POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Text Mining 2
Language Modeling

Madrid Summer School 2014
Advanced Statistics and Data Mining

Florian Leitner
florian.leitner@upm.es

A Motivation for Applying Statistics to NLP

Two sentences:

"Colorless green ideas sleep furiously." (from Noam Chomsky's 1955 thesis)

"Furiously ideas green sleep colorless."

Which one is (grammatically) correct?

An unfinished sentence:

"Adam went went jogging with his …"

What is a correct phrase to complete this sentence?

Incentive and Applications

- Manual **rule-based** language processing would become **cumbersome**.
- Word frequencies (**probabilities**) are **easy to measure**, because large amounts of texts are available to us.
- Modeling language based on probabilities enables many existing **applications**:
  - Machine Translation
  - Voice Recognition
  - Predictive Keyboards
  - **Spelling Correction**
  - **Part-of-Speech (PoS) Tagging**
  - Linguistic Parsing & **Chunking**
    - (analyzes the grammatical structure of sentences)

Zipf's (Power) Law

E = Expected Value →
the "true" mean →

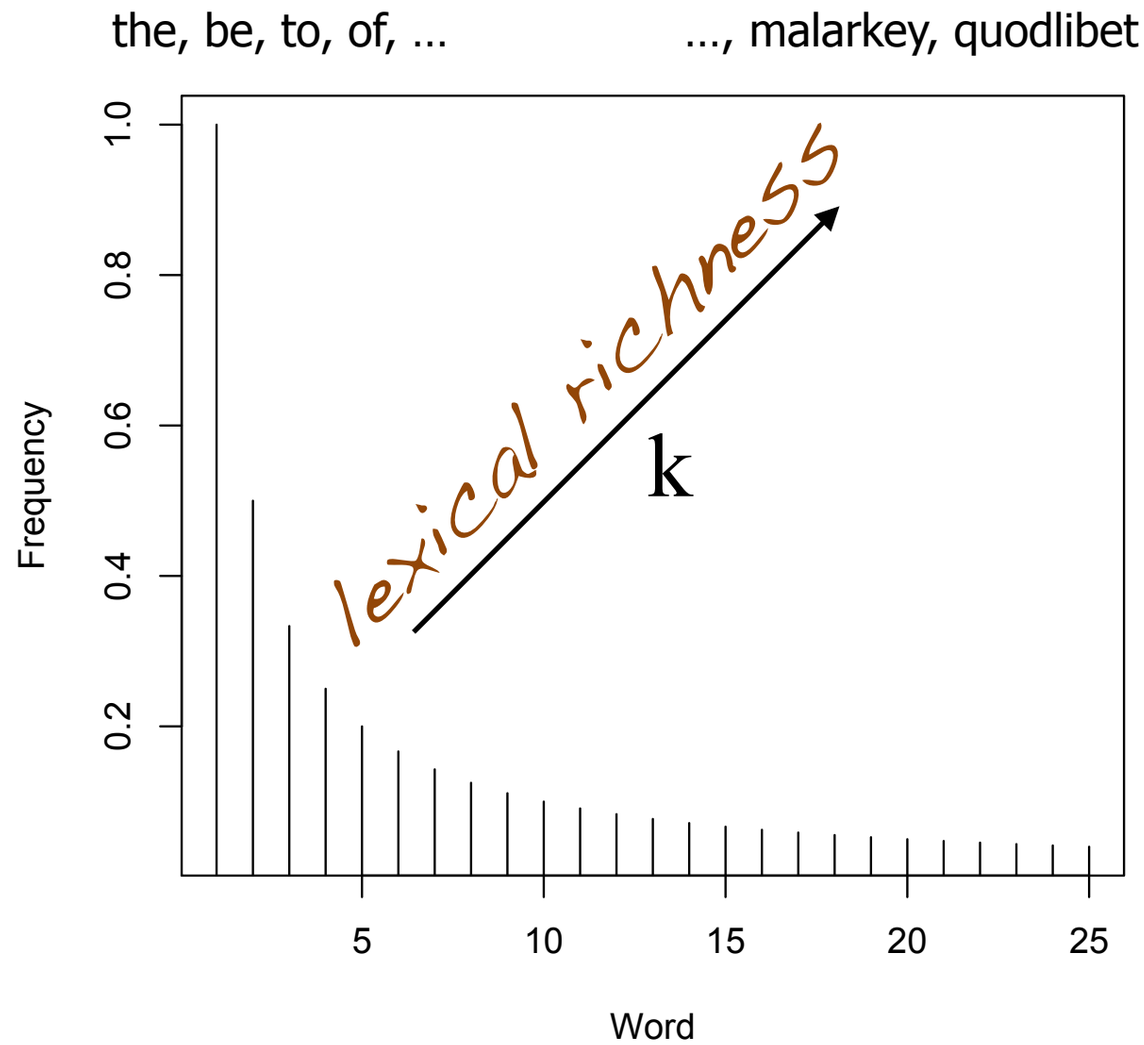
Word frequency inversely proportional to its rank (ordered by counts) ⟶ $f \propto 1 \div r$ *dim. reduction*

