Handling Massive N-Gram Datasets Efficiently

Giulio Ermanno Pibiri **Rossano Venturini**

The University of Pisa and ISTI-CNR
Pisa, Italy

ACM Transactions on Information Systems
37(2): 25:1-25:41 (2019)

https://dl.acm.org/doi/10.1145/3302913

Problems

Consider a large textual source.

Problems

Consider a large textual source.

How to represent **all its substrings** of size n = 1, …, N words (for a small N, e.g., 5), using **as few as possible** bits?

Fast **access** to individual n-grams?

Problems

Consider a large textual source.

How to represent **all its substrings** of size n = 1, …, N words (for a small N, e.g., 5), using **as few as possible** bits?

Fast **access** to individual n-grams?

How to **estimate** the probability of occurrence of the n-grams under a given probability model?

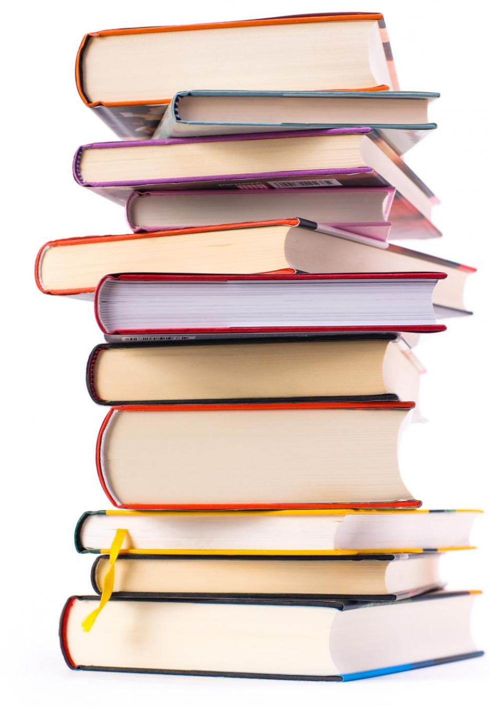
N-Grams Applications

- auto-completion in search engines
- spelling correction
- similarity search
- identification of text reuse and plagiarism
- automatic speech recognition
- machine translation
- and many others….

Many results and softwares available

- **CSTLM** [Shareghi et al., TACL 2016]
- **KenLM** [Heafield, WMT 2011]
- **BerkeleyLM** [Pauls and Klein, ACL 2011]
- **ExpGram** [Watanabe et al., IJCNLP 2009]
- **IRSTLM** [Federico et al., ACL 2008]
- **RandLM** [Talbot and Osborne, ACL 2007]
- **SRILM** [Stolcke, INTERSPEECH 2002]

Numbers are big



Google Books

~6% of the books ever published

n	number of n-grams
1	24,359,473
2	667,284,771
3	7,397,041,901
4	1,644,807,896
5	1,415,355,596

More than 11 billions of n-grams!

Indexing: Context-based Remapped Trie

(1) Store all distinct words in a hash table (the vocabulary)
mapping words to integer ids.
(2) Represent the (mapped) integer n-grams with a *trie* data structure.

The number of distinct words appearing after a given context is small.

Indexing: Context-based Remapped Trie

(1) Store all distinct words in a hash table (the vocabulary)
mapping words to integer ids.
(2) Represent the (mapped) integer n-grams with a *trie* data structure.

The number of distinct words appearing after a given context is small.

Idea: assign a word an integer in [0, *m*),
where *m* is the number of distinct words appearing after a *context*.

Indexing: Context-based Remapped Trie

(1) Store all distinct words in a hash table (the vocabulary)
mapping words to integer ids.
(2) Represent the (mapped) integer n-grams with a *trie* data structure.

The number of distinct words appearing after a given context is small.

