

Where does $P_{unigram}$ come from?
Estimates for unigrams

Where does $P_{\text{unigram}}$ come from?

$$\hat{P}(w_i = b) = \frac{\#(b) + \lambda}{\#(\bullet) + \lambda|V|}$$

(Note Charniak uses $\alpha$ instead of $\lambda$)

Turns out to be a special case of the Dirichlet smoothing equation.
Proof: skip this

We want to show that add-$\lambda$ is a special case of the Dirichlet smoother (when $P_u$ is uniform)

\[
\frac{\#(a \ b) + \beta P_{\text{unigram}}(w_{i+1})}{\#(a \cdot) + \beta} = \frac{\#(a \ b) + \lambda}{\#(a \cdot) + \lambda|V|}
\]

$P_u$ uniform means $P_u(w) = \frac{1}{|V|}$ so:

\[
\frac{\#(a \ b) + \beta \frac{1}{|V|}}{\#(a \cdot) + \beta} = \frac{\#(a \ b) + \lambda}{\#(a \cdot) + \lambda}
\]

Choose $\lambda = \frac{\beta}{|V|}$, thus $\beta = \lambda|V|$

\[
\frac{\#(a \ b) + \lambda|V|\frac{1}{|V|}}{\#(a \cdot) + \lambda|V|} = \frac{\#(a \ b) + \lambda}{\#(a \cdot) + \lambda|V|}
\]

And cancel $|V|\frac{1}{|V|}$, QED
Some example effects

depart

one real instance in train (“depart from”)
Effect of smoothing with unigrams vs add-$\lambda$

▶ depart the: .058 vs 6e-5
▶ depart from: .01 vs .007
▶ depart of: .03 vs 6e-5
▶ depart jewel: 63-6 vs 6e-5
▶ depart French: .0001 vs 6e-5

the

▶ the senate: .04 vs .04
▶ the the: .0002 vs 3e-5
▶ the jewel: 1e-8 vs 3e-7
High-order ngram models

This kind of smoothing enables us to go beyond bigrams:

► For instance, possible to build a practical *trigram* (3-gram) model from a moderate-size dataset

► Construction is pretty simple...

\[
\hat{P}(w_{i+1} = c | w_i = a, w_{i-1} = b) = \frac{\#(a \ b \ c) + \beta P_{\text{bigram}}(w_{i+1} | w_i)}{\#(a \ b \cdot) + \beta}
\]

...and so on, recursively

► *P_{\text{bigram}}* based on *P_{\text{unigram}},* etc
The assignment

This basically covers the whole assignment.

- Read in the training data to compute unigram and bigram counts
- Compute the dev data log-likelihood for various values of $\alpha$ ($\lambda$)
  - Remember to replace unknown words from dev with $<UNK>$
- Construct a bigram model using your best unigram model for smoothing
- Compute the dev data log-likelihood for various values of $\beta$
- Use the learned probabilities to make decisions in the good-bad dataset
An LM minipresentation

This is an example minipresentation of a paper on language models

- What is the task the paper is about?
- How does the technology (ie LMs) apply?
- Are there any novel extensions or clever tricks?
  - If so, one sentence about what they seem to be doing
- What are the results like?
Language modeling for determiner selection

Jenine Turner and Eugene Charniak, NAACL 2007

- The task: adding determiners (“a/an”, “the”) to sentences
  - Useful for machine translation, eg from Chinese
- Use LM to check probabilities with all possible determiners
  - “Kim bought the car” vs “Kim bought a car” vs “Kim bought car”
- LM is actually based on syntax, not just ngrams
  - We’ll cover this kind of model in the parsing section

Evaluation:

- It’s important to be fair in selecting test data
- Selecting sentences at random allows sentences from the same article to be in train and test
- This inflates results, since the article usually uses the same NPs
Results

This approach 86.3% accurate

Some errors:

- The computers were crude by the today’s standards
- In addition the Apple II was the/an affordable $1298

Some “acceptable” errors:

- Crude as they were, these early PCs triggered an explosive product development in desktop models
- Highway officials insist the ornamental railings on old bridges aren’t strong enough

Sometimes you need to know semantics

- IBM, a/the world leader in computers, didn’t offer its first PC until 1981
Modern language models: Kneser-Ney smoothing

Kneser-Ney smoothing

- Still a competitive algorithm for midsize datasets
- Popular modification introduced by Chen and Goodman 99
  - On Carmen if you want to read it
- Hierarchical Bayesian explanation recently given
- Teh and Goldwater/Griffiths each in 2006
Discounting estimators

So far, we’ve looked at smoothing by *adding* pseudo-observations

- the “fake” training counts are all strictly larger than the real counts

\[
\hat{P}(w_{i+1} = b | w_i = a) = \frac{\#(a b)}{\#(a \bullet) + \beta} + \frac{\beta}{\#(a \bullet) + \beta} P_{uni}(b)
\]

Kneser-Ney is based on *discounting*

- we subtract counts from real items and distribute them as pseudocounts

\[
\hat{P}(w_{i+1} = b | w_i = a) = \frac{\text{max}(0, \#(a b) - d)}{\#(a \bullet)} + \frac{D_a}{\#(a \bullet)} P_{uni}(b)
\]

and \(D_a = \sum_b \text{min}(d, \#(a b))\)
Why discounting?

Some words need more smoothing than others

- “San” (a few city names: Francisco, Diego)
- vs “green” (nearly anything: car, house, paint...)

- Counts for “green” are spread out
  - And we anticipate many new possibilities
- For “San”, concentrated on a few options
  - And we anticipate few more

\[ D_a = \sum_{b} \min(d, \#(a \ b)) \]
Kneser-Ney’s $P_{uni}$

We can drive the intuition behind discounting further

- Our intuition then: some words are very productive contexts
  - Others restrict what appears next
- Same idea: some words appear in many contexts
  - While others are very restricted
- “Francisco” (in US English) usually appears after San
- “report” appears all over the place

$$
\hat{P}_{KNuni}(b) = \frac{\sum_a 1 \text{if } (a \ b) \text{ ever appears}}{\sum_{ax} 1 \text{if } (a \ x) \text{ ever appears}}
$$

Proportion of unique word types appearing before $b$
Very large datasets

“More data is better data”

Google 5-grams

- File sizes: approx. 24 GB compressed (gzip’ed) text files
- Number of tokens: 1,024,908,267,229
- Number of sentences: 95,119,665,584
- Number of unigrams: 13,588,391
- Number of bigrams: 314,843,401
- Number of trigrams: 977,069,902
- Number of fourgrams: 1,313,818,354
- Number of fivegrams: 1,176,470,663

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
Dealing with massive data

Primary challenges are algorithmic

- How do you store this stuff?
- How do you query it?
- Subfield of CL devoted to this— very mathematical
  - Often enough to get *approximate* counts
  - May only need to know if count is zero or nonzero
  - Or save space by clustering/hashing similar words

Basic techniques (smoothing by interpolating high and low-order estimates) haven’t changed much
More exotic LMs

- An LM is just a distribution over strings
  - Doesn’t have to break down by n-grams
- Possible to incorporate more interesting syntax/semantics/etc
- ...Worth thinking about how the rest of the course can inform LMs
  - and people are trying
- But the tools we’ve covered so far aren’t sophisticated enough