Universal Source Coding

Abhik Rana
Ayan Ghosh
Himadri Mandal
Siddhartha Bhattacharya

Indian Statistical Institute (Kolkata)

Statistical Methods 2 Presentation

Table of Contents

Table of Contents

Formalized description

- **Input.** Alphabet $A = (a_1, a_2, \dots, a_n)$, which is the symbol alphabet of size n. Tuple $W = (w_1, w_2, \dots, w_n)$, which is the tuple of the (positive) symbol weights (usually proportional to probabilities), i.e. $w_i = \text{weight}\,(a_i)\,, i \in \{1, 2, \dots, n\}$.

Formalized description

- **Input.** Alphabet $A = (a_1, a_2, \dots, a_n)$, which is the symbol alphabet of size n. Tuple $W = (w_1, w_2, \dots, w_n)$, which is the tuple of the (positive) symbol weights (usually proportional to probabilities), i.e. $w_i = \text{weight}\left(a_i\right), i \in \{1, 2, \dots, n\}$.
- **Output.** Code $C(W) = (c_1, c_2, \dots, c_n)$, which is the tuple of (binary) codewords, where c_i is the codeword for $a_i, i \in \{1, 2, \dots, n\}$

First Goal

Come up with an instantaneous code

Consider C^* to be an extension of the code given by the input stream represented by $(x_1, x_2, \ldots, x_n)$, then we represent C^* to be $C(x_1)\,C(x_2)\ldots C(x_n)$, i.e., the concatenation of the individual code-words. Need to guarantee that C^* of individual alphabets are prefix free for a lossless decoding scheme.

First Goal

The last demand can be settled if it the code-lengths l_i's of the alphabets satisfy the following condition:

Consider a binary tree, let the branches of the tree represent the symbols of the code-word. Then each code-word is represented by a leaf on the tree. The path from the root traces out the symbols of the code-word. The prefix condition on the code-words implies that no code-word is an ancestor of any other code-word on the tree. Hence, each code-word eliminates its descendants as possible code-words.

First Goal

Let $l_{\max}$ be the length of the longest codeword of the set of codewords. Consider all nodes of the tree at level $l_{\max}$. Some of them are codewords, some are descendants of codewords, and some are neither. A codeword at level l_i has $2^{l_{\max}-l_i}$ descendants at level $l_{\max}$. Each of these descendant sets must be disjoint. Also, the total number of nodes in these sets must be less than or equal to $2^{l_{\max}}$. Hence, summing over all the codewords, we have,

$$\sum 2^{l_{\max}-l_i} \leq 2^{l_{\max}}$$

A sufficient condition

Hence, we obtain the following sufficient condition for satisfying the first goal:

$$\sum 2^{-l_i} \leq 1$$

Second Goal

Guaranteeing the optimality

We have a standard optimization problem here:

$$\text{Minimize } L = \sum p_i l_i$$

over all naturals $l_1, \ldots, l_m$ satisfying to

$$\sum 2^{-l_i} \leq 1$$

Second Goal

A simple analysis suggests the form of the minimizing l_i^*. We neglect the integer constraint on l_i and assume equality in the constraint. Hence, we can write the constrained minimization using Lagrange multipliers as the minimization of

$$J = \sum p_i l_i + \lambda \left(\sum 2^{-l_i} \right)$$

Result

Differentiating J and setting the derivatives to 0 we obtain

$$p_i = 2^{-l_i}$$

and yielding the theoretical expected most optimal code-length to be

$$L^* = H(X)$$

Second Goal

A trite calculation yields that the expected length L of any instantaneous binary code for a random variable X is greater than or equal to the entropy $H(X)$; that is, $L \geq H(X)$ with equality if and only if $2^{-l_i} = p_i$

A sufficient condition

If the process is stationary, then $\frac{H(X_1, X_2, \ldots, X_n)}{n} \to \mathcal{H}(X)$ and the expected description length tends to the entropy rate as $n \to \infty$. So we might be inclined to call an encoding scheme to be "optimal" $\iff$ the average description length approaches the entropy rate of the source.

