Language Model

Outline

- Definitions of Language Model (LM)
- Applications of LM, how it is useful.
- Computing (estimating) probabilites.
- Evaluating LMs
- Problems of spare data
  - Smoothing techniques
- Tools

Language Modeling

- We want to compute

  $P(w1,w2,\ldots,wn)$, the probability of a sequence
- Alternatively, we want to compute

  $P(wn|w1,w2,\ldots,wn-1)$, the probability of a word given some previous words
- The model, that computes $P(W)$ or $P(wn|w1,\ldots,wn-1)$ is called the language model.

Shannon's game

- Game's rule:
  - Given a sequence of word: $w_1 w_2 \ldots w_n$
  - Predict the next word?

- For example:
  - Anh ấy là một nhà khoa ___

  - Hắn cắm cúi chép vào __
    - Hắn cắm cúi chép vào A
    - Hắn cắm cúi chép vào B

Noise Channel

- Noise channel

0,1,1,1,0,1,0,1,…　　　　0,1,1,<u>0</u>,0,1,<u>1,0</u>,…

- Model: probability of error (noise):
- Example: $p(0|1) = .3$ $p(1|1) = .7$ $p(1|0) = .4$ $p(0|0) = .6$

- The task: from the *noisy output*, we have to recover the origin (*input*) -> This process is called ***Decoding***

Applications of noise channel

- OCR (optical character recognition)
  - straightforward: text → print (adds noise), scan → image
- Handwriting recognition (HR)
  - text → neurons, muscles (“noise”), scan/digitize → image
- Speech recognition- ASR (dictation, commands, etc.)
  - text → conversion to acoustic signal (“noise”) → acoustic waves
- Machine Translation - MT
  - text in target language → translation (“noise”) → source language
- Also: Part of Speech Tagging
  - sequence of tags → selection of word forms → text

Noisy Channel: The Golden Rule of OCR, ASR, HR, MT, ...

- Recall:

$$p(A|B) = p(B|A)\ p(A) / p(B) \quad \text{(Bayes formula)}$$

$$A_{best} = \text{argmax}_A\ p(B|A)\ p(A) \quad \text{(The Golden Rule)}$$

- p(B|A): the acoustic/image/translation/lexical model
  - application-specific name
  - will explore later
- p(A): ***the language model***

The Chain rule

$$W = (w_1, w_2, w_3, \ldots, w_d)$$

compute $\quad p(W) = ?$

- Use the Chain rule:

$$p(W) = p(w_1, w_2, w_3, \ldots, w_d) =$$

$$= p(w_1) \times p(w_2|w_1) \times p(w_3|w_1,w_2) \times \ldots \times p(w_d|w_1,w_2,\ldots,w_{d-1})$$

Problems

- There are a lot of possible sentences
- In general, we'll never be able to get enough data to compute the statistics for those long prefixes

Markov assumption (1)

- The perfect model: *without limitation of memory*
  - $\mathbf{w_i}$ -> know all previous words: $\mathbf{w_1, w_2, w_3, \ldots, w_{i-1}}$
- *With limitation of memory*:
  - Ignore the too far previous words (“too old” predecessors)
  - Just depends on the ***k*** nearest words: $\mathbf{w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}}$
  - “k^{th} order Markov approximation”

$$\mathbf{p(W) \cong \Pi_{i=1..d}\, p(w_i | w_{i-k}, w_{i-k+1}, \ldots, w_{i-1}),\ d = |W|}$$

Markov Assumption (2)

So for each component in the product replace with the approximation (assuming a prefix of N)

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

Bigram version

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

N-gram models

- Markov approximation with order (n-1) $\rightarrow$ n-gram LM:

prediction history

$$p(W) =_{df} \Pi_{i=1..d} p(w_i | w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1})$$