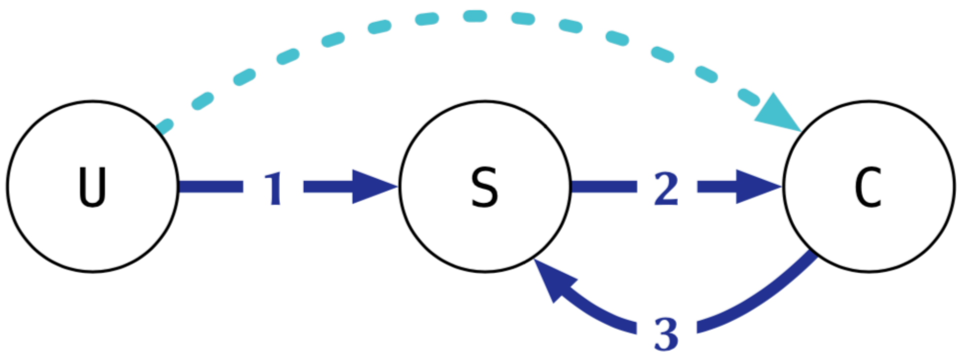
Idea: assign a word an integer in [0, *m*),
where *m* is the number of distinct words appearing after a *context*.

As fast as the fastest competitor, but up to **65%** smaller.

Even smaller than the most space-efficient competitors and up to **5X** faster.

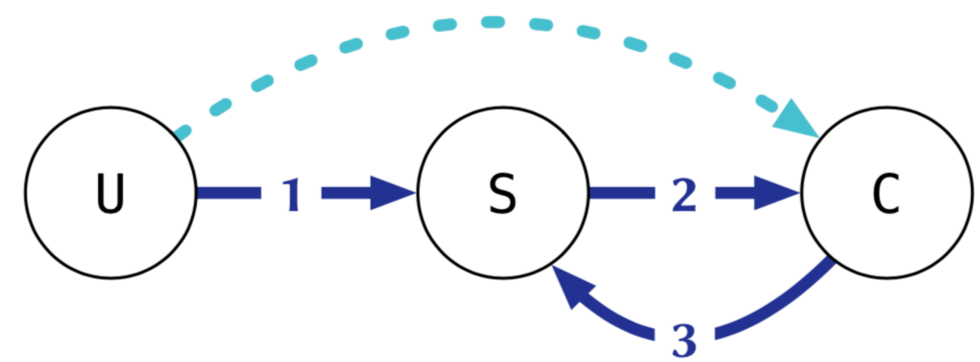
Estimation: 1-Sort Algorithm for Kneser-Ney

To compute the modified *Kneser-Ney* probabilities of the n-grams, the fastest algorithm in the literature uses **3 sorting steps** in external memory [Heafield et al., ACL 2013].



Estimation: 1-Sort Algorithm for Kneser-Ney

To compute the modified *Kneser-Ney* probabilities of the n-grams, the fastest algorithm in the literature uses **3 sorting steps** in external memory [Heafield et al., ACL 2013].

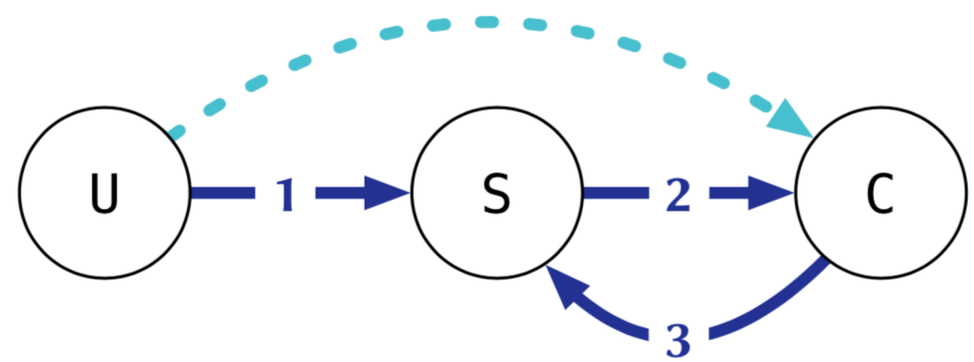


Idea: compute statistics directly over the *context*-sorted n-grams, using space only proportional to the vocabulary.

3-Sort → **1-Sort**

Estimation: 1-Sort Algorithm for Kneser-Ney

To compute the modified *Kneser-Ney* probabilities of the n-grams, the fastest algorithm in the literature uses **3 sorting steps** in external memory [Heafield et al., ACL 2013].



Idea: compute statistics directly over the *context*-sorted n-grams, using space only proportional to the vocabulary.

3-Sort → **1-Sort**

Estimation runs 4.5X faster with billions of strings.

Tongrams — Tons of N-Grams

C++

https://github.com/jermp/tongrams

https://github.com/jermp/tongrams_estimation

Thanks for your attention!