2007－AL－110 (3)
2007／1／23

ベイズ符号化アルゴリズムを用いたテキストデータ圧縮

中野 晶† 小泉 大城† 松嶋 敏泰†

†早稲田大学理工学部経営システム工学科

ベイズ符号は，情報源の確率分布のクラスは既知であり，そのパラメータは未知である場合のユニバーサル情報源符号化法で，ベイズ基準の下で冗長度を最小にする符号である．Context Tree 情報源に対してベイズ符号を実現する手法として，逐次型ベイズ符号化アルゴリズムがあり，その必要メモリ量を削減したアルゴリズムとして改良型ベイズ符号化アルゴリズムがある．これらを用いてテキストデータを圧縮した際，モデルやパラメータの事前分布の不適合性より既存の圧縮ソフトウェアの bzip2 などに圧縮性能で劣ることや，圧縮時の必要メモリ量が莫大なことなどから実用化には至っていない．本研究では，ベイズ符号の実用化に向けて，必要メモリ量に制約をおいた上で，テキストデータに適合する事前分布を導入することにより圧縮性能を向上させることを目的とする．

Text Data Compression by Bayes Coding Algorithm

Akira Nakano† Daiki Koizumi† Toshiyasu Matsushima†

†Department of Industrial Management System Engineering,
School of Science and Engineering, Waseda University

Bayes code is one of universal source coding algorithms, such that a class of the probabilistic model of source is known but its parameter is unknown. Bayes code provides Bayes optimality in terms of the redundancy. Bayes coding algorithm for the context tree sources has been proposed, and modified version of this algorithm to reduce the required memory has also been proposed. When text data is compressed by these algorithms, however, there are two problems. One is that the compression ratio is worse than conventional data compression algorithm such as bzip2, because the prior distribution of model or parameter does not fit text data. The other is that it needs enormous memory under the compression. This paper tries to improve Bayes coding algorithm in term of the compression ratio by using the other prior distribution, as well as constraint the limit of memory requirement, toward the implementation of text compression by Bayes coding algorithm.

1 はじめに

情報源の確率構造が完全に与えられていない場合の情報源符号化法，すなわちユニバーサル符号については従来からさまざまな研究が行われている．本研究では特に情報源の確率分布のクラスのみ仮定し，そのパラメータが未知である場合のユニバーサル符号について扱う．その中で，ベイズ符号はベイズ基準の下で冗長度（平均符号長とエントロピーの差）を最小とする符号である [1]．本研究では，情報源のクラスとして Context Tree 情報源（CT 情報源）を仮定した場合のベイズ符号を扱う．

CT 情報源とは，ある時点におけるシンボルの生起確率が過去の系列によって決まる情報源である．シンボルの生起確率が直前の i シンボルによって決まるとき，直前の i 系列を状態といい，i を CT 情報源の次数という．最大次数が未知である CT 情報源に対するベイズ符号の具体的なアルゴリズムとしては，Willems らによる拡張 Context Tree Weighting（CTW）法 [2] や松嶋らによる逐次型ベイズ符号化

アルゴリズム [3] などがあり，系列を木構造（文脈木と呼ぶ）で表現することで効率よく系列の生起確率の推定を行うことができる．系列長 n のデータ系列を符号化する際に必要なメモリ量は，拡張 CTW 法で $O(n)$，逐次型ベイズ符号化アルゴリズムでは $O(n^2)$ であり，必要な計算量はともに $O(n^2)$ である．筆者らはベイズ符号化アルゴリズムを改良し，その計算量は同等で，必要メモリ量が $O(n)$ となる改良アルゴリズムを提案した [4]．しかし，どのアルゴリズムもデータ系列を読み込むごとに文脈木を成長させながら符号化確率の計算を行っており，データ系列長の増加に従って文脈木を保持するための必要メモリ量が増加するアルゴリズムであるため，圧縮対象ファイルよってはメモリオーバーフローを起こしてしまう恐れがある．よって，実用上は必要メモリ量に何らかの制約を加える必要がある．