- Size of vocabulary |V| = 60k:
  - 0-gram LM: uniform model, $p(w) = 1/|V|$, 1 parameter
  - 1-gram LM: unigram model, $p(w)$, 6×10^4 parameters
  - 2-gram LM: bigram model, $p(w_i|w_{i-1})$ 3.6×10^9 parameters
  - 3-gram LM: trigram model, $p(w_i|w_{i-2},w_{i-1})$ 2.16×10^{14} parameters

LM: observations

- How large *n*?
  - Nothing is enough (theoretically)
  - But anyway: as much as possible (→ close to "perfect" model)
  - Empirically: **3**
    - parameter estimation? (reliability, data availability, storage space, ...)
    - 4 is too much: $|V|$=60k → 1.296×10^{19} parameters
    - but: 6-7 would be (almost) ideal (having enough data): *in fact, one can recover original from 7-grams!*

- For now, keep word forms (no "linguistic" processing)

Estimate parameters

- Parameter: the necessary values to compute $p(w|h)$
- Get from: text
- Preparing the data:
  - text
  - Define words: separate words …
  - Definition of sentences (insert “words” <s> and </s>)
  - Capitals: keep, discard, or be smart:
    - Nhận dạng tên riêng
    - Định nghĩa các kiểu số
  - numbers: keep, replace by <num>, or be smart (form ~ pronunciation)

Maximum Likelihood Estimate

- Trigrams from Training Data T:
  - count sequences of three words in T: $c_3(w_{i-2},w_{i-1},w_i)$
  - count sequences of two words in T: $c_2(w_{i-1},w_i)$:
    - either use $c_2(y,z) = \Sigma_w\, c_3(y,z,w)$
    - or count differently at the beginning (& end) of data!

$$\boxed{p(w_i|w_{i-2},w_{i-1}) =_{est.} c_3(w_{i-2},w_{i-1},w_i) \;/\; c_2(w_{i-2},w_{i-1})}\ !$$

Estimate bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

ML estimates

- The maximum likelihood estimate of some parameter of a model M from a training set T.
  - Is the estimate that maximizes the likelihood of the training set T give the model M
- Suppose the word "Chinese" occurs 400 times in a corpus of a million words (Brown corpus).
- What is the probability that a random word (from some other text from the same distribution) will be "Chinese"
- MLE estimate is 400/1000000 = 0.04
  - This may be a bad estimate for some other corpus
- But it is the esimate that makes it most likely that "Chinese" will occur 400 times in a million word corpus

An example

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

An example

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

Berkeley Restaurant Project Sentences

- *Can you tell me about any good cantonese restaurants close by mid priced thai food is what i'm looking for.*
- *Tell me about chez panisse.*
- *Can you give me a listing of the kinds of food that are available.*
- I *'m looking for a good place to eat breakfast.*
- *When is caffe venezia open during the day*
- ….

Raw Bigram counts

- Out of 9222 sentences: Count(col | row)

	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

Raw Bigram Probabilities

- Normalize by unigrams:

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	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 Estimates of Sentence Probabilities

- P(<s> I want english food </s>) =

  p(i|<s>) x p(want|I) x p(english|want)
  x p(food|english) x p(</s>|food)
  =.000031

Kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

- World knowledge
- Syntax
- Discourse

Evaluation

- We train parameters of our model on a **training set.**
- How do we evaluate how well our model works?
- We look at the models’ performance on some new data
- This is what happens in the real world; we want to know how our model performs on data we haven’t seen
- So a **test set. A dataset which is different than our** training set

Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a speech recognizer
  - Run recognition, get Word Error Rate (WER) for A
  - Put model B in speech recognition, get WER for B
  - Compare WER for A and B
  - Called **Extrinsic Evaluation** (application-based evaluation)

Difficulty of extrinsic evaluation

- Extrinsic evaluation
  - It is specific to the application
    - It is usually called application dependent evaluation
    - It is no evidence to be sure how it is good to other applications
  - This is really time-consuming
- So how to independent/self evaluation
  - Use an intrinsic evaluation called **perplexity**
    - Based on a test data
    - It is good if the test data looks like the training data

Perplexity

- Perplexity is the probability of the test set (assigned by the LM), normalized by the number of words

$$\mathrm{PP}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

For bigrams: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- Minimizing perplexity is the same as maximizing probability.
  - The best language model is one that best predicts an unseen test set.