Optimality of Huffman

- Recall that the problem is given frequencies $f_1, \ldots, f_n$ to find the optimal prefix-free code that minimizes

$$\sum_{i}^{n} f_i \cdot \text{ length of encoding of the } i\text{-th symbol.}$$

Optimality of Huffman

- Recall that the problem is given frequencies $f_1, \ldots, f_n$ to find the optimal prefix-free code that minimizes

$$\sum_i^n f_i \cdot \text{ length of encoding of the } i\text{-th symbol.}$$

- From the condtion of sufficiency of the first goal, this is the same as finding the full binary tree with n leaves, one per symbol in $1, \ldots, n$, that minimizes

$$\sum_{i=1}^n f_i \cdot (\text{ depth of leaf of the } i\text{-th} symbol)$$

Proposition I

There's an optimal tree where the two smallest frequency symbols are at the deepest level in the tree and are siblings of the same ancestor.

Optimality of Huffman

Proposition II

Huffman's encoding gives an optimal choice of the prefix-tree.

- The proof is by induction on n, the number of symbols. The base case $n = 2$ is trivial since there's only one full binary tree with 2 leaves.

Optimality of Huffman

Proposition II

Huffman's encoding gives an optimal choice of the prefix-tree.

- The proof is by induction on n, the number of symbols. The base case $n = 2$ is trivial since there's only one full binary tree with 2 leaves.
- Inductive Step: We will assume the claim to be true for any sequence of $n - 1$ frequencies and prove that it holds for any n frequencies. Let $f_1, \ldots, f_n$ be any n frequencies. Assume without loss of generality that $f_1 \leq f_2 \leq \ldots \leq f_n$ (by relabeling). By Claim 1, there's an optimal tree T for which the leaves marked with 1 and 2 are siblings. Let's denote the tree that Huffman strategy gives by H. Note that we are not claiming that $T = H$ but rather that T and H have the same cost. Here we define cost as $\sum f_i \cdot \text{depth } f_i$.

Optimality of Huffman

- We will now remove both leaves marked by 1 and 2 from T, making their ancestor a new leaf with frequency $f_1 + f_2$. This gives us a new binary tree T' on $n-1$ leaves with frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$. We do the same for the Huffman tree giving us a tree H' on $n-1$ leaves with frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$. Note that H' is exactly the Huffman tree on frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$ by definition of Huffman's strategy. By the induction hypothesis,

$$\text{cost}\left(H'\right) = \text{cost}\left(T'\right).$$

Optimality of Huffman

- We will now remove both leaves marked by 1 and 2 from T, making their ancestor a new leaf with frequency $f_1 + f_2$. This gives us a new binary tree T' on $n-1$ leaves with frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$. We do the same for the Huffman tree giving us a tree H' on $n-1$ leaves with frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$. Note that H' is exactly the Huffman tree on frequencies $f_1 + f_2, f_3, f_4, \ldots, f_n$ by definition of Huffman's strategy. By the induction hypothesis,

$$\text{cost}\left(H'\right) = \text{cost}\left(T'\right).$$

- Observe further that

$$\text{cost}\left(T'\right) = \text{cost}(T) - \text{cost}\left(f_1 + f_2\right)$$

Optimality of Huffman

- since to get T' from T we replaced two nodes with frequencies f_1 and f_2 at some depth d with one node with frequency $f_1 + f_2$ at depth $d - 1$. This lowers the cost by $f_1 + f_2$. Similarly,

$$\text{cost}\left(H'\right) = \text{cost}(H) - \text{cost}\left(f_1 + f_2\right).$$

Optimality of Huffman

- since to get T' from T we replaced two nodes with frequencies f_1 and f_2 at some depth d with one node with frequency $f_1 + f_2$ at depth $d - 1$. This lowers the cost by $f_1 + f_2$. Similarly,

$$\text{cost}\left(H'\right) = \text{cost}(H) - \text{cost}\left(f_1 + f_2\right).$$

