Deplump: A streaming lossless compressor

Nicholas Bartlett Frank Wood

Department of Statistics, Columbia University, New York, USA

Abstract

1 Algorithm

Given an ordered symbol set Σ, probabilistic compression algorithms work by using a generative model to predict a sequence of symbols. The predictive distribution function is then used as the parameter in a range encoder to compress the stream. The details of a range encoder implementation are not included here, we only note that if the predictive distribution function is F and the next symbol in the stream is s then the parameters required by the range encoder are $F(s-1)$ and $F(s)$. In the algorithm the function RangeEncode() implements the encoding and returns a bit sequence (possibly null). The use of a cumulative distribution function is well defined since the symbols are ordered. The notation $s-1$ refers to the symbol prior to s in the symbol ordering. In order to decompress the stream the exact same predictive model will need to be built from the compressed stream. This requires that the model estimate prior to compressing s_n is a function of fixed parameters and the symbols $[s_0, s_1, \ldots, s_{n-1}]$ because those are the only symbols available to the decompressor for decompressing s_n.

The algorithm operates primarily on a suffix tree. A suffix tree is a data structure for keeping track of unique suffices of a set of strings. The tree structure arranges the suffices hierarchically which makes it easy to search. In the case of a single stream the set of strings to consider is the set of contexts $\{[], [s_0], [s_0, s_1], [s_0, s_1, s_2], \ldots\}$. Each node of the suffix tree corresponds to a context of the form $[s_m, \ldots, s_{m+k}]$. In general we will use the notation $\mathcal{N}$ to refer interchangeably to a node object and the context to which the node corresponds. The function CreateNode($\mathcal{N}, \mathcal{M}$) makes explicit the creation of a node $\mathcal{N}$ associated with context $\mathcal{N}$ and with parent $\mathcal{M}$. We use the function PA($\mathcal{N}$) to refer to the parent of node $\mathcal{N}$.

Each node object $\mathcal{N}$ contains two integer counts for each $s \in \Sigma$, c_s, t_s. The notation c and t refer to the marginal counts in a node, namely $\sum_{s\in\Sigma} c_s$ and $\sum_{s\in\Sigma} t_s$. Each node also has a discount d associated with it. The discount associated with $\mathcal{N}$ is a function of $\mathcal{D}$, the fixed discount parameters of the model, $|\mathcal{N}|$, and $|\text{PA}(\mathcal{N})|$. The function GetDiscount describes how the discount of a node $\mathcal{N}$ is calculated. Suffix tree data structures use a reference sequence to maintain the unique suffices in the tree. We use the notation $\mathcal{RS}$ to refer to this reference sequence. Each node $\mathcal{N}$ in a suffix tree contains two indices into $\mathcal{RS}$ from which the context specific to node $\mathcal{N}$ can be reconstructed. If the indices for $\mathcal{N}$ are i and j, then the context associated with $\mathcal{N}$ is $\mathcal{RS}[i:j]$.