$$k = E[f \times r]$$

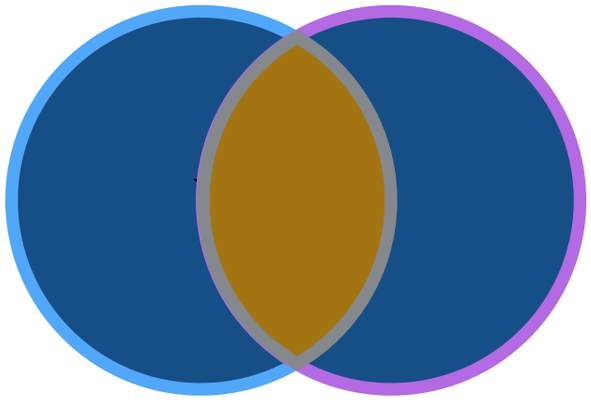
Words of lower rank "clump" within a region of a document

Word frequency is inversely proportional to its length

Almost all words are rare
This is what makes language modeling hard!



The Conditional Probability for Dependent Events



and Joint Probability

$$P(X \cap Y) = P(X, Y) = P(X \times Y)$$

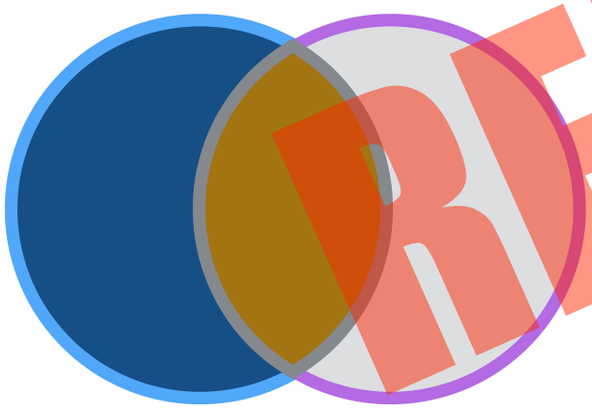
The **multiplication principle** for dependent events*:

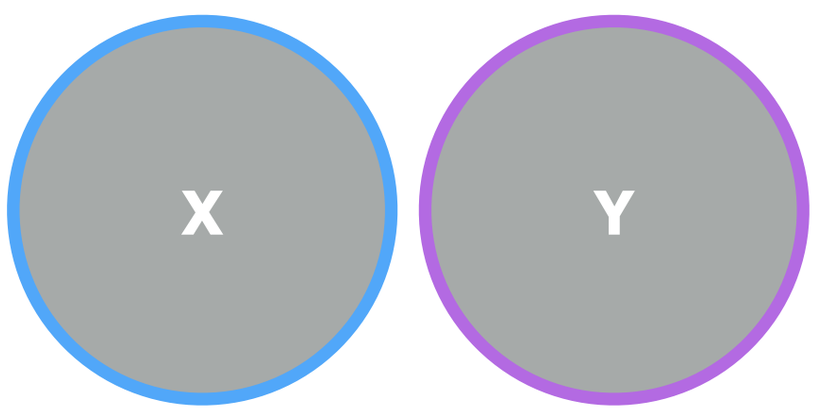
$$P(X \cap Y) = P(Y) \times P(X \mid Y)$$