- Combining the three equations together we have that

$$\text{cost}(H) = \text{cost}\left(H'\right) + \text{cost}\left(f_1 + f_2\right) = \text{cost}\left(T'\right) + \text{cost}\left(f_1 + f_2\right) = \text{cost}(T$$

Table of Contents

Naive Idea

Naive Idea #1: Estimate using the Empirical CDF

Data comes in character by character. We don't know the source distribution of the data. Therefore, we try to estimate the source distribution and use the many techniques available thereon. We can wait till the n^{th} character is inputted and use that to find the empirical CDF $\hat{P}$ and then use Huffman Encoding on it.

Result

Huffman Encoding a truncated input stream based on its Empirical Distribution $\hat{P}$ is *not* a Universal Source Code.

Proof of Result

Huffman Encoding a truncated input stream based on its Empirical Distribution

Let Ω be the character set. Consider a truncated input stream I of length n. Let A consist of the Huffman encoded words based on I. Let the empirical probability distribution be $\hat{P}$.
Mathematically the encoder is

$$f_n(x) = \begin{cases} \text{Huffman encoding of x} & x \in A \\ 0 & x \notin A \end{cases}$$

The decoder is the Huffman decoder.

Proof of Result

Contd.

Assuming the code $X_1, X_2, \cdots X_n$ are coming from an arbitrary distribution Q, denote the probability of a subset C of Ω^n coming from the distribution Q as $Q^n(C)$.

$$P_e^{(n)} = 1 - Q^n(A) \geq 1 - 2^{-nD_{KL}(\hat{P}||Q)}$$

$$D_{KL}(\hat{P}||Q) = \sum_{\omega} \hat{P}(\omega) log \left(\frac{\hat{P}(\omega)}{Q(\omega)} \right)$$

which is independent of n.
Therefore, $P_e^{(n)} \to 1$ as $n \to \infty$.

Idea 2:

Formulation

Let $\{p_1, p_2, \cdots, p_m\}$ be a set of pmfs. Define the universe of probabilities to be

$$\mathcal{U} = \left\{ \alpha_1 \cdot p_1 + \cdots + \alpha_m \cdot p_m \middle| \alpha_1 + \cdots + \alpha_m = 1, 0 \leq \alpha_i \leq 1 \right\}$$

Let $\chi_d(P, Q)$ be a measure of "difference" between two probability distributions P, Q. Let $\mathcal{U}$ be a universe of probabilities. The two centers of $\mathcal{U}$ are $\mathcal{P}_1$ and $\mathcal{P}_2$

$$\mathcal{P}_1 = \arg\min_{P} \mathop{\mathbb{E}}_{Q \sim \mathcal{U}} [\chi_d(P, Q)]$$

$$\mathcal{P}_2 = \inf_{P \sim U} \sup_{Q \sim U} \chi_d(P, Q)$$

$\mathcal{P}_1, \mathcal{P}_2$ give best average performance and best worst performance, respectively.

Idea 2:

The measure of the difference between the two probability distributions we tried to work with was $\chi_d(P, Q) = \mathrm{D}_{\mathrm{KL}}(Q||P)$. The idea is the following:

- Figure out a good way to fix the universe $\mathcal{U}$.

We believe there is promise in this idea, and should be explored further. However, we figured out new perspectives to look at this problem and so didn't devote any more time into this.

Idea 2:

The measure of the difference between the two probability distributions we tried to work with was $\chi_d(P, Q) = \mathrm{D}_{\mathrm{KL}}(Q||P)$. The idea is the following:

- Figure out a good way to fix the universe $\mathcal{U}$.
- Find the centers $\mathcal{P}_!$, $\mathcal{P}_2$ using gradient descent or such.

We believe there is promise in this idea, and should be explored further. However, we figured out new perspectives to look at this problem and so didn't devote any more time into this.