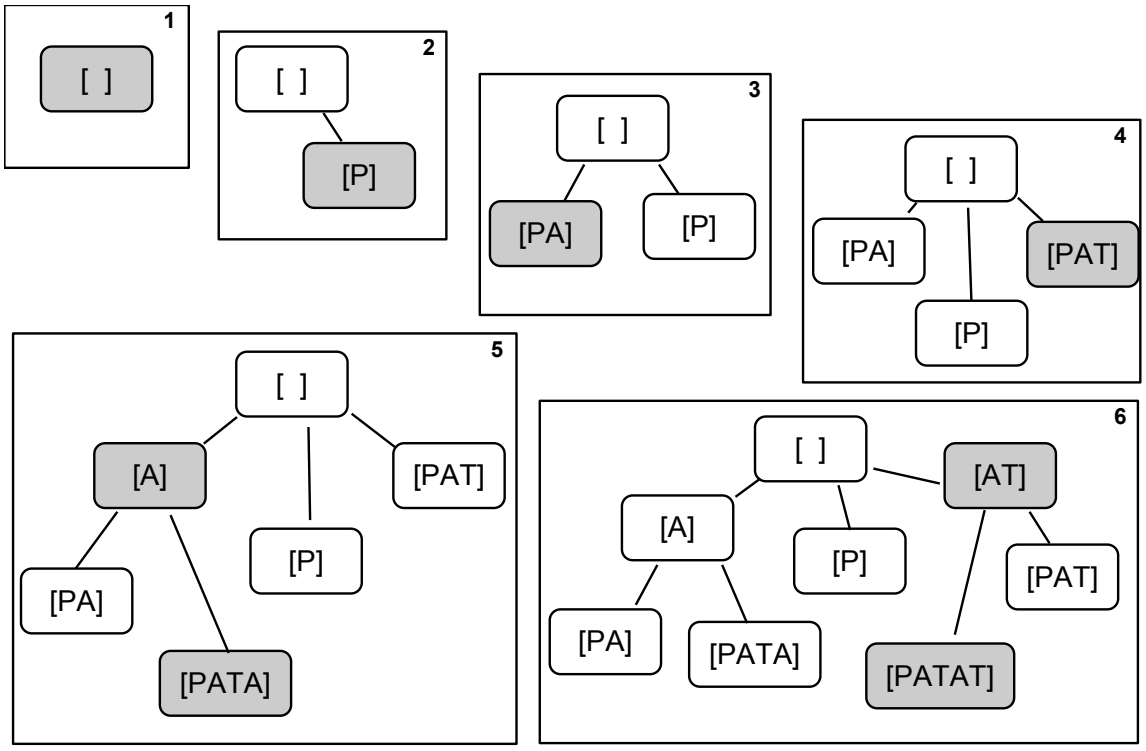

Figure 1: Construction of suffix tree for string "PATAT". In each frame the new nodes are shaded in gray.

The reference sequence $\mathcal{RS}$ is implemented as a linked list. Therefore, the indexing of a suffix tree node into $\mathcal{RS}$ includes a pointer to a linked list node , an offset within that node and the length of the respective context. Each node of the linked list must also contain a list of suffix tree nodes which reference it. The reference sequence grows as the length of the compressed sequence grows and must be shortened as the algorithm progresses. The shortening of $\mathcal{RS}$ is made explicit by the σ operator in CDFNextSymbol, which returns the input sequence shortened by removing the fist element. When $\mathcal{RS}$ is shortened, nodes in the suffix tree which reference removed sections are no longer usable and must be removed from the tree to prevent a memory leak. The cost of these operations can be amortized by shorting $\mathcal{RS}$ in chunks by removing nodes of the linked list. Without a list of suffix tree nodes which reference each linked list node, deletion of the unusable nodes requires a search over the tree which is prohibitive for large trees. To minimize the impact of rendering nodes unusable by shortening the reference sequence, nodes should be updated to point to the most recent part of $\mathcal{RS}$ as possible.

Since the model must be estimated incrementally, the suffix tree must also be incrementally constructed. Construction of the tree is handled by the function GetNode in Algorithm 1. An illustration of the incremental construction of a suffix tree can be seen in Figure 1 for the toy sequence [PATAT]. In frame 4 the function GetNode assigns [ ] to $\mathcal{M}$ and then [PAT] to $\mathcal{S}$ with $\mathcal{M} = \text{PA}(\mathcal{S})$. In Frame 5 GetNode assigns [PA] to $\mathcal{M}$, but then must assign [A] to $\mathcal{P}$ with $\mathcal{P} = \text{PA}(\mathcal{M})$. $\mathcal{S}$ is then created by CreateNode$([\text{PATA}], \mathcal{P})$. In

each frame the first step is to find $\mathcal{M}$, which can be achieved by descending an appropriate path of the suffix tree. All of the nodes on the path to $\mathcal{M}$ and possibly $\mathcal{M}$ itself can have the indices into $\mathcal{RS}$ updated to point to a later section of the reference sequence. Finally, although not made explicit in the algorithm, it usually makes sense to limit the maximum length of a context in order to limit the depth of the suffix tree. In Section **??** we empirically investigate performance for three values of depth.

For each s in the input sequence the function CDFNextSymbol is used to obtain the predictive cumulative distribution function values of interest. After encoding the symbol first step to updating the model estimate is performed by the function UpdateCountsAndDiscounts. Starting at node $\mathcal{N}$ and progressing up to the root of the tree, c_s is incremented if t_s was incremented in the node below. If c_s is incremented, a stochastic decision is made to increment t_s. The gradients for the discount parameters $\mathcal{D}$ are updated by the function UpdateDiscountParameterGradients. Finally, if c is larger than k in any of the nodes, the counts c_s and potentially t_s are reduced by the function ThinCounts. The parameter k is a fixed parameter of the model.