また，これらのアルゴリズムでは CT 情報源モデルやそのパラメータの事前分布を定める必要がある．従来は，パラメータの事前分布として Jeffrey's prior [5]，モデルの事前分布としてモデルにおける状態の次数が大きいほどその生起確率が小さくなるような離散分布をおいてきたが，これを用いてテキストデータを圧縮した場合に既存の圧縮ソフトウェアである bzip2 などと比較して圧縮性能が劣るという問題がある．これは，先述した事前分布がテキストデータに適合していないからである．

本研究はベイズ符号化アルゴリズムの実用化に向け，アルゴリズムで生成される文脈木や計算に必要なメモリ量の上限値に制約をかけた下で，テキストデータに適合した事前分布を導入することにより，圧縮性能や必要メモリ量の観点からテキストデータに対するベイズ符号の性能を向上させることを目的とする．

本論文では，第 2 章で本研究の問題設定を行い，CT 情報源に対するベイズ符号の従来手法である改良アルゴリズムと，テキストデータに対するベイズ符号の問題点について述べる．第 3 章ではテキストデータ圧縮を効率的に行うためのモデル，パラメータの事前分布の設定を行い，これを用いた場合のデータ圧縮アルゴリズムの提案に向けた実験について述べる．第 4 章で設定を行った事前分布を用いた場合のテキストデータ圧縮を行った場合の性能評価を行い．第 5 章で考察を行う．

2 準備

2.1 問題設定

本研究では，以下のノーテーションを用いる．A を有限アルファベットとし，$x_i \in A$ をアルファベット中の任意のシンボルとする．x^n を長さ n の系列 $x_1x_2\cdots x_n$，x_i^j を x^n の部分系列 $x_ix_{i+1}\cdots x_j$ $(1 \leq i \leq j \leq n)$ とする．$m \in M$ を情報源モデル（M は情報源モデルの集合）としたとき，$\theta(m)$ を m のパラメータ（シンボルを出力する確率），$\mathcal{S}(m)$ をモデル m の持つ状態の集合とする．また，$s \in \mathcal{S}(M)$ をある状態としたとき，$\theta(s)$ を s のパラメータとする．

本研究では，情報源モデルのクラスとして CT 情報源を仮定し，系列 x^{t-1} が与えられたときにシンボル x_t の生起確率の推定を行い，これを下に算術符号等のエントロピー符号化を用いて符号語を出力する．このとき，真のモデル m^* とその下での真のパラメータ $\theta^*(m^*)$ は未知であり，モデルの事前分布 $P(m)$ とパラメータの事前分布 $P(\theta(m))$ は既知であるとする．

2.2 CT 情報源

CT 情報源とは，過去の有限系列 x^{t-1} によって現在のシンボルの生起確率が決まる情報源である．CT 情報源は階層モデルであり，モデルとその下でのパラメータの 2 つにより定まる．時点 t における情報源の状態 $s(x^{t-1}) \in \mathcal{S}(m)$ は x^{t-1} から一意に定まる．$s(x_{t-i}^{t-1})$ において状態が一意に定まる最小の i を，その状態の次数という．各状態 $s \in \mathcal{S}(m)$ にはシンボルの生起確率 $\theta(s)$ が定められており，これによりシンボル x_t の生起確率 $P(x_t|x^{t-1}, \theta(m), m)$ が決まる．

CT 情報源はマルコフ過程の一種で，状態遷移図で表すことができる．CT 情報源はまた，完全木（文脈木とよぶ）を用いて表すことができ，このとき，葉が各状態 s_k，枝がシンボル $a \in A$ を表しており，文脈木の根から過去の系列 $x_{t-1}, x_{t-2}, \cdots$ へ順に枝をたどることで時点 t の状態がわかる．

CT 情報源から出力された系列 x^n の生起確率 $P(x^n)$ は，以下のようになる．

$$P(x^n) = P(x_1|\lambda)P(x_2|s(x^1))$$

$$\cdots P(x_n|s(x^{n-1})). \quad (1)$$

ここで，λ は空系列を表す．

2.3 逐次型ベイズ符号 [1]