Perplexity

- There is another way to think about perplexity, as the weighted avarage branching fact (or of a language).
- The branching factor of a language is the number of possible next words that can follow any word.
- For example:
  - Training unigram, bigram, and trigram on 38 million words from Wall Street Journal, using 19,979 word vocabulary.
  - Test set of 1.5 million words.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Problems: the perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models, adapt to test set, etc

Problem: zeros or nots?

- Zipf's Law:
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! no counts at all for the vast bulk of things we want to estimate!
  - Some of the zeroes in the table are really zeros But others are simply low frequency events you haven't seen yet. After all, ANYTHING CAN HAPPEN!
  - How to address?
- Answer:
  - Estimate the likelihood of unseen N-grams!

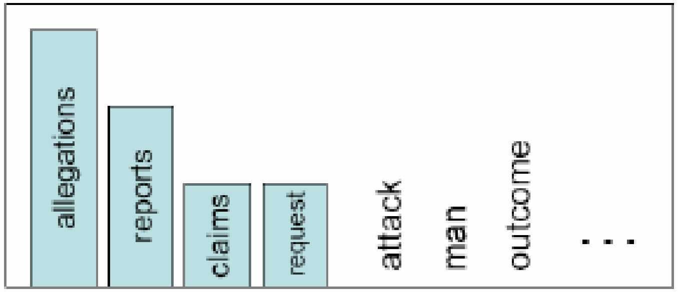
Zero problem

- Two kinds of zeros: $p(w|h) = 0$, or even $p(h) = 0$!
- Indeterminate:
  - happens when an event is found in test data which has not been seen in training data
- To make the system more robust
  - low count estimates:
    - they typically happen for “detailed” but relatively rare appearances
  - high count estimates: reliable but less “detailed”

Smoothing is like Robin Hood: Steal from the rich and give to the poor (in probability mass)

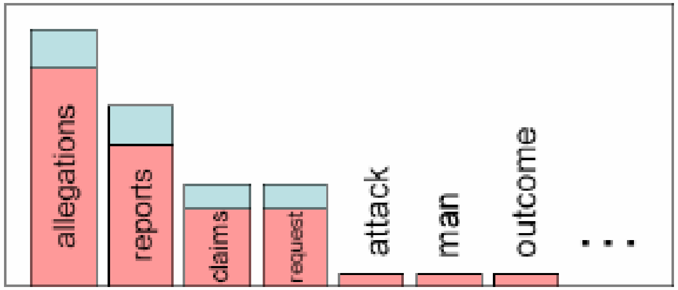
- We often want to make predictions from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Very important all over NLP, but easy to do badly!

Remove zero probability: smoothing

- Get new p'(w) (same Ω): almost p(w) but no zeros
- Discount w for (some) p(w) > 0: new p'(w) < p(w)

$$\Sigma_{w \in \text{discounted}} (p(w) - p'(w)) = D$$

- Distribute D to all w; p(w) = 0: new p'(w) > p(w)
  - possibly also to other w with low p(w)
- Make sure $\Sigma_{w \in \Omega} p'(w) = 1$
- There are many ways of ***smoothing***

Laplace smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple

$$P(w_i) = \frac{c_i}{N}$$

- MLE estimate:

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Laplace estimate:

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

- Reconstructed counts:

Raw Bigram counts

- Out of 9222 sentences: Count(col | row)

	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

Laplace smoothed 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

Laplace-smoothed bigrams

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

	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

Big changes to count

- C(count to) went from 608 to 238!
- P(to|want) from .66 to .26!
  - Discount d= c*/c
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge.