First, c_s is incremented in node $\mathcal{N}$. A

to the root a stochastic decision is made to increment c_s

Things to make sure to include

Assumed ordering of the symols in the symbol set. Describe paremetrization of discounts Mention learning rate η Describe RangeEncode function and return Describe σ operator Describe size of tree L Describe length of $\mathcal{RS}$ Describe the deletion of leaf nodes uniformly at random Describe pa($\mathcal{N}$) operator Desribe count notation Describe CreateNode function Mention arrays indexed from 0 Describe Draw multinomial Discribe k in thin counts

Algorithm 1 Deplump

procedure DEPLUMP/PLUMP($\mathcal{IS}, k$)
  $\mathcal{RS} \leftarrow [\,]$ ▷ reference sequence
  Initialize [ ] node of $\mathcal{T}$ ▷ suffix tree
  $nc \leftarrow 1$ ▷ node count
  $\mathcal{D} \leftarrow \{d_0, d_1, d_2, \ldots, d_{10}, \alpha\}$ ▷ discount parameters
  $\mathcal{G} \leftarrow \vec{0}$ ▷ discount parameter gradients, $|\mathcal{G}| = |\mathcal{D}|$
  $\mathcal{OS} \leftarrow [\,]$ ▷ output sequence
  if Plump **then**
    $h \leftarrow$ InitializePointOnCDF($\mathcal{IS}$)
  end if
  for i = 0: $|\mathcal{IS}|$ **do**
    if Plump **then**
      $[h, s, F(s-1), F(s)] \leftarrow$PointOnCDFNextSymbol($h, \mathcal{T}, \mathcal{RS}$)
      $b \leftarrow s$
    else
      $s = \mathcal{IS}[i]$
      $[F(s-1), F(s)] \leftarrow$ CDFNextSymbol($s_i, \mathcal{T}, \mathcal{RS}$)
      $b \leftarrow$ RangeEncode($F(s-1), F(s)$)
    end if
    $\mathcal{OS} \leftarrow [\mathcal{OS}\ b]$ ▷ append b to output sequence
    $\mathcal{D} \leftarrow \mathcal{D} + \mathcal{G}\eta/(F(s) - F(s-1))$ ▷ update discount parameters
    $\mathcal{G} \leftarrow \vec{0}$ ▷ reset gradients to zero
    $\mathcal{RS} \leftarrow [\mathcal{RS}\ s_i]$ ▷ append symbol to reference sequence
  end for
  return $\mathcal{OS}$
end procedure
function CDFNEXTSYMBOL($s, \mathcal{T}, \mathcal{RS}$)
  while $|\mathcal{RS}| \geq 100L$ **do**
    Delete nodes referencing $\mathcal{RS}$[0]
    $\mathcal{RS} \leftarrow \sigma(\mathcal{RS})$
  end while
  while $nc > (L-2)$ **do**
    Delete leaf node uniformly at random
  end while
  $\mathcal{N} \leftarrow$ GetNode($\mathcal{RS}, \mathcal{T}$)
  $\pi \leftarrow$ PMF($s, \mathcal{N}, \vec{0}, 1.0$) ▷ $|\vec{0}| = |\Sigma|$
  $F(s-1) \leftarrow \Sigma_{i=1}^{s-1}\pi_i$
  $F(s) \leftarrow F(s-1) + \pi_s$
  UpdateCountsAndDiscountGradients($\mathcal{N}, s, F(s) - F(s-1)$,TRUE)
  return $[F(s-1), F(s)]$
end function