ベイズ符号は，ベイズ基準の下で冗長度が最小となるように系列 x^n の生起確率の推定を行い，ブロック型と逐次型の二つに分けられる [1]．逐次型ベイズ符号ではすべての時点 $t(t=1,2,\cdots,n)$ において x^{t-1} が与えられた下でのシンボル x_t の生起確率 $P_C(x_t|x^{t-1})$（以下，逐次型ベイズ符号化確率と呼ぶ）を推定し，モデルとパラメータの事後分布 $P(m|x^t), P(\theta(m)|x^t,m)$ を出力する．次の時点 $(t+1)$ では，出力された $P(m|x^t), P(\theta(m)|x^t,m)$ を事前分布として用いて $P_C(x_{t+1}|x^t)$ の計算を行う．逐次型ベイズ符号化確率は，次式で計算することができる [1]．

$$\begin{aligned} &P_C(x_t|x^{t-1}) \\ &= \sum_{m\in M}\int_{\theta(m)} P(x_t|x^{t-1},\theta(m),m) \\ &\quad \cdot P(\theta(m)|x^{t-1},m)P(m|x^{t-1})d\theta(m). \end{aligned} \quad (2)$$

逐次型ベイズ符号化アルゴリズム [3] や改良アルゴリズム [4] は，有限アルファベット上の CT 情報源を対象としている．

2.4 改良ベイズ符号化アルゴリズム [4]

2.4.1 概要

モデルの事前分布 $P(m)$ に対して，状態の事前確率 $P(s)$ を次式で定義する．

$$P(s) = \sum_{m:s\in S(m)} P(m). \quad (3)$$

$P(s)$ を用いることで，逐次型ベイズ符号化確率は次式で計算できる．

$$\begin{aligned} &P_C(x_t|x^{t-1}) \\ &= \sum_{s\in S(M)}\int_{\theta(s)} P(x_t|x^{t-1},\theta(s),s) \\ &\quad \cdot P(\theta(s)|x^{t-1},s)P(s|x^{t-1})d\theta(s). \end{aligned} \quad (4)$$

ここで，状態 s' に対して s が次数の小さい状態であり，$s'=as$（a は任意の系列）であることを $s<s'$ と表す．状態 $s_{i1}, s_{i2}, \cdots, s_{ij}$，$(s_{i1}<s_{i2}<\cdots<s_{ij}, j\geq 1)$ におけるシンボルの出現頻度が全ての状態で等価であるとき，これらの状態をひとつの状態集合 S_i として表現する．そのような状態集合で表現された文脈木を用いた上で，CT 情報源に対して逐次型ベイズ符号化確率を計算するアルゴリズムである．圧縮を行う際の必要メモリ量は，文脈木の状態数に依存するので，複数の状態を一つの状態とみなすことで必要メモリ量の削減となる．このアルゴリズムでは，各状態集合 S に対して，S の次数 $d(S)$ と親側の枝に対応する系列 $y(S)$ とその系列長 $l(S)$ を保持させる．このアルゴリズムでは，各時点 t において，

1) 符号化確率の計算を行う．

2) 時点 $(t-1)$ で生成した文脈木から，時点 t の過去系列 x^{t-1} に対応した状態を追加する．

という 2 段階で構成される．

2.4.2 改良ベイズ符号化確率計算式

各状態に対して $q(s)$ を次式で定義する．

$$q(s) = \frac{P(s)}{\sum_{s\leq s'} P(s')}. \quad (5)$$

$q(s)=r$ となるように $P(s)$ の事前分布を設定すると，逐次型ベイズ符号化確率は次式で計算できる．

$$P_C(x_t|x^{t-1}) = q(x_t|x^{t-1}, S_\lambda), \quad (6)$$

$$\begin{aligned} &q(x_t|x^{t-1},S) \\ &= \begin{cases} P^S(x_t|x^{t-1},S) \quad (S:leaf), \\ \dfrac{(1-r^{l(S)})P^S(x_t|x^{t-1},S)}{(1-r^{l(S)})+Q(S|x^{t-1})} \\ \quad + \dfrac{q(x_t|x^{t-1},S_c(t))Q(S|x^{t-1})}{(1-r^{l(S)})+Q(S|x^{t-1})} \\ \qquad (S:elsewhere). \end{cases} \end{aligned} \quad (7)$$

