Language Model

Palacode Narayana Iyer Anantharaman

25 Aug 2016

What is a Language Model?

- Language Model assigns probability to a sequence of words P(w1, w2.., wn) formed out of the vocabulary V of a language

- Language Models can also generate text using a probabilistic approach (Shannon's work)

Unigrams

- From chain rule of probability, we know:

  $P(w1, w2, \ldots, wn) = P(w1)\, P(w2|w1)\, P(w3|w1, w2) \ldots P(wn|w1, w2 \ldots wn\text{-}1)$

  If we treat the words w as independent RVs, we can write:

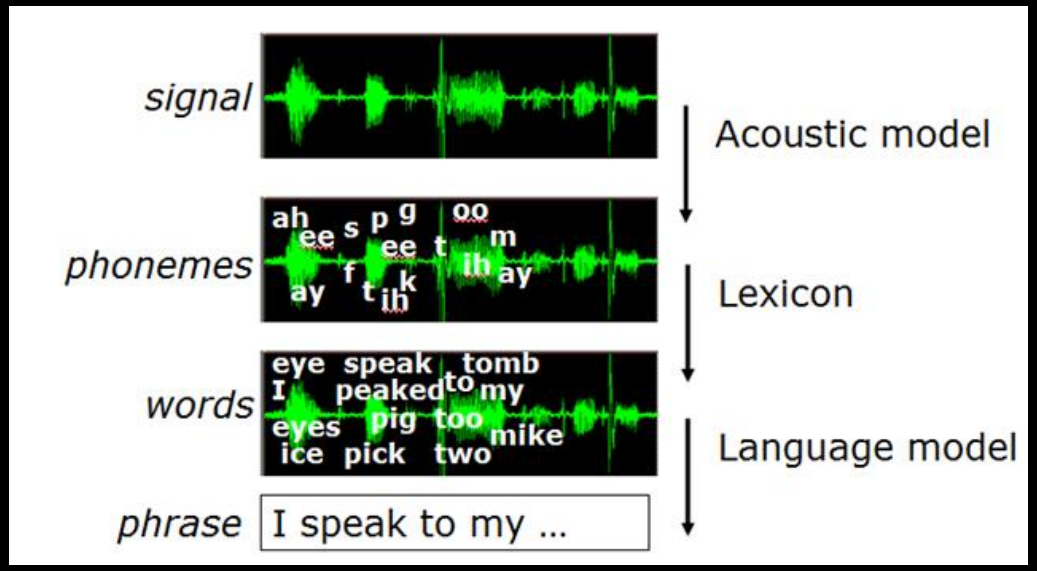
  $P(w1, w2, w3 \ldots wn) = P(w1)\, P(w2) \ldots P(wn)$

- The above is the Unigram model, where we treat each word to be independent of other words in the sequence. This leads to the notion of bag of words

- Unigram models are widely used for Information Retrieval tasks as they are adequate in many situations to identify topics in a given corpus of documents

- However, unigram models are too simplistic for many NLP tasks as the different words in a words sequence are not often independent

Example 1: Speech Recognition



How Dragon works

The goal of Dragon is to make it easier to interact with a computer without using a keyboard or mouse. To succeed at this goal, the program must be able to analyze an incoming stream of sounds and interpret those sounds as commands and dictation. This process of interpretation is called speech recognition, and its success is measured by the percentage of correct interpretations, or *recognition accuracy*.

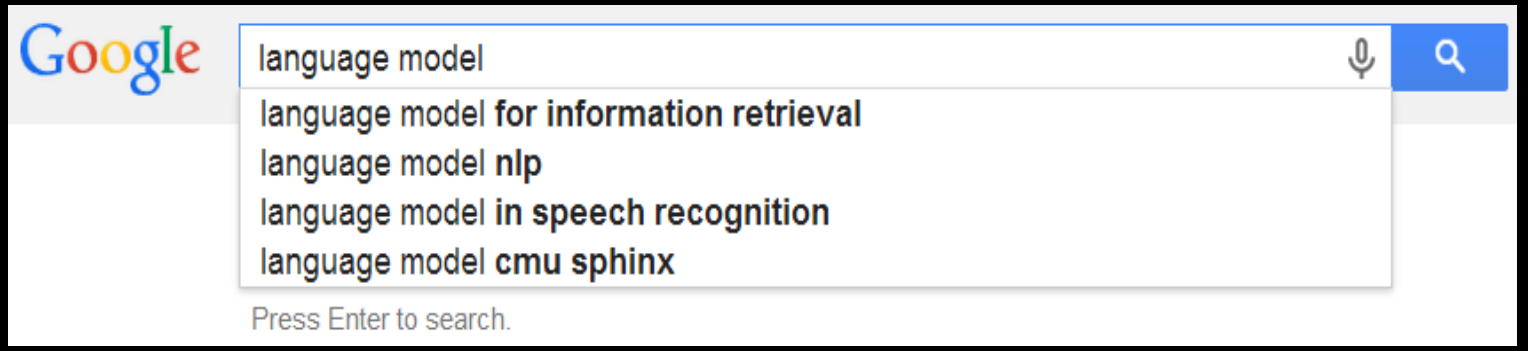
To achieve high recognition accuracy, Dragon relies on several sources of information:

- **Acoustic model**—a mathematical model of the sound patterns used by the speaker's language.
- **Vocabulary**—a list of words that the program can recognize. Each word in the Vocabulary has a text representation and a pronunciation.
- **Language model**- -statistical information associated with a Vocabulary that describes the likelihood of words and sequences of words occurring in the user's speech.

Example 2: Information Retrieval

- Intuition behind LM for IR:
  - A document is a good match to a query if the document model is likely to generate the query.

  $P(d|q) = P(q|d)\ P(d) / P(q)$



- At a high level, this technique may be implemented as:
  - Determine the LM for the document d
  - Determine the probability of a query given the document
  - Rank the documents for the given query as per these probabilities

The Language Model Problem

- The LM problem is to assign a probability to a word sequence
  - $P(w_1, w_2,\ldots, w_n)$
  - Example: P(tomorrow, is, a, holiday)

- This may also be stated as a problem of computing the probability of each word given its preceding context.
  - $P(w_i \mid w_1, w_2 \ldots w_{i-1})$
  - Example: P(holiday | tomorrow, is, a)

N-gram Models

- An N-gram model uses only the previous (n-1) words in order to compute the probability of n^{th} word.
  - Unigram: P(holiday)
  - Bigram: P(holiday|a)
  - Trigram: P(holiday|is, a)

- While the probability of a given word in a word sequence may depend on the probability of joint occurence of all the preceding words, often we assume that the local context is adequate to predict the probability of subsequent words.

- The Markov model is a theoretical framework for expressing this assumption

Computing LM Probabilities

- Consider our example sentence: W = “Tomorrow is a holiday”

- Computing the probability of this sentence amounts to computing the joint probability of the word sequence that constitutes the sentence
  - That is, compute: P(w1 = Tomorrow, w2 = is, w3 = a, w4 = holiday)

- From chain rule of probability we know:
  - P(A, B, C, D) = P(A) P(B|A) P(C|A, B) P(D | A, B, C)
  - General form of chain rule:
    - P(w1, w2, …wn) = P(w1) P(w2|w1) P(w3|w1, w2)…P(wn|w1, w2,..wn-1)

- In our example we need to compute: P(“Tomorrow is a holiday”)
  - P(W) = P(Tomorrow) P(is|Tomorrow) P(a|Tomorrow is) P(holiday|Tomorrow is a)

Estimating probabilities of LM: Naïve method

- Suppose we have N training sentences
- For any sentence S expressed as a word sequence w1, w2…wn:
  - let c(w1…wn) be the count of times that sentence is seen in the training corpus
  - P(S) = c(w1..wn) / N
- For example:
  - P(“Tomorrow is a holiday”) = c(tomorrow is a holiday) / N
  - P(holiday|Tomorrow is a) = count(Tomorrow is a holiday) / count(Tomorrow is a)
  - We may not get enough data with the exact word sequence to estimate these probabilities

Markov Processes

- Consider a sequence of random variables X1, X2, …Xn.
- Each random variable can take any value in a finite set V, the vocabulary
- For now we assume the length n is fixed (e.g., n = 100).
- Our goal is to model: P(X1 = x1, X2 = x2, ... Xn = xn)

  P(X1) P(X2|X1) P(X3|X1, X2)…P(Xn|X1, X2,..Xn-1)

- First order Markov model:

  P(X1 = x1, X2 = x2, ... Xn = xn) = $P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1})$

