Natural Language Processing

Some screenshots are taken from NLP course by Jufrasky
— Used only for educational purpose

N-Gram Modeling

Language Modeling

N-Gram Modeling

- Assign a probability to a sentence
  - P(“large score”) < P(“high score”)
  - P(“scored a point”) < P(“scored a goal”)
  - P(“cute summer”) < P(“hot summer”)=
- Applications
  - Spell checker
  - Machine translation
  - Summarisation
  - Q&A Systems
  - Speech Recognition

N-Gram Modeling

- Goal: Compute the probability of sentence or sequence of words
- $P(w) = P(w_1, w_2, \ldots, w_n)$
- For example: $P(w_5 \mid w_1, w_2, w_3, w_4)$ = ??? is called language modeling
- Above modeling can be called as a grammar!!
- Chain rule of probability is used

LNMIIT

Chain Rule of Probability

- $P(A|B) = P(A,B)/P(B)$
- $P(A,B) = P(A/B).P(B)$
- If we generalise the rule:
  - $P(X_1,X_2,\ldots,X_n) = P(X_1).P(X_2|X_1).P(X_3|X_1,X_2)\ldots.P(X_n|X_1,X_2,\ldots X_{(n-1)})$

Joint Probability of Words in Sentence

- P("It's a rainy day in Jaipur") is given by
  - P(It's).P(a|It's).P(rainy|It's a).P(day|It's a rainy)……..
- How to estimate the probabilities?
  - P(day|It's a rainy) = count("It's a rainy day") / count("It's a rainy")
- But the above counts are __________?

LNMIIT

Use Markov Assumption

- P(day|It's a rainy) = P(day|rainy)
- Or, P(day|It's a rainy) = P(day| a rainy)
- Markov assumption:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