Lidstone's & Jeffreys-Perks Laws

Because Laplace's law overestimates non-zero events, variations were created:

- Lidstone's law: instead of adding one, add some smaller value λ

  (6) $P(w_1 ... w_n) = \frac{C(w_1 ... w_n) + \lambda}{N + B\lambda}$

- Jeffreys-Perks law: set λ to be $\frac{1}{2}$ (the expectation of maximized MLE):

  (7) $P(w_1 ... w_n) = \frac{C(w_1 ... w_n) + \frac{1}{2}}{N + \frac{1}{2}}$

Problems: How do we guess λ? And still not good for low frequency n-grams

Better discounting methods

- used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell
- use the count of things we’ve seen once to help estimate the count of things we’ve never seen

Good Turing

- Imagine you are fishing
- There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- •You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel
  - = 18 fish (tokens)
  - = 6 species (types)
- How likely is it that you'll next see another trout?

Good-Turing

- Now how likely is it that next species is new
  - There were 18 distinct events… 3 of those represent singleton species.

Good-Turning

- Idea: use the count of things you've seen ***once*** to estimate count of things you've ***never seen***.
- Denote that: N_c is the number of N-gram which apears c time: N_0 for the number of N-gram appearing 0 time, N_1 -> appearing 1 time, …

$$N_c = \sum_{x:count(x)=c} 1$$

- The Good-Turning estimates the smoothed count c* of c base on N_c as follows:

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

Good-Turning

- similar idea: discount/boost the relative frequency estimate:

$$p(w) = c(w) / T$$

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

$$p(w) = ((c(w)+1) * \frac{N_{c(w)+1}}{N_{c(w)}}) / T$$

- specifically, for c(w) = 0 (unseen words):

$$p_r(w) = N(1) / (|T| \times N(0))$$

Good-Turning

- Example: remember: $p_r(w) = (c(w) + 1) \times N(c(w) + 1) / (|T| \times N(c(w)))$

  Training data: <s> what is it what is small ? $|T| = 8$

  - V = { what, is, it, small, ?, <s>, flying, birds, are, a, bird, . }, $|V| = 12$
    p(it)=.125, p(what)=.25, p(.)=0 p(what is it?) = $.25^2 \times .125^2 \cong .001$
    p(it is flying.) = $.125 \times .25 \times 0^2 = 0$

  - Raw reestimation (N(0) = 6, N(1) = 4, N(2) = 2, N(i) = 0 for i > 2):
    $p_r(it) = (1+1) \times N(1+1)/(8 \times N(1)) = 2 \times 2/(8 \times 4) = .125$
    $p_r(what)=(2+1) \times N(2+1)/(8 \times N(2))=3 \times 0/(8 \times 2)=0$: keep orig. p(what)
    $p_r(.) = (0+1) \times N(0+1)/(8 \times N(0)) = 1 \times 4/(8 \times 6) \cong .083$

  - Normalize (divide by $1.5 = \Sigma_{w \in |V|} p_r(w)$) and compute:
    $p'(it) \cong .08$, $p'(what) \cong .17$, $p'(.) \cong .06$

$p'(\text{what is it?}) = .17^2 \times .08^2 \cong .0002$
$p'(\text{it is flying.}) = .08 \times .17 \times .06^2 \cong .00004$

Smoothing by Combination: Linear Interpolation

- Weight in less detailed distributions using $\lambda=(\lambda_0,\lambda_1,\lambda_2,\lambda_3)$:

  $$p'_\lambda(w_i | w_{i-2}, w_{i-1}) = \lambda_3\, p_3(w_i | w_{i-2}, w_{i-1}) + \lambda_2\, p_2(w_i | w_{i-1}) + \lambda_1\, p_1(w_i) + \lambda_0 / |V|$$

- Normalize:

  $\lambda_i > 0$, $\Sigma_{i=0..n} \lambda_i = 1$ is sufficient ($\lambda_0 = 1 - \Sigma_{i=1..n} \lambda_i$) ($n=3$)