Algorithm 2 Deplump Continued

function POINTONCDFNEXTSYMBOL($h, \mathcal{T}, \mathcal{RS}$)
  while $|\mathcal{RS}| \geq 100L$ **do**
    Delete nodes referencing $\mathcal{RS}$[0]
    $\mathcal{RS} \leftarrow \sigma(\mathcal{RS})$
  end while
  while $nc > (L-2)$ **do**
    Delete leaf node uniformly at random
  end while
  $\mathcal{N} \leftarrow$ GetNode($\mathcal{RS}, \mathcal{T}$)
  $\pi \leftarrow$ PMF($\mathcal{N}, \vec{0}, 1.0$) ▷ $|\vec{0}| = |\Sigma|$
  $F(s) \leftarrow 0$
  $s \leftarrow 0$
  while $F(s) < h$ **do**
    $i \leftarrow i + 1$
    $F(s) \leftarrow F(s) + \pi_s$
  end while
  $F(s-1) \leftarrow F(s) - \pi_s$
  Update h **return** $[h, s, F(s-1), F(s)]$
end function
function GETDISCOUNT($\mathcal{N}$)
  $d = 1.0$
  for $i = (|\text{PA}(\mathcal{N})| + 1) : |\mathcal{N}|$ **do**
    if $i \leq 10$ **then**
      $d \leftarrow d\mathcal{D}[i]$ ▷ multiply by discount parameter i
    else
      $d \leftarrow d\mathcal{D}[10]^{\mathcal{D}[11]^i}$
    end if
  end for
  return d
end function
function GETNODE($\mathcal{S}, \mathcal{T}$)
  Find the node $\mathcal{M}$ in the suffix tree sharing the longest suffix with $\mathcal{S}$.
  if $\mathcal{M}$ is a suffix of $\mathcal{S}$ **then**
    if $\mathcal{S} = \mathcal{M}$ **then**
      return $\mathcal{M}$
    else
      $\mathcal{S} \leftarrow$ CreateNode($\mathcal{S}, \mathcal{M}$)
      return $\mathcal{S}$
    end if
  else
    $\mathcal{P} \leftarrow$ FragmentNode($\mathcal{M}$, $\mathcal{S}$)
    PA($\mathcal{M}$) $\leftarrow \mathcal{P}$
    $\mathcal{S} \leftarrow$ CreateNode($\mathcal{S}, \mathcal{P}$)
    return $\mathcal{S}$
  end if
end function

Algorithm 3 Deplump Continued

function UPDATECOUNTSANDDISCOUNTS($\mathcal{N}, s, p$, BackOff)
  $d \leftarrow$ GetDiscount($\mathcal{N}$)
  if $c > 0$ **then**
    $pp \leftarrow (p - \frac{c_i - t_i d}{c}))(\frac{c}{td})$
    $tw \leftarrow c_i + d(t * pp - t_i)$
  else
    $pp \leftarrow p$
  end if
  if BackOff and $c > 0$ **then**
    $c_s^{\mathcal{N}} \leftarrow c_s^{\mathcal{N}} + 1$
    BackOff $\leftarrow 0$
    BackOff $\leftarrow 1$ w.p. $pp(\frac{t^{\mathcal{N}} d}{tw})$ ▷ w.p abbreviates "with probability"
    if BackOff **then**
      $t_s \leftarrow t_s + 1$
    end if
  else
    if BackOff **then**
      $c_s \leftarrow c_s + 1$
      $t_s \leftarrow t_s + 1$
    end if
  end if
  UpdateDiscountParameterGradients(t_s, t, pp, d)
  UpdateCountsAndDiscounts(PA($\mathcal{N}$), s, pp, BackOff)
  ThinCounts($\mathcal{N}$,k)
end function
function THINCOUNTS($\mathcal{N}, k$)
  $d \leftarrow$ GetDiscount($\mathcal{N}$)
  while $\sum_{s \in \Sigma} c_s > k$ **do**
    $s \leftarrow$ DrawMultinomial($\pi(\Sigma)$) s.t. $\pi_l = \frac{c_l}{c}$
    $\phi \leftarrow$ SamplePartition(c_s, t_s, d)
    $i \leftarrow$ DrawMultinomial($[\frac{\phi_0}{c_s}, \frac{\phi_1}{c_s}, \ldots, \frac{\phi_{t_s-1}}{c_s}]$)
    if $\phi_i == 1$ **then**
      $t_s \leftarrow t_s - 1$
    end if
    $c_s \leftarrow c_s - 1$
  end while
end function