therefore, by using a little algebra:

Conditional Probability

$$P(X \mid Y) = P(X \cap Y) \div P(Y)$$





*Independence

$$P(X \cap Y) = P(X) \times P(Y)$$

$$P(X \mid Y) = P(X)$$

$$P(Y \mid X) = P(Y)$$

next word vs. whole n-gram

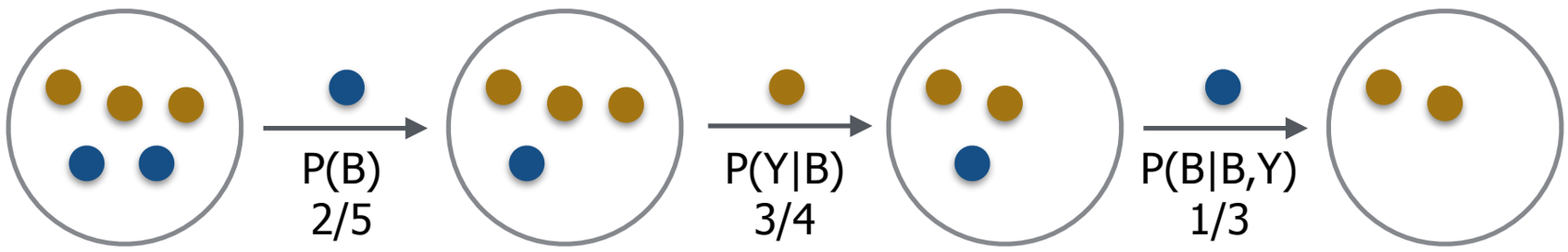
The Chain Rule of Conditional Probability

"The Joint Probability of Dependent Variables"

(don't say I told you that...)

$$P(A,B,C,D) = P(A) \times P(B|A) \times P(C|A,B) \times P(D|A,B,C)$$

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1}) = \prod_{i=1}^{n} P(w_i|w_1^{i-1})$$



$$P(B,Y,B) = P(B) \times P(Y|B) \times P(B|B,Y,B)$$
$$1/10 = 2/5 \times 3/4 \times 1/3$$

NB: the ∑ of all "trigrams" will be 1!

Probabilistic Language Modeling

- Manning & Schütze. Statistical Natural Language Processing. 1999
- A sentence W is defined as a **sequence** of words $w_1, \ldots, w_n$
- Probability of **next word** w_n in a sentence is: $P(w_n | w_1, \ldots, w_{n-1})$
  - a **conditional probability**
- The probability of the **whole sentence** is: $P(W) = P(w_1, \ldots, w_n)$
  - the **chain rule** of conditional probability
- These counts & probabilities form the **language model** [for a given document collection (=**corpus**)].
  - the model variables are **discrete** (counts)
  - only needs to deal with **probability mass** (not density)

Words, Tokens, Shingles and N-Grams

Text with words

This is a sentence.

Character-based, Regular Expressions, Probabilistic, …

"tokenization"

Tokens

This | is | a | sentence | NB: .

Token **N-Grams**

2-Shingles

This is | is a | a sentence | sentence .

a.k.a. k-Shingling

3-Shingles

This is a | is a sentence | a sentence .