- Estimation using MLE:
  - fix the p_3, p_2, p_1 and $|V|$ parameters as estimated from the training data
  - then find such $\{\lambda_i\}$ which minimizes the cross entropy (maximizes probability of data): $-(1/|D|)\Sigma_{i=1..|D|}\log_2(p'_\lambda(w_i|h_i))$

Held-out Data

- What data to use?
  - try the training data T: but we will always get $\lambda_3 = 1$
    - why? (let p_{iT} be an i-gram distribution estimated using relative freq. from T)
    - minimizing $H_T(p'_\lambda)$ over a vector λ, $p'_\lambda = \lambda_3 p_{3T} + \lambda_2 p_{2T} + \lambda_1 p_{1T} + \lambda_0/|V|$
      - remember: $H_T(p'_\lambda) = H(p_{3T}) + D(p_{3T}||p'_\lambda)$; ($p_{3T}$ fixed $\rightarrow H(p_{3T})$ fixed, best)
      - which p'_λ minimizes $H_T(p'_\lambda)$? Obviously, a p'_λ for which $D(p_{3T}|| p'_\lambda)=0$
      - ...and that's p_{3T} (because $D(p||p) = 0$, as we know).
      - ...and certainly $p'_\lambda = p_{3T}$ if $\lambda_3 = 1$ (maybe in some other cases, too).
      - $(p'_\lambda = 1 \times p_{3T} + 0 \times p_{2T} + 0 \times p_{1T} + 0/|V|)$
  - thus: do <u>not use the training data for estimation of λ!</u>
    - must hold out part of the training data (***heldout*** data, <u>H</u>):
    - ...call the remaining data the (true/raw) ***training*** data, <u>T</u>
    - the ***test*** data <u>S</u> (e.g., for comparison purposes): still different data!

The Formula

- Repeat: minimizing $-(1/|H|)\Sigma_{i=1..|H|}\log_2(p'_\lambda(w_i|h_i))$ over λ

$$p'_\lambda(w_i|\ h_i) = p'_\lambda(w_i|\ w_{i-2}, w_{i-1}) = \lambda_3\, p_3(w_i|\ w_{i-2}, w_{i-1}) + \lambda_2\, p_2(w_i|\ w_{i-1}) + \lambda_1\, p_1(w_i) + \lambda_0 / |V|$$

- "Expected Counts (of lambdas)": $j = 0..3$ – next page

$$c(\lambda_j) = \Sigma_{i=1..|H|}\ (\lambda_j p_j(w_i|h_i)\ /\ p'_\lambda(w_i|h_i))$$

- "Next λ": $j = 0..3$

$$\lambda_{j,next} = c(\lambda_j)\ /\ \Sigma_{k=0..3}\ (c(\lambda_k))$$

Example

- Raw distribution (unigram only; smooth with uniform):
  $p(a) = .25$, $p(b) = .5$, $p(\alpha) = 1/64$ for $\alpha \in \{c...r\}$, $= 0$ for the rest: s,t,u,v,w,x,y,z
- Heldout data: <u>baby</u>; use one set of λ (λ_1: unigram, λ_0: uniform)
- Start with $\lambda_1 = .5$; $p'_\lambda(b) = .5 \times .5 + .5 / 26 = .27$
  $p'_\lambda(a) = .5 \times .25 + .5 / 26 = .14$
  $p'_\lambda(y) = .5 \times 0 + .5 / 26 = .02$

$c(\lambda_1) = .5x.5/.27 + .5x.25/.14 + .5x.5/.27 + .5x0/.02 = 2.72$

$c(\lambda_0) = .5x.04/.27 + .5x.04/.14 + .5x.04/.27 + .5x.04/.02 = 1.28$

Normalize: $\lambda_{1,next} = .68$, $\lambda_{0,next} = .32$.

Repeat from step 2 (recompute p'_λ first for efficient computation, then $c(\lambda_i)$, ...)

Finish when new lambdas almost equal to the old ones (say, < 0.01 difference).