ただし，$n(i|x^{t-1},s)$ を状態 S の下でのシンボル i の出現頻度，α_i をシンボル i のハイパーパラメータとし，$Q(S|x^{t-1}), P^S(x_t|x^{t-1},S)$ は次式で与えられる値とする．

$$\begin{aligned} &Q(S|x^{t-1}) \\ &= r^{l(S)} \prod_{\{\tau<t:S\in S_t\}} \frac{q(x_\tau|x^{\tau-1},S_c(\tau))}{P^S(x_\tau|x^{\tau-1},S)}. \end{aligned} \quad (8)$$

$$P^S(x_t|x^{t-1},S) = \frac{n(x_t|x^{t-1},s)+\alpha_{x_t}}{\sum_i (n(i|x^{t-1},S)+\alpha_i)}. \quad (9)$$

$Q(S|x^t)$ は，時点ごとに次式で更新される．

$$Q(S|x^t) = \begin{cases} Q(S|x^{t-1}) & (S \notin S_t), \\ Q(S|x^{t-1})\dfrac{q(x_t|x^{t-1},S_c(t))}{P^S(x_t|x^{t-1},S)} & (S \in S_t). \end{cases} \quad (10)$$

2.4.3 文脈木の更新

改良アルゴリズムでは，以下のように文脈木を生成していく．

Step1)
$i=0$，$S=S_\lambda$ として，
1-1) $x_{t-i-l(S)+1}^{t-i}=y(S)$ なら **1-2)** へ，
それ以外なら **Step2)** へ．
1-2) $i=i+l(S)$，$S=S'$ として **1-1)** へ．
ただし，S' は S の $x_{t-i-l(S)}$ 側の子ノード．

Step2)
$y(S)$ の i 番目から j 番目までの部分系列を y_i^j とし，$x_{t-i-k+1}^{t-i}=y_{i-k+1}^{i}$ となる最大の k を $\tilde{k}$ とする．

Step3)
$\tilde{k}>0$ なら，S' とその親ノードの間にノード $\tilde{S}$ を追加，$l(\tilde{S})=\tilde{k}$，$h(\tilde{S})=h(S')$ とし，$h(S')=h(S')-\tilde{k}$，$l(S')=l(S')-\tilde{k}$ とする．

Step4)
$Q(S'|x^t)$，$Q(\tilde{S}|x^t)$ の計算を行う．$Q(S'|x^t)$ は分解前の $Q(S'|x^t)$ を用いて，

$$Q(S'|x^t)=r^{-k}Q(S'|x^t), \quad (11)$$

ただし，$k=l(\tilde{S})$．$Q(\tilde{S}|x^{t'-1})$ は次式で計算する．

$$Q(\tilde{S}|x^t)=r^k((1-r^{l(S')-k})+Q(S'|x^t)). \quad (12)$$

Step5)
$\tilde{S}$ から $x_{t-h(S')-\tilde{k}}$ 側に子ノード S_{new} を作る．
($\tilde{k}=0$ なら $\tilde{S}$ を S' の親ノードとする)

Step6)
$t=t+1$ として，逐次型ベイズ符号化確率の計算 (6) 式へ．

2.4.4 逐次型ベイズ符号化アルゴリズムを用いたテキストデータ圧縮における問題点

テキストデータに対してこのアルゴリズムを用いて圧縮を行った場合，2つの問題点がある．ひとつは必要メモリ量が系列長に依存して大きくなる点である．実用上は，使用メモリ量に何らかの制約をかけないと系列長によってはメモリオーバーフローを起こしてしまう可能性がある．もうひとつは既存の圧縮ソフトウェアである bzip2 などと比較した際の圧縮率である．従来，パラメータの事前分布には局所一様事前分布となる Jeffrey's prior，モデルの事前分布には $q(s)=0.5$ という事前分布を利用している．これらの事前分布は，ベイズ統計学においては無情報事前分布と呼ばれ，圧縮対象データに関する事前情報が得られない場合は，このような事前分布を利用するのが妥当であるといわれている．しかし，圧縮対象ファイルに対して何らかの事前情報が得られた場合には，無情報事前分布は必ずしもその適合していない．実際に英文テキストデータを，無情報事前分布を用いて符号化した場合に，bzip2 と比較して圧縮性能がよいとはいえない．