Character **N-Grams**

all **trigrams** of "sentence": [sen, ent, nte, ten, enc, nce]

Modeling the Stochastic Process of "Generating Words" Using the Chain Rule

"This is a long sentence having many..." ➡

$$P(W) = P(\text{this}) \times$$

$$P(\text{is} \mid \text{this}) \times$$

$$P(\text{a} \mid \text{this, is}) \times$$

$$P(\text{long} \mid \text{this, is, a}) \times$$

$$P(\text{sentence} \mid \text{this, is, a, long}) \times$$

$$P(\text{having} \mid \text{this, is, a, long, sentence}) \times$$

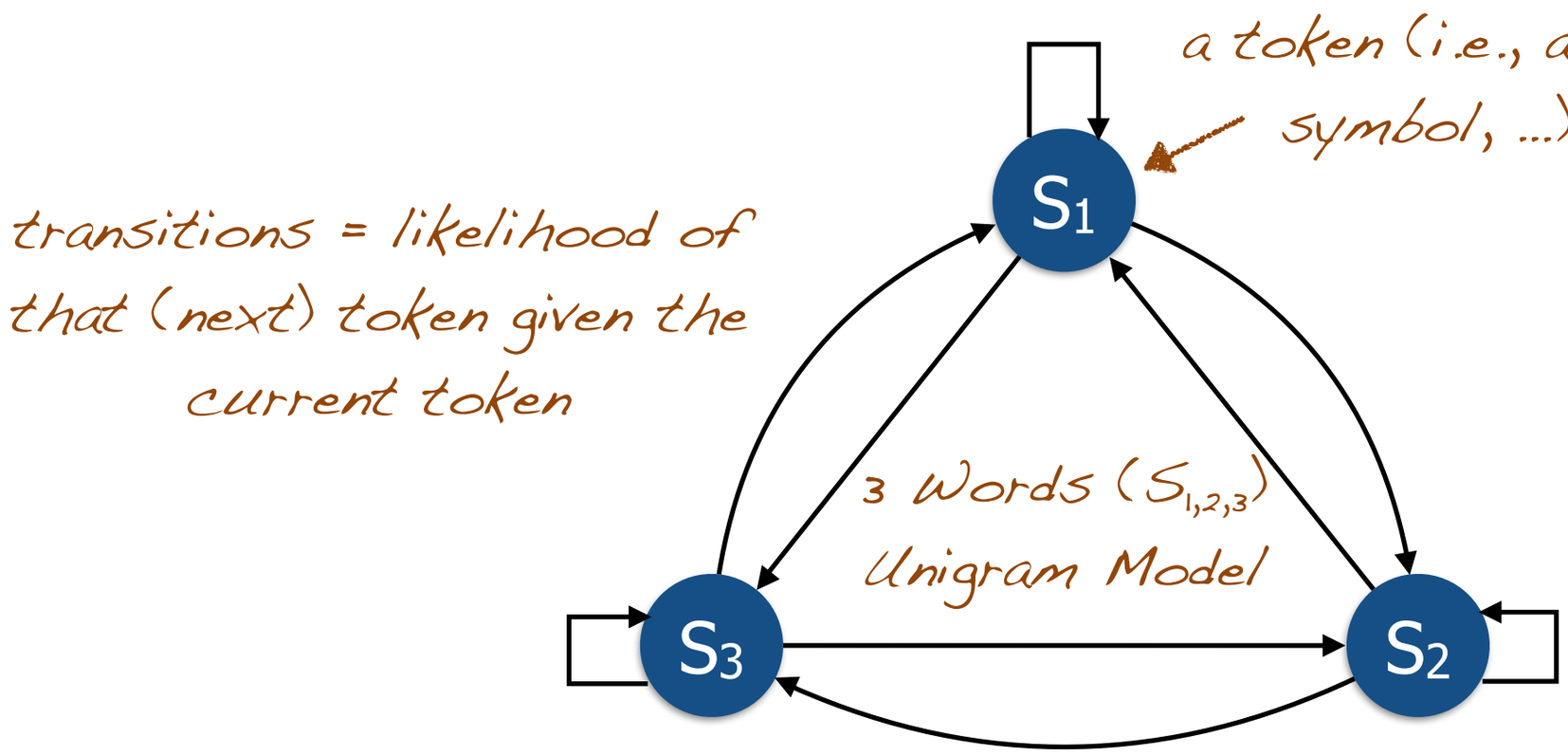
$$P(\text{this, is, a, long, sentence, having}) \times$$