That is,

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Markov Assumption

- Unigram is the simplest model but very naive

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish
```

Markov Assumption

- Bigram is relatively a good model

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

Markov Assumption

- Can extend this to trigram, 4-grams and 5-grams also
- n-gram in general is an insufficient model of language to a general model
- But often we can work around with this
- There will be many sentences that are having long distance dependency — for example: “The system which we had bight a month back for our computer lab crashed”

LNMIIT

How to estimate bigram probabilities

- Maximum likelihood estimate:
  - $P(w_i \mid w_{i-1}) = \frac{Count(w_{i-1}, w_i)}{Count(w_{i-1})}$
- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like rain </s>

- P(I|<s>) = 2/3
- P(</s>|Sam) = 1/2
- P(Sam|<s>) = 1/3
- P(Sam|am) = ?
- P(do|I) = ?

Berkeley Restaurant Project Sentences

9222 sentences are there:

- 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

Berkeley Restaurant Project Sentences

	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

Berkeley Restaurant Project Sentences

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

Berkeley Restaurant Project Sentences

- Calculate P("I want Chinese food") = P(I|<s>).P(want/I).P(Chinese|want).P(food|Chinese)

  = ???

Berkeley Restaurant Project Sentences

- P(english|want) = 0.0011
- P(chinese|want) = 0.0065

→ "want Chinese" is more common

- P(to|want) = 0.66 → grammatical reason
- P(eat|to) = 0.28
- P(food|to) = 0 → contingent 0. No training example
- P(want|spend) = 0 → grammatical reason
- P(i|<s>) = 0/.25

Practical Issue

$$p_1 \cdot p_2 \cdot \cdots \cdot p_n = \log p_1 + \log p_2 + \cdots + \log p_n$$

Why we take log?

When we take products we may get an underflow

Addition is faster than multiplication

Language Model Toolkits

- http://www.speech.sri.com/projects/srilm
- Google N-Gram (2006): http://ngram.googlelabs.com
  - Trillion words
  - 14 million unique words (> 200 times)
  - 1 billion 5-word sequences (> 40 times)

Evaluation and Perplexity

- How good is our model?
- Should prefer “good” sentences than “bad” sentences
- We train the parameters on a training set
- And test the model with a test data that is not seen previously
- Evaluation metric is taken and checked to see how the model behaved

Evaluation and Perplexity

- Best evaluation is:
  - Take the model A and B and test it in-vivo:
    - Spell correction system
    - Machine translation system
- Above method is good but
  - Time consuming and Tedious
  - Extrinsic evaluation

Evaluation and Perplexity

- We do intrinsic evaluation
- This is called as perplexity — instead of in-vivo we do in-vitro

Perplexity

- The Shannon Game:
  - How well can we predict the next word?

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
....
fried rice 0.0001
....
and 1e-100

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

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

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

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

LNMIIT

Perplexity

- Perplexity is also called as average branching factor

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

LNMIIT

Perplexity

- Recognising digits — 0,1,2,3,4,5,6,7,8,9
- For each digit probability = 1/10
- Hence $P(W) = ((1/10)^{10})^{-1/10}$
- Hence at each point there are ten possible branches for recognising of digits

Maximum Likelihood Estimate

Issues:

- Sparse Data
- Corpus for training is limited
- Phrases or sequence of words not in the training set will have 0 probability value
- Some common phrases might have very low probability value in the N-gram matrix

Maximum Likelihood Estimate

- In the Shakespeare corpus that we had seen previously we had seen 300,000 bigrams
- But according to the the vocabulary there are possible V^2= 844 million bigrams
- This means 99.96% of the possible bigrams were never seen and so they all have zero probability in the bigram matrix

Berkeley Restaurant Project Sentences

- P(english|want) = 0.0011
- P(chinese|want) = 0.0065

“want Chinese” is more common

- P(to|want) = 0.66

grammatical reason

- P(eat|to) = 0.28
- P(to|food) = 0

contingent 0. No training example

- P(want|spend) = 0

grammatical reason

- P(i|<s>) = 0/.25

Maximum Likelihood Estimate

- Some zeroes have to be zeroes (phrases that are not possible)
- Some are common but that are rare in the training corpus
- A small number of phrases happen very frequently — Zipf's Law
  - This, the system can learn quickly
- A larger number of phrases happen very rarely — Zipf's Law
  - This, the system will take more time to learn