Algorithm 4 Deplump Continued

function PMF($\mathcal{N}, \pi, m$)
  $d \leftarrow$ GetDiscount($\mathcal{N}$)
  if $c > 0$ **then**
    for $i \in \Sigma$ **do**
      $\pi_i \leftarrow \pi_i + m(\frac{c_i - t_i d}{c})$
    end for
  end if
  if PA($\mathcal{N}$) $\neq$ null **then**
    return PMF(PA($\mathcal{N}$), π, dm)
  else
    $\pi \leftarrow (1 - dm)\pi + dm\mathcal{U}(\Sigma)$ ▷ $\mathcal{U}(\Sigma)$ is the uniform distribution over Σ
    return π
  end if
end function
function FRAGMENTNODE($\mathcal{M}, \mathcal{S}$)
  $d^{\mathcal{M}} \leftarrow$ GetDiscount($\mathcal{M}$)
  $\mathcal{P} \leftarrow$ maximum overlapping suffix of $\mathcal{M}$ and $\mathcal{S}$
  $\mathcal{P} \leftarrow$ CreateNode($\mathcal{P}$, PA($\mathcal{M}$))
  $d^{\mathcal{P}} \leftarrow$ GetDiscount($\mathcal{P}$)
  for $s \in \Sigma$ **do**
    $\phi \leftarrow$ SamplePartition $(c_s^{\mathcal{M}}, t_s^{\mathcal{M}}, d^{\mathcal{M}})$
    $c_s^{\mathcal{P}} \leftarrow 0$
    $t_s^{\mathcal{P}} \leftarrow t_s^{\mathcal{M}}$
    $t_s^{\mathcal{M}} \leftarrow 0$
    for $i = 1 : |\phi|$ **do**
      $a \leftarrow$ DrawCRP($\phi[i], d^{\mathcal{M}}/d^{\mathcal{P}}, -d^{\mathcal{M}}$)
      $t_s^{\mathcal{M}} \leftarrow t_s^{\mathcal{M}} + a$
      $c_s^{\mathcal{P}} \leftarrow c_s^{\mathcal{P}} + a$
    end for
  end for
  return $\mathcal{P}$
end function

Algorithm 5 Deplump Continued

function SAMPLEPARTITION(c, t, d)
  $M \leftarrow t \times c$ matrix of zeros
  $M(t, c) \leftarrow 1.0$
  for $j = (c-1) : 1$ **do**
    for $i = 1 : (t-1)$ **do**
      $M(i, j) \leftarrow M(i+1, j+1) + M(i, j+1)(j - id)$
    end for
    $M(d, j) \leftarrow M(t, j+1)$
  end for
  $\phi \leftarrow \vec{0}$ ▷ $|\vec{0}| = t$
  $\phi[1] \leftarrow 1$
  $k \leftarrow 1$
  for j = 2 : c **do**
    $M(k, j) \leftarrow M(k, j)(j - 1 - kd)$
    $r \leftarrow 0$
    $r \leftarrow 1$ w.p. $\frac{M(k+1,j)}{M(k+1,j)+M(k,j)}$
    if r = 1 **then**
      $k \leftarrow k + 1$
      $\phi[k] \leftarrow 1$
    else
      $i \leftarrow$ DrawMultinomial($[\frac{\phi[1]-d}{j-1-kd}, \frac{\phi[2]-d}{j-1-kd}, \ldots, \frac{\phi[k]-d}{j-1-kd}]$)
      $\phi[i] \leftarrow \phi[i] + 1$
    end if
  end for
  return ϕ
end function
function DRAWCRP(n, d, c)
  $t \leftarrow 1$
  for i = 2 : n **do**
    $r \leftarrow 0$
    $r \leftarrow 1$ w.p. $\frac{td+c}{i-1+c}$
    $t \leftarrow t + r$
  end for
  return t
end function
function UPDATEDISCOUNTPARAMETERGRADIENTS($p\_depth, depth, t_s, c, t, pp, d, m$)
  if $c > 0$ **then**
    if $depth = 0$ **then**
      derivLogDd $\leftarrow \frac{1.0}{\mathcal{D}[0]}$
      $\mathcal{G}[0] \leftarrow \mathcal{G}[0] + (d(t * pp - t_s)$derivLogDd$/c)m$
    else
      $z \leftarrow p\_depth + 1$
      while $z \leq depth$ and $z < 10$ **do**
        derivLogDd $\leftarrow \frac{1.0}{\mathcal{D}[z]}$
        $\mathcal{G}[z] \leftarrow \mathcal{G}[z] + (d(t * pp - t_s)$derivLogDd$/c)m$
      end while
      if $depth \geq 10$ **then**
        $a \leftarrow z - 10$
        $b \leftarrow depth - z + 1$