$$\ldots$$

n-grams for n > 5:
Insufficient Data
(and expensive to calculate)

The Markov Property

A Markov process is a stochastic process who's future state only depends on the current state, but not its past.



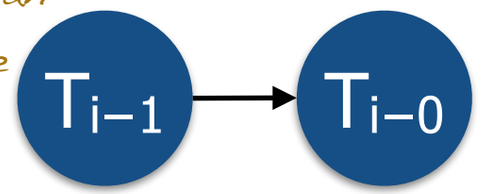
Modeling the Stochastic Process of "Generating Words" Assuming it is Markovian

"Unigram" Model"

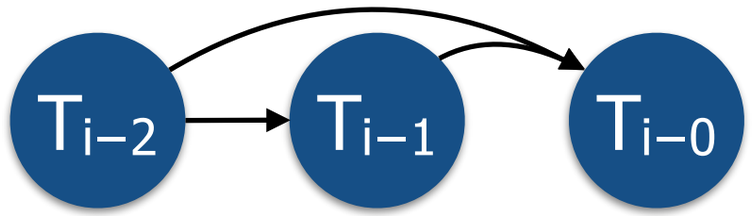
$$\prod P(w_i|w_1^{i-1}) \approx \prod P(w_i|w_{i-k}^{i-1})$$

NB: these are the Dynamic Bayesian Network representations of the Markov Chains (see lesson 5)

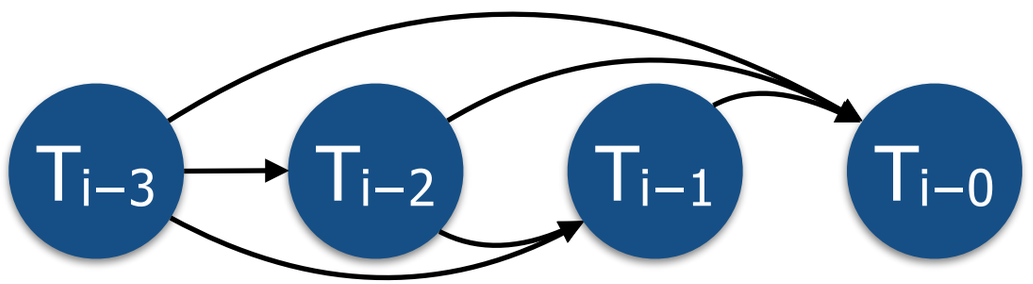
- 1st Order Markov Chains
  - Bigram Model, k=1, P(be | this)



- 2nd Order Markov Chains
  - Trigram Model, k=2, P(long | this, be)



- 3rd Order Markov Chains
  - Quadrigram Model, k=3, P(sentence | this, be, long)



- ...

dependencies could span over a dozen tokens, but these sizes are generally sufficient to work by

Calculating n-Gram Probabilities

- Unigrams:

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

- Bigrams:

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

- **n-Grams** (n=k+1):

$$P(w_i|w_{i-k}^{i-1}) = \frac{count(w_{i-k}^{i})}{count(w_{i-k}^{i-1})}$$

N = total word count

- **Language Model:**

$$P(W) = \prod P(w_i|w_{i-k}^{i-1}) =$$

$$= \prod P(w_i|w_{i-k}, ...w_{i-1})$$

Practical Tip:
transform all probabilities
to logs to avoid underflows
and work with addition

k = n-gram size - 1

Calculating Probabilities for the Initial Tokens

- How to calculate the first/last few tokens in n-gram models?
  - "This is a sentence." ➡ P(this | ???)
  - $P(w_n|w_{n-k}, \ldots, w_{n-1})$
- Fall back to **lower-order Markov models**
  - $P(w_1|w_{n-k},\ldots,w_{n-1}) = P(w_1);\ P(w_2|w_{n-k},\ldots,w_{n-1}) = P(w_2|w_1);\ \ldots$
- Fill positions prior to $n = 1$ with a **generic "start token"**.
  - left and/or right **padding**
  - conventionally, something like "**<s>**" and "**</s>**", "*" or "▪" is used
  - (but anything will do, as long as it does not collide with a possible, real token in your system)

NB: it is important to maintain the sentence terminal token to generate robust probability distributions; do not drop it!

Unseen n-Grams and the Zero Probability Issue

- Even for unigrams, unseen words will occur sooner or later
- The longer the n-grams, the sooner unseen cases will occur
- "Colorless green ideas sleep furiously." (Chomsky, 1955)
- As unseen tokens have **zero probability**, $P(W) = 0$, too

- Intuition/Idea: Divert a bit of the overall probability mass of each [seen] token to all possible unseen tokens
- Terminology: model **smoothing** (Chen & Goodman, 1998)

Additive / Laplace Smoothing

- Add a smoothing factor α to all n-gram counts:

$$P(w_i|w_{i-k}^{i-1}) = \frac{count(w_{i-k}^{i}) + \alpha}{count(w_{i-k}^{i-1}) + \alpha V}$$

  - V is the size of the vocabulary
    - the number of unique words in the training corpus
  - $\alpha \leq 1$ usually; if $\alpha = 1$, it is known as **Add-One Smoothing**
  - Very old: first suggested by Pierre-Simon Laplace in 1816
  - But it performs poorly (compared to "modern" approaches)
    - Gale & Church, 1994

`nltk.probability.LidstoneProbDist`
`nltk.probability.LaplaceProbDist` $(\alpha = 1)$

Linear Interpolation Smoothing

- A "Finite Mixture Model" in terms of statistics.
  - a.k.a. **Jelinek-Mercer Smoothing** (1980)
- Sum the conditional probability over all n n-grams (the uni-, bi-, trigram, etc.) at the current token, using the parameter λ for interpolation:

$$P(w_i|w_{i-k}^{i-1}) = \sum_{\kappa=0}^{k} \lambda_\kappa P(w_i|w_{i-\kappa}^{i-1})$$

  - With: $\sum \lambda_\kappa = 1$
- Unseen n-gram probabilities can now be interpolated from their "root"
  - i.e., the n-gram's $0, 1, \ldots, \kappa = n\text{-}1$ interpolated probabilities
- Condition the λ parameter on a held-out development set using Expectation Maximization or any other MLE optimization technique.
- The "baseline model" in Chen & Goodman's (1998) study.

[nltk.probability.WittenBellProbDist]

Advanced Smoothing Techniques

- Use the frequencies of **extremely rare items** to **estimate** the frequencies of **unseen items**
- Approach is generally rather costly to calculate
- **Good-Turing Estimate** (1953 [and WW2 Bletchey Park])
  - **Reallocate** probability mass **from higher-**order n-grams **to** their **lower-order** "stems".

original count

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

N_c: number of unique n-grams of n-gram size c/c+1

  - Needs a MLE (max. likelihood estimate) when **no** higher-order n-gram **count** is **available**.
  - Higher order n-gram counts tend to be **noisy** (imprecise because rare)
  - GTE ≈ "Smoothed Add-One", and the **foundation** of most advances smoothing techniques

nltk.probability.SimpleGoodTuringProbDist

Interpolated Kneser-Ney Smoothing

- A lower-order probability should be smoothed proportional to Γ, the **number of different words that it follows**.
  - In "San Francisco", "Francisco" alone might not be significant, but will lead to a high unigram probability for "Francisco".