- Solution is to get some values on board at least for the zero values of probability

LNMIIT

Smoothing Methods

- Need to modify the MLE method and make probability values non-zero
- Increase the low probability value of phrases in the N-Gram matrix
- This method is generally called as Smoothing
- Smoothing addresses the poor estimates that comes because of variability

Laplace Smoothing

- Laplace smoothing is the simplest method
- Take the N-gram count matrix and increase all values by 1
- This is called as Laplace smoothing or Laplace law

Laplace Smoothing

- Unigram MLE: $P(w_i) = \frac{c_i}{N}$
- Unigram Laplace Estimate: $P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$ → Method 1

Laplace Smoothing

- Method 1: needs to change both numerator and denominator
- Another method:
  - $c_i^* = (c_i + 1)\frac{N}{N+V}$
  - Normalise C_i^* with respect to N then we shall get $P_{Laplace}(w_i)$

Method 2

- **Main idea here is**: re-estimation via C_i^* lowers the larger values of C_i and increases the lower values or zero values of C_i

Bigram Matrix Count after it is Laplace Smoothed

	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

LNMIIT

Bigram Matrix Probabilities after it is Laplace Smoothed

	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

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

Laplace Smoothing

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

	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

Laplace Smoothing

Before

	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

After

	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

Laplace Smoothing

- C(want to) changed from 608 to 238! Its potability changed from 0.66 to 0.26
- C(Chinese food) changed from 0.52 to 0.052
  - A reduction of 10 times!!
- Laplace is very naive but its very simple. So it is still in use in some restricted domains:
  - Some pilot studies
  - Where there are not many zero entries in the Ngram matrix

δ-Laplace Smoothing

- Instead of adding 1 we can add δ to reduce the discounting
- But fixing δ will be necessary in this case and it needs to be dynamically changing too
- But it will suffer from poor variance

Good-Turing Discounting

- Good (1953)
- Good credited Turing for the original idea
- Main idea is to use the count of things we have seen once to help with things we have never seen

Good-Turing Discounting

- A word or N-Gram that occurs once is called a **singleton** or **hapax legomenon**
- Good-Turning method uses singletons as a re-estimate of the frequency of zero-count bigrams

Good-Turing Discounting

- Formulation:
  - Let N_c be the number of N-grams that occur c times
  - N_c is called frequency of frequency!
  - N_0 is the number of N-grams with zero probability
  - N_1 is the number of N-grams with count 1
  - N_2 is the number of N-grams with count 2 and so on
  - In general N_c is a bin where it stores the N-grams that occur c times

Good-Turing Discounting

- MLE count for N_c is c
- Good-Turing estimates things that occur c times in training corus by MLE count of N-grams that has counts of c+1
- Good-Turing replaces the count c by a smoothed count c*
- $c* = (c+1)\frac{N_{c+1}}{N_c}$ for $N_1, N_2, \ldots$
- And P^*_{GT} (things in N_0) = $\frac{N_1}{N}$

Good-Turing Discounting

- Good-Turing method proposed for estimating populations of animal species in 1953
- Example:
  - There are 8 species of fish in the tank
  - Those 8 species are carp, perch, whitefish, trout, salmon, eel, catfish and bass
  - But we have only seen 6 species as of now
  - Can you estimate the probability of finding catfish?

Example

- $N_0 = \{catfish, bass\}$
- $N_1 = \{trout, salmon, eel\}$
- $N_2 = \{whitefish\}$
- $N_3 = \{perch\}$
- $N_4 = N_4 = N_6 = N_7 = N_8 = N_9 = \{\}$
- $N_{10} = \{carp\}$

Calculate $P^*_{GT}(unseen)$

Calculate $C^*(trout)$

What is MLE for trout?

What is $P^*_{GT}(trout)$

What is $P^*_{GT}(catfish)$?

LNMIIT

After applying GT smoothing for corpus AP and BeRP

AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

LNMIIT

Issues in Good-Turing Estimation

- Assumes binomial distribution (Church et al 1991)
- Assumes N_0 is known
- Estimate $c*$ for N_c depends on N_{c+1}. What if $N_{c+1} = 0$?
- In previous example $N_4 = 0$
  - So $c*(perch)$ will be 0!

Simple Good-Turing Method

- Computer bins N_c
- Before calculating $c* = (c+1)\dfrac{N_{c+1}}{N_c}$, smoothen all values of N_c
- Simplest method is using the values N_c and c in logspace get a linear regression and predict the scores of N_c that are of zero values

$$\log(N_c) = a + b\log(c)$$

Katz method in Good-Turing

- No need to change c to $c*$ for $c > k$ where k is a threshold value
- For $1 \leq c \leq k$,