Idea 2:

The measure of the difference between the two probability distributions we tried to work with was $\chi_d(P, Q) = \mathrm{D}_{\mathrm{KL}}(Q||P)$. The idea is the following:

- Figure out a good way to fix the universe $\mathcal{U}$.
- Find the centers $\mathcal{P}_!$, $\mathcal{P}_2$ using gradient descent or such.
- Obtain encoding schemes H_1, H_2 accordingly. Choose as need be.

We believe there is promise in this idea, and should be explored further. However, we figured out new perspectives to look at this problem and so didn't devote any more time into this.

An extension to a result.

Extension of the Result by Csiszar and Körner

For any $R > 0$, an increasing sequence $R_n \uparrow R$, determines a universal source code of length n having size $O(2^{nR})$.

Proof

Fix $R > 0$. For each sequence $x \in \Omega^n$, denote its empirical distribution by $P(x)$. Denote a sequence $y \in P(x)$ if it has same distribution as that of x. Denote the cardinality of such a set as $T(P(x))$ Denote the set
$A = \{x \in \Omega^n : H(P(x)) \leq R_n\}$.
$|A| = \sum_{y:y \in P(x), x \in A} 1 = \sum_{x \in A} T(P(x)) \leq \sum_{x \in A} 2^{nH(P(x))} \leq 2^{nR_n}(n+1)^{|\Omega|}$ which gives $|A| = O(2^{nR})$
Now to show this scheme is universal. Define the encoder

$$f_n(x) = \begin{cases} \text{index of x in A} & x \in A \\ 0 & x \notin A \end{cases}$$

The decoder maps the index back to A. The underlying distribution is Q where $H(Q) < R$.

Proof

Therefore
$P_e^{(n)} = 1 - Q^n(A) = \sum_{P:H(P)>R_n} Q^n(T(P)) \leq (n+1)^{|\Omega|} \max_{P:H(P)>R_n} Q^n(T(P))$.
Now we use the bound $Q^n(T(P)) \leq 2^{-nD(P||Q)}$ to get
$P_e^{(n)} \leq (n+1)^{|\Omega|} 2^{-n \min_{P:H(P)>R_n} D(P||Q)}$.
Since $R_n \uparrow R$, we can say that for some N, $\forall n \geq N$, $H(Q) \leq R_n$.
This gives $H(P_n) = H(\arg\max_{P:H(P)>R_n} Q^n(T(P))) > H(Q)$.
Hence by Gibbs inequality, $\min_{P:H(P)>R_n} D(P||Q) >$ for all $n > N$. Therefore $P_e^{(n)} \to 0$, completing the proof.

Table of Contents

Lempel-Ziv-Welch

LZW

Define dictionary D with all characters in the input alphabet and their encodings. Define a running string variable Pattern, and iteratively add the next character to it. If the resulting string is already in D, continue reading characters until you find a string that is *not* in the dictionary D. Add it to D with its encoding. Then, encode Pattern as $D[\text{Pattern}[0:-1]] + \text{Pattern}[-1]$. Continue.

Bubbly LZW: Permutation optimization

Obtain the normal **LZW** dictionary D, and find the number of times each encoding is obtained, for a phrase X call this Counter(X). Sort D in decreasing order according to the key $f(v) = \text{len}(v) \cdot \text{Counter}(v)$.

Define $P_D = \text{SortedD}^{-1}$ such that $\text{SortedD}[P_D(x)] = x$. P_D is the optimized permutation.

Obstacles

Okay, let's try to figure out the first obstacle now.

- Parent: (nX), where n is an encoding whose definition can not be realized given the text before this phrase and X is some character.

Example with Ambiguity

Text = #a3b1b2a1a#b4a
Phrase Decomposition = (#a)(3b)(1b)(2a)(1a)(#b)(4a)
Here "(1a)" is an ambiguous phrase whose meaning cannot be derived by the decoder using the phrases before it.

Obstacles

Okay, let's try to figure out the first obstacle now.

- Parent: (nX), where n is an encoding whose definition can not be realized given the text before this phrase and X is some character.
- Child: (nX), where n is an encoding whose definition can be realized using the text before this phrase and X is some character.