let's call this a words history

1. steal some probability... 2. ...reassign it to its lower-order terms w^{i-1} ...

$$P_{KN}(w_i|w_{i-k}^i) = \frac{max\{count(w_{i-k}^i) - \delta, 0\}}{N_k} + \frac{\delta\Gamma(w_{i-k}^{i-1}, w_i)}{N_k} P_{KN}(w_i|w_{i-k+1}^{i-1})$$

$$N_k = \sum count(w_{i-k}^i)$$

the total count of n-grams of size n=k+1

3. ...proportional to the word w_i's history

$$\Gamma(w_{i-k}^{i-1}, w_i) = |\{(w_{i-k}^{i-1}, w_i) : count(w_{i-1}, w_i)\}|$$

the number of different words w_{i-1} that precede w_i

- **Subtracts** some fraction δ of probability mass **from non-zero counts**.
- **Lower-order counts made** significantly **smaller** than their higher-order counts.
  - Reduces the mass of "Francisco" with an artificially high unigram probability (because it almost exclusively occurs as "San Francisco"), so it is less likely to be used to interpolate unseen cases.
- (Chen & Goodman, 1998) interpretation of (Kneser & Ney, 1995)

`nltk.probability.KneserNeyProbDist` (NLTK3 only)

Stupid Backoff Smoothing for Large-scale Datasets

- Brants et al. Large language models in machine translation. 2007

$$P_{SB}(w_i|w_{i-k}^{i-1}) = \begin{cases} P \quad (w_i|w_{i-k}^{i-1}) \quad if\ count(w_{i-k}^{i}) > 0 \\ \alpha P_{SB}(w_i|w_{i-k+1}^{i-1})\ otherwise \end{cases}$$

← the "backoff"

  - Brants set $\alpha = 0.4$ NB that if you were backing off from a tri- to a unigram, you'd apply $\alpha\alpha P(w_i)$

- Intuition is similar to Kneser-Ney Smoothing, but no scaling with a word's history is performed to make the method efficient.
- Efficiently smoothes billions of n-grams (>10^{12} [trillions] of tokens)
- Other **useful large-scale techniques**:
  - Compression, e.g., Huffman codes (integers) instead of (string) tokens
  - Transform (string) tokens to (integer) hashes (at the possible cost of collisions)

(Katz Smoothing) **`[nltk.model.NgramModel]`** (NB: broken in NLTK3!)

The Final Piece: Language Model Evaluation

- How good is the model you just trained?
- Did your update to the model do any good…?
- Is your model better than someone else's?
  - NB: You should compare on the same test set!

Extrinsic Model Evaluation: Error Rate

- Extrinsic evaluation: minimizing the Error Rate
  - evaluates a model's **error frequency**
  - **estimates** the model's per-**word error rate** by comparing the generated sequences to all true sequences (which cannot be established) using a manual classification of errors (therefore, "extrinsic")
  - time consuming, but can evaluate non-probabilistic approaches, too

a "perfect prediction" would evaluate to zero

Intrinsic Model Evaluation: Cross-Entropy / Perplexity

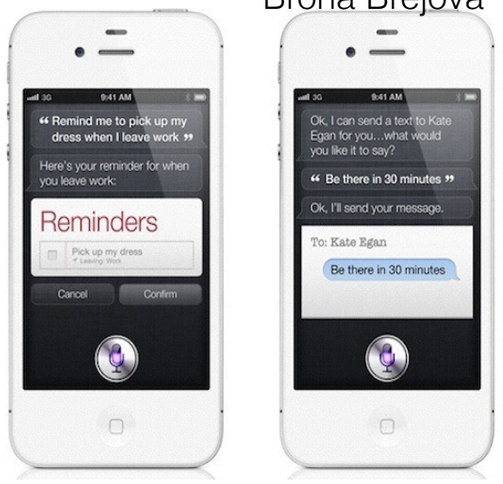
perplexity = "confusion"