The first-order Markov assumption: For any $i \in \{2 \ldots n\}$, for any $x_1 \ldots x_i$,

$$P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

Example

- In our example we need to compute: P("Tomorrow is a holiday")
  - P(W) = P(Tomorrow) P(is|Tomorrow) P(a|Tomorrow is) P(holiday|Tomorrow is a)
- Using First Order Markov Assumption, we can write the above as:

  P(W) = P(Tomorrow) P(is|Tomorrow) P(a|is) P(holiday|a)
- Bigram LMs can be represented by the first order Markov model
- Bigram probabilities can be computed from:

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

Exercise:

Bigram probabilities can be computed as a table where the rows and columns are the words in the corpus

Compute Bigram probabilities for:

<s>The scale of innovation & level of precision among the people of Japan is admirable.</s>

<s>Both our nations can learn a lot from each other.</s>

<s>Have very warm memories of visiting Japan as a CM.</s>

<s>The hospitality & the immense scope for cooperation left a deep impression on my mind.</s>

Example

- In our example we need to compute: P("Tomorrow is a holiday")
  - P(W) = P(Tomorrow) P(is|Tomorrow) P(a|Tomorrow is) P(holiday|Tomorrow is a)
- Using First Order Markov Assumption, we can write the above as:

  P(W) = P(Tomorrow) P(is|Tomorrow) P(a|is) P(holiday|a)
- Bigram LMs can be represented by the first order Markov model
- Bigram probabilities can be computed from:

Exercise:

Bigram probabilities can be computed as a table where the rows and columns are the words in the corpus

Compute Bigram probabilities for:

<s>The scale of innovation & level of precision among the people of Japan is admirable.</s>

<s>Both our nations can learn a lot from each other.</s>

<s>Have very warm memories of visiting Japan as a CM.</s>

<s>The hospitality & the immense scope for cooperation left a deep impression on my mind.</s>

Second order Markov Model

In our example: S = “Tomorrow is a holiday”

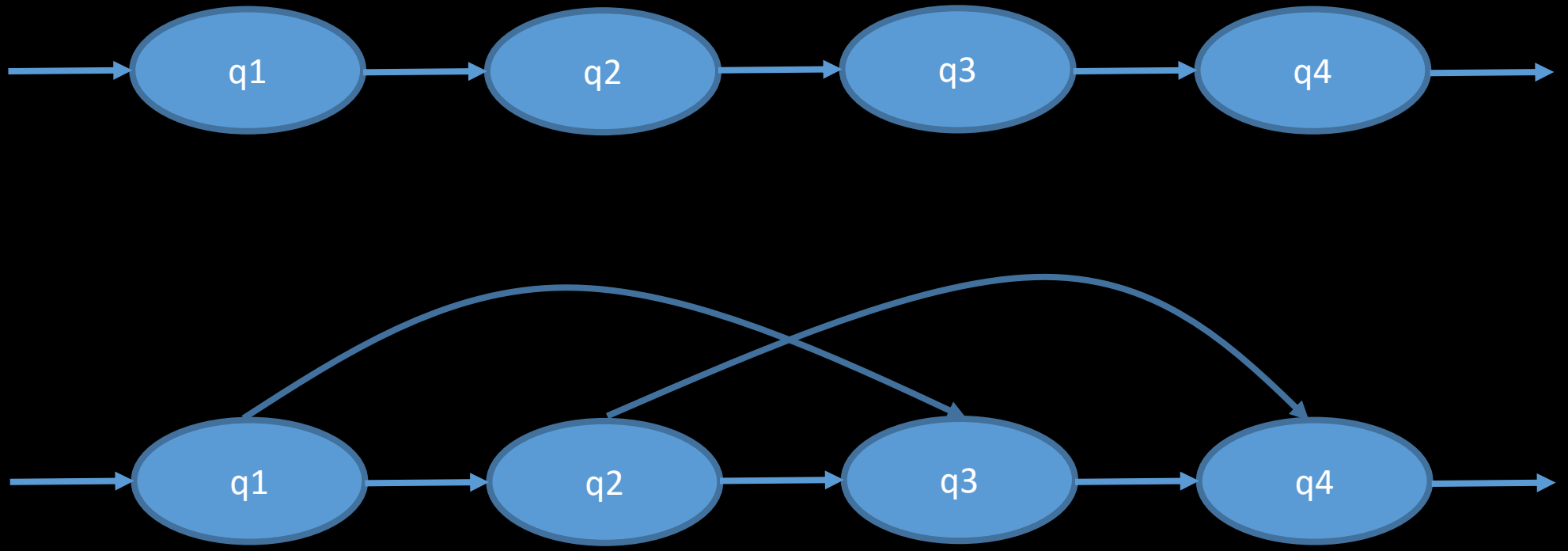
X1 = Tomorrow, X2 = is X3 = a, X4 = holiday