Example with Ambiguity

Text = #a3b1b2a1a#b4a
Phrase Decomposition = (#a)(3b)(1b)(2a)(1a)(#b)(4a)
Here "(1a)" is an ambiguous phrase whose meaning cannot be derived by the decoder using the phrases before it.

Obstacles

Okay, let's try to figure out the first obstacle now.

- Parent: (nX), where n is an encoding whose definition can not be realized given the text before this phrase and X is some character.
- Child: (nX), where n is an encoding whose definition can be realized using the text before this phrase and X is some character.
- New: (#X), where X is some character.

Example with Ambiguity

Text = #a3b1b2a1a#b4a
Phrase Decomposition = (#a)(3b)(1b)(2a)(1a)(#b)(4a)
Here "(1a)" is an ambiguous phrase whose meaning cannot be derived by the decoder using the phrases before it.

Characterisation of The First Ambiguity

Theorem

A phrase P is the first ambiguity $\iff$ the phrase P is the last parent phrase before the first child phrase (OR) the last parent phrase before the first new phrase.

Recursively solve ambiguities

```
1: procedure SolveAmbiguity(Encoded)
2:     while IsAmbiguity: True do
3:         Find the first occurence of one of the two:
4:         (parent)(child)
5:         (parent)(new)
6:         Perform the definition of the (parent)
7:         Mark the next phrase a parent, and continue.
8:     end while
9: end procedure
```

Parent

Defining a Parent

By defining a parent phrase P, we mean giving the decoder some information that can then be used to determine the encoding of P unambiguously. Here we used the most naive idea: we defined a parent phrase P by attaching its definition to it.

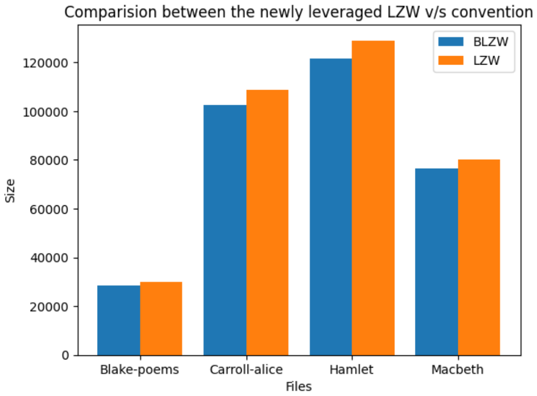

Figure: Improvement on Language Datasets

Table of Contents

Introduction to Adaptive Huffman Coding

- **Adaptive Huffman Coding** is a dynamic data compression algorithm that adjusts the Huffman tree as new data is processed.

Introduction to Adaptive Huffman Coding

- **Adaptive Huffman Coding** is a dynamic data compression algorithm that adjusts the Huffman tree as new data is processed.
- Unlike static Huffman coding, which requires two passes over the data, adaptive Huffman coding works in a single pass.

Introduction to Adaptive Huffman Coding

- **Adaptive Huffman Coding** is a dynamic data compression algorithm that adjusts the Huffman tree as new data is processed.
- Unlike static Huffman coding, which requires two passes over the data, adaptive Huffman coding works in a single pass.
- It is particularly useful for compressing data streams where the statistical properties of the data can change over time.

Key Concepts

- **Huffman Tree**: A binary tree used to assign variable-length codes to symbols based on their frequencies.

Key Concepts

- **Huffman Tree**: A binary tree used to assign variable-length codes to symbols based on their frequencies.
- **NYT (Not Yet Transmitted) Node**: A special node representing symbols that have not been seen yet.

Key Concepts

- **Huffman Tree**: A binary tree used to assign variable-length codes to symbols based on their frequencies.
- **NYT (Not Yet Transmitted) Node**: A special node representing symbols that have not been seen yet.
- **Incremental Updates**: The Huffman tree is updated incrementally as each new symbol is processed.