- Intrinsic evaluation: minimizing Perplexity (Rubinstein, 1997)
  - compares a model's **probability distribution** (to the "perfect" model)
  - **estimates** a distance based on **cross-entropy** (Kullback-Leibler divergence) between the generated word distribution and the true distribution (which cannot be observed) using the empirical distribution as a proxy
  - efficient, but can only evaluate probabilistic language models and is only an approximation of the model quality

again, a "perfect prediction" would evaluate to zero

Shannon Entropy

Image Source:
WikiMedia Commons,
Brona Brejova



- Answers the questions:
  - “How much information (bits) will I gain when I see w_n?”
  - “How predictable is w_n from its past?”

- Each outcome provides $-log_2 P$ bits of information (“surprise”).
  - Claude E. Shannon, 1948
  - The more probable an event (outcome), the lower its entropy.
  - A certain event (p=1 or 0) has zero entropy (“no surprise”).

- Therefore, the entropy H in our model P for a sentence W is:
  - $H = 1/N \times -\sum P(w_n|w_1, \ldots, w_{n-1}) \log_2 P(w_n|w_1, \ldots, w_{n-1})$

From Cross-Entropy to Model Perplexity

- Cross-entropy compares the model probability distribution $\boldsymbol{P}$ to the true distribution $\boldsymbol{P_T}$:
  - $H(P_T, P, \boldsymbol{W}) = 1/N \times -\sum P_T(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}}) \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$
- CE can be simplified if the "true" distribution is a "stationary, ergodic process": *an "unchanging" language -> a rather naive assumption*
  - $H(P, \boldsymbol{W}) = 1/N \times -\sum \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$
- The relationship between perplexity PPX and cross-entropy is defined as:
  - $PPX(\boldsymbol{W}) = 2^{H(P, \boldsymbol{W})} = P(\boldsymbol{W})^{1/N}$
  - where $\boldsymbol{W}$ is the sentence, $P(\boldsymbol{W})$ is the Markov model, and N is the number of tokens in it

Interpreting Perplexity

- The lower the perplexity, the better ("less surprised") the model.
- Perplexity is an indicator of the number of equiprobable choices at each step.
  - $PPX = 26$ for a model generating an infinite random sequence of Latin letters
  - "`[A-Z]+`"
  - Perplexity produces "big numbers" rather than cross-entropy's "small bits"
  - Typical (bigram) perplexities in **English texts** range from 50 to almost 1000 corresponding to a cross-entropy from about 5.6 to 10 bits/word.
  - Chen & Goodman, 1998
  - Manning & Schütze, 1999

Practical: Develop your own Language Model

- Build a **3rd order** Markov model with **Linear Interpolation** smoothing and evaluate the **perplexity**.
- Note: as of NLTK 3.0a4, language modeling is "broken"
  - try, e.g.: http://www.speech.sri.com/projects/srilm/
- `nltk.NgramModel(n, train, pad_left=True, pad_right=False, estimator=None, *eargs, **ekwds)`
  - `pad_*` pads sentence starts/ends of n-gram models with empty strings
  - `estimator` is a smoothing function that a `ConditionalFreqDist` to a `ConditionalProbDist`
  - `eargs` and `ekwds` are extra arguments and keywords for creating the estimator's CPDs.
- How do smoothing parameters and (smaller…) n-gram size change perplexity?

Write a simple Spelling Corrector in pure Python

- http://www.norvig.com/spell-correct.html
- http://www.norvig.com/spell.py

```
from nltk.corpus import brown
from collections import Count
CORPUS = brown.words()
NWORDS = Counter(w.lower() for w in CORPUS if w.isalpha())
```

“Not to be absolutely certain is, I think, one of the essential things in rationality.”

Bertrand Russell, 1947