P(“Tomorrow is a holiday”) =

P(Tomorrow) x P(is|tomorrow) x P(a|Tomorrow is) x P(holiday|is a)

Trigrams can be represented by the second order Markov Models

Graph Representation of Markov Models



Extending to N-grams

- We can extend the discussion in the previous slides on bigram and trigram to 4-gram, 5-gram and so on and represent them as Markov processes
- We note that a n-gram LM can be represented by $(n-1)^{th}$ order Markov Model
- However, these are approximations and not exact computations of probabilities. The chain rule specifies the exact computation but for practicality it is necessary to make simplifying assumptions.
- While natural language sentences have long range dependencies (global context), n-gram models, trigrams in particular, are often adequate for many applications
  - Example: Now many talented players are identified due to IPL, let us encourage them.

How to evaluate Language Models?

- Every NLP tool/technique needs to be evaluated in order to:
  - Determine if the given tool meets the performance requirements of the application where it is proposed to be used
  - Benchmark one technique/implementation with the other

- What is a good LM?
  - The one that assigns high probability to frequently occurring sentences and low probability to sentences that occur rarely

Training and Testing

- We train the model on the given training data. During this process we learn the model parameters
- We evaluate the model's performance on the data that is unseen
- We need an evaluation metric to find out how good our LM is
- Extrinsic versus Intrinsic evaluation
  - Extrinsic evaluation is time consuming and might be expensive
  - Intrinsic evaluation is about the LMs themselves and doesn't depend on external application

Perplexity

- Perplexity is a LM evaluation metric and is used for intrinsic evaluation
- Unless the test data is similar to training data, perplexity is not a good measure of the LM
- One can use perplexity in conjunction with extrinsic evaluation

Perplexity

- The best LM is the one that best predicts the unseen test set
- Given a test data, we predict the probability of the test corpus.
- For a uniform distribution, where the probability of a word is given by 1 / N, where N = |V| + 1, the perplexity is N
- Results from Goodman indicates for |V| = 50000, the trigram model gives a perplexity of 74, while bigram and unigram yield 137 and 955 respectively

- We have some test data, m sentences

$$s_1, s_2, s_3, \ldots, s_m$$

- We could look at the probability under our model $\prod_{i=1}^{m} p(s_i)$. Or more conveniently, the *log probability*

$$\log \prod_{i=1}^{m} p(s_i) = \sum_{i=1}^{m} \log p(s_i)$$

- In fact the usual evaluation measure is *perplexity*

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

and M is the total number of words in the test data.

Exercises

- Suppose we have a vocabulary of size 20000 and we would like to build a trigram model. What is the maximum number of parameters that we may deal with?
  - From the above, what can you say about the size of the training data required?
- Can you suggest a good way to represent a tri gram model in Python?
- Given the MaxEnt model, can you come up with a design to generate LM using this classifier?
- If you are required to compute trigram LM probabilities using a combination of cache and the training corpus, how would you go about?

Exercises

- What is the impact of domains (e.g: News articles, technical articles, tweets, Facebook posts etc) on the perplexity of a LM?
- Give examples where n-gram with $n < 4$ might not work well
- If an LM gives excellent performance with respect to perplexity measure, would it give corresponding performance on a real world application, say, speech recognition?
- Does a Unigram model perform better than a model that assumes uniform distribution of word probabilities?

The Bias Variance Trade Off

- Supervised learning algorithms attempt to generalize to unseen input given the model parameters
- The model may be too simplistic (e.g, Unigram) or may be complex (e.g, a 4-gram)
- There are two sources of error that arise due to this:
  - Errors due to underfitting (high bias)
  - Errors due to overfitting (high variance)
- Complex models need more data to train, have many more parameters and are sensitive to noise as they try to fit the outputs to the training data. Their performance on training data may be good but they may perform poorly on test data. Overfitting impacts generalization.