Pseudo Code

Adaptive Huffman Coding

```
 1: Initialize the Huffman tree with NYT node.
 2: for each symbol in the input data do
 3:     if symbol is not in the tree then
 4:         Add symbol to the tree with NYT node.
 5:     else
 6:         Increment the frequency of the symbol.
 7:     end if
 8:     Adjust the tree to maintain Huffman property.
 9: end for
10: Use the updated tree for encoding/decoding.
```

First-Order Markov Model

Definition

Token: Tokens may be characters, words, n-grams, or other units. Their probabilities are estimated by their relative frequencies, measured from some sample-which is typically all the text seen so far

- **Markov Property**:
  - Assumes that the probability of each token depends only on the immediately preceding token.
- **Transition Probabilities**:
  - $P(t_{i+1}|t_i)$ represents the probability of transitioning from token t_i to token t_{i+1}.
  - Calculated using the frequencies of token pairs in the data.
- **Model Representation**:
  - Can be represented as a transition matrix.
  - Each entry $P(t_j|t_i)$ represents the transition probability from t_i to t_j.

Application in Text Compression

- **Predictive Coding**:
  - Uses transition probabilities to predict the next token.
  - Improves compression efficiency by encoding more probable tokens with fewer bits.
- **Adaptive Arithmetic Coding**:
  - Utilizes a first-order model to dynamically update probabilities as tokens are processed.
  - Encodes sequences based on the changing probabilities, achieving better compression.

Example

Given the sequence "ABABBA":

Transition probabilities: $P(B|A) = 1$, $P(A|B) = 2/3$, $P(B|B) = 1/3$.

These probabilities guide the encoding process for efficient compression.

Escape Probabilities

Definition

The outcome that is encoded when the next word is not in the list is called an escape, since it signals an escape to a different scheme-character instead of word coding.

Adaptive Text Compression

- Encoders and decoders share a statistical model.

Adaptive Text Compression

- Encoders and decoders share a statistical model.
- Model adapts dynamically based on the preceding message.

Adaptive Text Compression

- Encoders and decoders share a statistical model.
- Model adapts dynamically based on the preceding message.
- Statistical model components: structure (conditioning classes) and parameters (probabilities).

Methods for Computing Escape Probabilities

- Laplace's Law of Succession: initial method for zero-frequency problem.

Methods for Computing Escape Probabilities

- Laplace's Law of Succession: initial method for zero-frequency problem.
- Generalized Law of Succession: extends Laplace's method to q-symbol alphabets.

Methods for Computing Escape Probabilities

- Laplace's Law of Succession: initial method for zero-frequency problem.
- Generalized Law of Succession: extends Laplace's method to q-symbol alphabets.
- Method A: allocating part of the code space for the escape event.

Methods for Computing Escape Probabilities

- Laplace's Law of Succession: initial method for zero-frequency problem.
- Generalized Law of Succession: extends Laplace's method to q-symbol alphabets.
- Method A: allocating part of the code space for the escape event.
- Method B: classifying a symbol as novel unless it has occurred twice.

Poisson Process Model

- Assumption: tokens appear according to a Poisson process.

Poisson Process Model

- Assumption: tokens appear according to a Poisson process.
- Model parameters: expected number of occurrences (λ_i).

Poisson Process Model

- Assumption: tokens appear according to a Poisson process.
- Model parameters: expected number of occurrences (λ_i).
- Extrapolation from sample size n to larger size $N = (1 + \delta)n$.

Poisson Process Model

- Assumption: tokens appear according to a Poisson process.
- Model parameters: expected number of occurrences (λ_i).
- Extrapolation from sample size n to larger size $N = (1 + \delta)n$.
- Novelty probability and token probability calculation.

Laplace's Law of Succession

- **Overview**:
  - Laplace's Law of Succession is a method for estimating the probability of an event that has not been observed yet.
- **Formula**:
  - Probability of a novel event: $\hat{p} = \frac{c+1}{n+2}$
  - Where:
    - $c =$ Count of observed occurrences of the event.
    - $n =$ Total number of observations.