$$c* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)(N_{k+1})}{N_1}}{1 - \frac{(k+1)(N_{k+1})}{N_1}}$$

Often N-grams with very low counts are also considered as zero counts and then applied smoothing methods

Good-Turing is often used with methods like interpolation and backoff algorithms

Interpolation and Backoff

Backoff

- If we are trying to compute $P(w_n \mid w_{n-2}w_{n-1})$ and we do not have the example of the trigram in the training then use the value of the bigram $P(w_n \mid w_{n-1})$
  - If the bigram is also not seen in the example then go for unigram $P(w_n)$
- In some cases we use the trigram, bigram and unigram to get a mix-up value that we will use for finding the probability of a trigram

Interpolation

Interpolation

Simple Interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

Interpolation by conditioning on the context

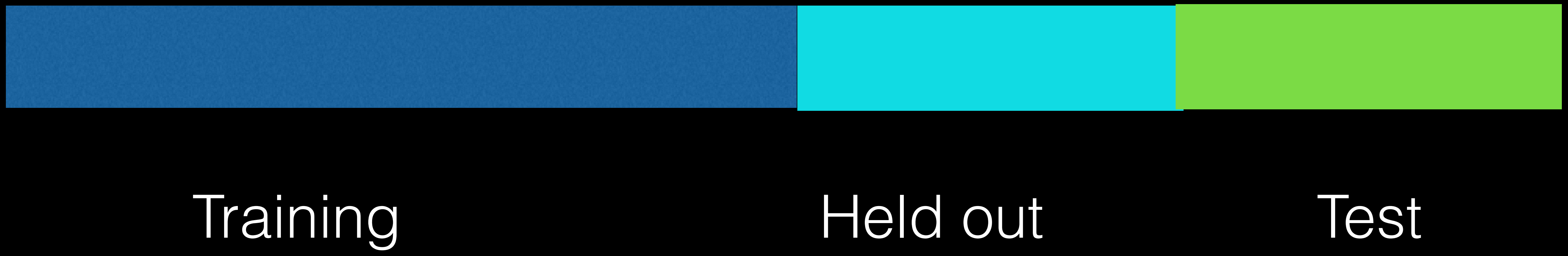
$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n)$$

Setting λ Values

- λ values can be learned using the held-out corpus
- held-out corpus is an additional training corpus not to set the N-gram counts but to set other parameters
- We can use this method to try out different values of λ and then see which maximises the likelihood of the held-out corpus
- One better way of finding this λ is to use the EM algorithm

LNMIIT

Held out Corpus and Re-estimate



After training is used, held out data is used to re-estimate the parameters. For example λ

Choose λs to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for λs that give largest probability to held-ou

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Held out Corpus and Re-estimate

After training is used, held out data is used to re-estimate the parameters. For example λ

$$N_c \qquad MLE = \frac{c}{N}$$

$$Classcount(c) = \Sigma_{w \in N_c} Count_{HO}(w)$$

$$AveCount(c) = \frac{Classcount(c)}{|N_c|} \Rightarrow P_{HO}(w) = \frac{AveCount(c)}{N_{HO}}$$

Backoff

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.} \end{cases}$$

Need for discounts and α values

- We use discussed probability since we need to preserve the probability rule that the sum of probability = 1
    - If we use the raw ones then the sum of probability values might exceed one. This is the reason we use the discounted probability when we calculate the Katz probability
- The parameter α is to make sure that whatever values that are used to increase some of the zero values of probability equates to the value discounted from some values
- Hence discount and α both are necessary here

Good-Turing for AP Newswire

Church and Gale (1991)

From 22 million words AP Newswire Corpus

Discounted score = 0.75 (approx)

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	4.19
7	6.21
8	7.24
9	8.25

Absolute Discounting with Interpolation

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

Kneser-Ney Smoothing

- Estimate the probabilities of the unigrams in a better way
- Consider the Shannon Game: *I can't see without my reading* ______
- Will you fill up with glasses or Fransisco
- But Francisco is more common than glasses as an unigram? Is it not?
- But it looks less common here since we have seen reading! And another perspective is we have not seen “San" before Francisco!
- **Instead of the question how likely is w?, we shall ask now how likely w is to continue here?**

Kneser-Ney Smoothing

- Instead of the question
  - $P(w)$: **how likely is w?**
- We shall ask now
  - $P_{conti}(w)$: **how likely w is to continue here from previous word in a novel way?**
- For each word let us compute how many bigrams it completes

Kneser-Ney Smoothing

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

This will make sure that the unigram word "Fransisco" is less likely!

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left|\{w : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Generalised Recursive Formulation

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}w_i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-n})$$

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuationcount}(\cdot) & \text{for lower orders} \end{cases}$$

The best-performing version of Kneser-Ney smoothing is called modified Kneser-Ney smoothing (Chen and Goodman,1998)

This modified Kneser-Ney uses three different discounts d_1, d_2, and d_{3+} for N-grams with counts of 1, 2 and three or more, respectively

Out of Vocabulary Words

- Called as OOV words
- Create a new token <UNK>
- Smoothing not applied since we need to know the count of OOV words and that we do not know
- At text normalisation phase replace all known words to <UNK> and treat it as another word
- Train its probabilities as a normal word
- Use <UNK> probabilities if there are unknown words encountered, that is not in the training set