Summary so far

- Given a sequence of words drawn from a vocabulary, Language Model assigns a probability for this word sequence
  - This probability distribution satisfies the 2 fundamental axioms of probability: Probability of all possible word sequences sums to 1 and Probability of any sequence is bounded between 0 to 1, both limits inclusive
- Computing this probability in its exact form as a joint distribution is intractable for non trivial vocabulary and sequence length sizes
- Unigram models assume the words in the word sequence are independent and hence the joint probability reduces to multiplication of individual word probabilities
- Though unigram models are useful for certain class of applications, they have severe limitations, as in reality there is often a dependency between words of a sequence.
- Bigrams approximate the computation of a joint probability distribution by assuming that a word in a sequence depends only on the previous word of the sequence. Trigrams extend this further by allowing 2 previous words to be considered while computing the probability of a word in a sequence.
- Bigrams can be mathematically represented by the first order Markov process while the second order Markov process can be used to present trigrams. In general a k-gram LM can be represented by (k-1)st order Markov.
- Note that it is good to think of domain concepts (such as bigram, trigram) separate from the representation (such as Markov processes) of such concepts. A given representation may be applicable to multiple class of problems from different domains and a given concept may be represented using multiple representations. Markov Model is a formal framework applicable to multiple problems – NLP LM is one of them.

Summary - continued

- The model complexity and the number of model parameters increase as we increase the order of our representation. The simplest representation of Unigram involves a direct multiplication of probabilities of words of a sequence and computing trigram probabilities would involve N^3 computations as each word in a trigram is a random variable that can take $N = |V|$ values.
- Perplexity is a metric that is used to evaluate the performance of a LM
- Based on the nature of application, simple models might result in errors due to underfitting while increasing the model complexity requires a larger corpus for training and may also cause errors due to overfitting
- Hence, Bias-Variance trade offs need to be considered while selecting the LM
- From the studies done by Goodman it was shown that the Trigrams provide an optimal trade off for many applications
- Back off, Interpolation techniques can be used to select the right mix of models for the application

Back off and Interpolation

- Back off techniques successively back off starting from a higher complexity language model to lower ones based on the performance of the model
  - Use trigram if there is good evidence that it produces good results. That is, we have adequate data to train the system and the trigram estimates converge close to their true values
  - If trigrams are not showing good results, try using bigrams, else use unigrams
- Interpolation produces a new language model by mixing all the different n-gram models using weights as coefficients
- In general interpolation is known to produce better results compared to back off

Interpolation

- Main challenge: Sparse data model
- Natural estimate is MLE
  - $q(w_i \mid w_{i-2}, w_{i-1})$ = trigram_counts / bigram_counts
- There are N^3 parameters in the trigram model – eg: If N = |V| = 10000, then there are 10^{12} parameters in the model
- In many cases trigram_counts may be zero or even bigram_counts may be zero leading to sparse data problem

Bias Variance

- Unigram, Bigram and Trigram models have different strengths and weaknesses
- Trigram models exhibit relatively low bias as it takes in to account the local context but need more data to train
- Unigram models ignore the context and hence suffer from high bias
- We need an estimator that can trade off these strengths and weaknesses

Linear Interpolator

- We define the probability of a word w_i given w_{i-1} and w_{i-2} to be:

  $q(w_i|w_{i-2}, w_{i-1}) = \lambda_1 q_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 q_{ML}(w_i|w_{i-1}) + \lambda_3 q_{ML}(w_i)$

  With the constraints:

  1. $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  2. $\lambda_i >= 0$ for all i

- The constraints ensure that the interpolator produces a valid probability distribution

  Let V be the vocabulary and V' = V U {STOP}

  $\Sigma_{w \varepsilon V'} q(w \mid u, v) = 1$ and $q(w \mid u, v) >= 0$ for all $w \varepsilon V'$

Exercises

- Show that the trigram linear interpolated model provides a valid probability distribution

- Write the equation for a 4 gram interpolated model with constraints on the coefficients

Determining the λ coefficients

- Create a validation dataset which is the held out dataset separate from the training data
- Compute q_{ML} parameters for unigram, bigram and trigram from the training data
- Define $c'(w_1, w_2, w_3)$ as the count of trigram (w_1, w_2, w_3) in the validation set
- Choose $\lambda_1, \lambda_2, \lambda_3$ such that these coefficients maximize:

  $L(\lambda_1, \lambda_2, \lambda_3) = \Sigma_{w1,w2,w3}\ c'(w_1, w_2, w_3) \log q(w3 \mid w1, w2)$ such that:

  $q(w_i|w_{i-2}, w_{i-1}) = \lambda_1\ q_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2\ q_{ML}(w_i|w_{i-1}) + \lambda_3\ q_{ML}(w_i)$

  With the constraints:

  1. $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  2. $\lambda_i >= 0$ for all i

Varying the values of λ coefficients

- Often it is useful to keep different sets of λ coefficients in order to get better performance.
- For this, we define a partition function $\pi(w_{i-1}, w_{i-2})$
  - Example:
    - $\pi(w_{i-1}, w_{i-2}) = 1$ if $\text{Count}(w_{i-1}, w_{i-2}) = 0$
    - $\pi(w_{i-1}, w_{i-2}) = 2$ if $1 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 2$
    - $\pi(w_{i-1}, w_{i-2}) = 3$ if $2 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 5$
    - $\pi(w_{i-1}, w_{i-2}) = 4$ $otherwise$
- We can write:

$$q(w_i|w_{i-1}, w_{i-2}) = \lambda_1^{\pi(w_{i-1},w_{i-2})} * q_{ML}(\text{w}_\text{i}|\text{w}_\text{i-2}, \text{w}_\text{i-1}) + \lambda_2^{\pi(w_{i-1},w_{i-2})} * q_{ML}(\text{w}_\text{i}|\text{w}_\text{i-1}) + \lambda_3^{\pi(w_{i-1},w_{i-2})} * q_{ML}(\text{w}_\text{i})$$

Subject to the constraints:

1. $\lambda_1^{\pi(w_{i-1},w_{i-2})} + \lambda_2^{\pi(w_{i-1},w_{i-2})} + \lambda_3^{\pi(w_{i-1},w_{i-2})} = 1$
2. $\lambda_i^{\pi(w_{i-1},w_{i-2})} \geq 0 \ \forall i$

Add 1 Smoothing

- Add 1 smoothing doesn't always work well when the n-gram matrix is sparse
- It may be adequate when dealing with classification problems (as we saw in Naïve Bayes classifier) where the sparsty is less but does a poor job for language model tasks
- However, it is still useful as a baseline technique

	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
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Table: Courtesy Dan Jurafsky's Coursera Slides

Discounting Methods

- Motivation
  - Language Models are sensitive to the training corpus.
  - N-gram models are sparsely populated as n increases.
  - Test data might have several word sequences that might not have been covered in the training data.
  - The probability may be zero or sometime indeterminate even if the sequence may be a very valid and frequently found sequence
  - Add 1 smoothing doesn't always work well
- Discounting methods attempt to address this problem
- The strategy is to extract some probability mass out of the probability distribution of training data and assign them to new sequences
- For each n-gram, we subtract a fixed number from the counts, hoping to distribute them to new sequences in the test data in a fair manner
- Define: $count^{*}(x) = count(x) - d$, where d is a number (say 0.5)

w1	w2	count
the	researchers	515
the	university	421
the	journal	315
the	first	312
the	study	281
the	same	253
the	world	246
the	new	205
the	most	185
the	scientists	183
the	australian	164
the	us	153
the	research	140
the	brain	131
the	sun	122
the	team	118
the	other	113
the	findings	97
the	past	97
the	next	94
the	earth	85

the	toddlers	1
the	larynx	1
the	scripts	1
the	investor	1
the	vagus	1
the	fradulent	1
the	lomonosov	1
the	fledgling	1
the	cotsen	1
the	elgin	1
the	single-wing	1
the	stout	1
the	britons	1
the	bonds	1
the	armand	1
the	construct	1
the	fin	1
the	lebanese	1
the	frog-sniffing	1
the	did	1
the	nano-nylon	1
the	dread	1
the	lizard-like	1
the	auditory	1
the	manuscipts	1
the	closet	1

Discounting Methods

- Calculate the new probabilities using Count* for each word w

- As we have subtracted some counts from the initial value, we will be left with a probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \text{count}^*(w_{i-1}, w) / count(w_{i-1})$$

- Our goal is to assign this probability mass to the missing n-grams in the training data in a principled way

Allocating the probability mass

- Let the vocabulary be V and let A be the set of words that are in the n-grams we encountered in the training data. Let B be: V – A

- For each word in A compute the probability as:

$$count^*(w_{i-1}, w)/count(w_{i-1})$$

- If w is not in A, then scale α with the bigram (in case of trigram LM) q_{BO} probability of w (Refer M Collins)

Code Walkthrough