Coding it

The law of succession can be applied to the coding of a binary source as follows. Suppose $n = c_1 + c_2$ tokens have been encountered so far, of which c_1 are zeros and c_2, are ones. Then the prediction probabilities are estimated as $P(\text{next token } 0) = \frac{c_1+1}{n+2}$

Generalized Laplace

We are introducing this to remove the assumption of binary sources

Definition

Suppose that there are Q event types instead of 2.And out of the N observations suppose there are C_1 of type 1 and so on s.t. $C_1 + \cdots + C_q = N$.Assuming all C_i are equally likely
$P(\text{Next token of type Ci}) = \frac{c_i+1}{n+2}$

Analysis of Poisson Process

- **Poisson Process**:
  - A stochastic process that models the occurrence of events over a fixed interval of time or space.
  - Events occur independently and at a constant average rate.
- **Poisson Distribution**:
  - Describes the probability of a given number of events occurring in a fixed interval.
  - Probability Mass Function (PMF):

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

  - λ is the average rate (mean) of occurrences.

Analysis Contd.

Defining

Suppose $G(\lambda)$ is the Empirical CDF of $\lambda_1, \cdots, \lambda_q$.By our observations we can find the number of types observed k times in a sample of n denoted by t_k.

$$\tau_k = \mathrm{E}(t_k) = q \int_0^\infty \frac{e^{-\lambda}\lambda^k}{k!} d(G(\lambda))$$

Now we want to extrapolate to the larger sample of size N

Extrapolating

The expected number of new types

$$q \int_0^\infty e^{-\lambda}(1 - e^{-\lambda\theta}) d(G(\lambda))$$

Continued..

Substituting the Taylor expansion of $e^{-\lambda\theta}$we get the following results

Expected Number of New Types

Using the previous result We obtain the Expected Number of new types $\tau_1\theta - \tau_2\theta^2 \cdots$ We Empirically substitute the data $t_1, \cdots, t_n$ n sample tokens for their expected values to get the following

$$\sum_{i=1}^{\infty} t_i(-1)^{i+1}\theta^i$$

Special Case: $N = n + 1$ which gives us the following

$$\mathcal{P}(\text{Next event will be novel}) = \frac{t_1}{n} - \frac{t_2}{n^2} \cdots$$

For computational processes we may truncate the the sum to get

$$\mathcal{P}(\text{Next Event will be novel}) = \frac{t_1}{n}$$

Comparison Of Empirical Methods

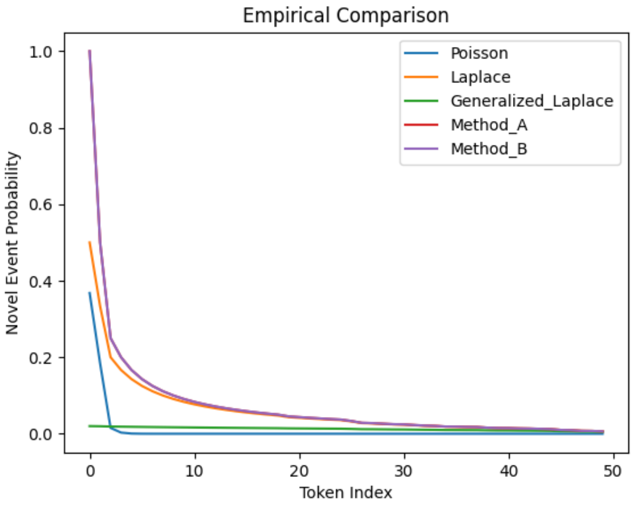

Figure: Comparison

Conclusion and Application

Incorporating these methods in Adaptive Huffman Coding it would not improve the Time Complexity but it would clearly improve the compression ratio since it is also dynamically predicting the event of a new n-gram/character/word probability Clearly Poisson is the most efficient Empirically For Integrating The Poisson Process in Adaptive Huffman Coding we take the Special Case $N = n + 1$ since every character is getting updated in the Huffman tree dynamically

Thanks

Thanks!