The Problem

- Not enough data
  - Language Modeling: we do not see "correct" n-grams
    - solution so far: smoothing
  - suppose we see:
    - short homework, short assignment, simple homework
  - but not:
    - simple assigment
  - What happens to our (bigram) LM?
    - p(homework | simple) = high probability
    - p(assigment | simple) = low probability (smoothed with p(assigment))
- They should be much closer!

Word Classes

- Observation: similar words behave in a similar way
  - trigram LM:
    - in the … (all nouns/adj);
    - catch a … (all things which can be catched, incl. their accompanying adjectives);
  - trigram LM, conditioning:
    - a … homework (any atribute of homework: short, simple, late, difficult),
    - … the woods (any verb that has the woods as an object: walk, cut, save)
  - trigram LM: both:
    - a (short,long,difficult,…) (homework,assignment,task,job,…)

Solution

- Use the Word Classes as the “reliability” measure
- Example: we see
  - short homework, short assignment, simple homework
  - but not:
    - simple assigment
  - Cluster into classes:
    - (short, simple) (homework, assignment)
      - covers “simple assignment”, too
- Gaining: realistic estimates for unseen n-grams
- Loosing: accuracy (level of detail) within classes

The New Model

- Rewrite the n-gram LM using classes:
  - Was: [k = 1..n]
    - $p_k(w_i|h_i) = c(h_i,w_i) / c(h_i)$ [history: (k-1) <u>words</u>]
  - Introduce classes:

$$p_k(w_i|h_i) = p(w_i|c_i)\, p_k(c_i|h_i)$$

    - history: <u>classes</u>, too: [for trigram: $h_i = c_{i-2},c_{i-1}$, bigram: $h_i = c_{i-1}$]
  - Smoothing as usual
    - over $p_k(w_i|h_i)$, where each is defined as above (except uniform which stays at $1/|V|$)

Training Data

- Suppose we already have a mapping:
  - r: V → C assigning each word its class ($c_i = r(w_i)$)
- Expand the training data:
  - $T = (w_1, w_2, \ldots, w_{|T|})$ into
  - $T_C = (\langle w_1,r(w_1)\rangle, \langle w_2,r(w_2)\rangle, \ldots, \langle w_{|T|},r(w_{|T|})\rangle)$
- Effectively, we have two streams of data:
  - word stream: $w_1, w_2, \ldots, w_{|T|}$
  - class stream: $c_1, c_2, \ldots, c_{|T|}$ (def. as $c_i = r(w_i)$)
- Expand Heldout, Test data too

Training the New Model

- As expected, using ML estimates:
  - $p(w_i|c_i) = p(w_i|r(w_i)) = c(w_i) / c(r(w_i)) = c(w_i) / c(c_i)$
    - !!! $c(w_i,c_i) = c(w_i)$ [since c_i determined by w_i]
  - $p_k(c_i|h_i)$:
    - $p_3(c_i|h_i) = p_3(c_i|c_{i-2}, c_{i-1}) = c(c_{i-2}, c_{i-1}, c_i) / c(c_{i-2}, c_{i-1})$
    - $p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = c(c_{i-1}, c_i) / c(c_{i-1})$
    - $p_1(c_i|h_i) = p_1(c_i) = c(c_i) / |T|$
- Then smooth as usual
  - not the $p(w_i|c_i)$ nor $p_k(c_i|h_i)$ individually, but the $\mathbf{p_k(w_i|h_i)}$

Classes: How To Get Them

- We supposed the classes are given
- Maybe there are in [human] dictionaries, but…
  - dictionaries are incomplete
  - dictionaries are unreliable
  - do not define classes as equivalence relation (overlap)
  - do not define classes suitable for LM
    - small, short… maybe; small and difficult?
- → we have to construct them <u>from data</u> (again…)

Toolkits and Open sources

- You can make your own language models with tools freely available for research
- CMU language modeling toolkit
  - http://www.speech.cs.cmu.edu/SLM_info.html
- SRI language modeling toolkit
  - http://www-speech.sri.com/projects/srilm/