3 実験目的

先に述べたベイズ符号化アルゴリズムの圧縮率の問題点に関して，あらかじめ符号化対象データに関する事前情報が得られた場合，それらの情報を考慮に入れた事前分布を用いることで，圧縮性能を向上させることができると考えられる．そこで本章では，テキストデータ圧縮に特化した逐次型ベイズ符号化アルゴリズムの提案に向けた実験を行う．ベイズ符号では圧縮を行った際に，入力したデータ x^n と事前分布 $P(m)$，$P(\theta(m)|m)$ に対して事後分布 $P(m|x^n)$，$P(\theta(m)|m,x^n)$ を得られる．事後分布には，圧縮を行ったファイルの状態ごとの文字の出現頻度や，モデルのベイズ推定値などの情報が含まれている．そこで，あらかじめ用意した学習データを Jeffrey's prior および $q(s)=0.5$ を事前分布に用いた逐次型ベイズ符号による圧縮を行い，出力された事後分布を用いて文脈木モデルやモデルの事前分布，パラメータの事前分布の設定を行う．また，これらを用いてテキストデータ圧縮における逐次型ベイズ符号化アルゴリズムについての考察を行う．

3.1 モデルとその事前分布，頻度カウンタの初期値の設定

学習データ1ファイルごとに改良ベイズ符号化による圧縮を行い，頻度カウンタの事後値 $n(i|x^n, S)$ と $q(S)$ の事後値 $q(S|x^n)$ の出力を行う．ただし，$q(S|x^n)$ は次式で求める．

$$q(S|x^n) = \frac{1 - r^{l(S)}}{(1 - r^{l(S)}) + Q(S|x^n)}. \qquad (13)$$

系列長 n が十分に大きい場合，$q(S|x^n) \approx 1$ になる S がある．このような分布をモデルの事前分布として圧縮を行ったとき，$S' < S$ となる S' に関しては，(7) 式の計算を省略しても符号化確率に微小な差しか生じない．そこで，学習ファイルごとに S' を削除した文脈木モデルを生成し，ファイルクラスの全ての学習ファイルの文脈木モデルを組み合わせた混合文脈木モデルを作る．混合文脈木モデルのノード S の事前確率 $P(S)$ が，文脈木モデルを生成した際に S が葉ノードとなる学習ファイルの割合，となるように $Q(S)$ もしくは $q(S)$ の設定を行う．また，S の頻度カウンタの事前値に関しては，S における全学習ファイルの頻度カウンタの事後値の合計値とし，次節でこれを調整することで事前分布の設定を行う．

3.2 事前分布の調整

3.1 節で設定したモデルの事前分布や頻度カウンタの値の調整を行い，モデルの事前分布の決定を行う．具体的には，テキストファイルのクラスごと（分布3は全クラスあわせて一つ）に以下のような事前分布を準備する．

事前分布 1 3.1 節で設定した事前分布，頻度カウンタをそのまま用いる．

事前分布 2 事前分布はそのまま，頻度カウンタは状態ごとに，カウンタ値の割合を変えずに合計値が定数 c となるように正規化を行ったものを用いる．

事前分布 3 3.1 節で設定した事前分布，頻度カウンタをそれぞれ重み付けを行った事前分布，カウンタ値を用いる．

3.3 テキストデータに対する逐次型ベイズ符号化アルゴリズム

テキストデータに対して逐次型ベイズ符号化アルゴリズムを用いる際，以下の手順で符号化確率の計算を行う．

1) 文脈木モデルの生成．

2) モデルの事前分布と頻度カウンタの初期値として事前分布 1，2 または 3 を与える．

3) 文脈木を固定した下で，(6) 式により各時点の符号化確率の計算を行う．

4 事前分布の性能評価

本章では，前章で設定した事前分布 1〜3 を用いたベイズ符号化によるテキスト圧縮の性能評価を行う．本論文では簡単のため，テキストデータのファイルクラスを

英文書式なしテキスト（$|A| = 128$）
英文 HTML 文書（$|A| = 128$）
日本語書式なしテキスト（$|A| = 256$）
日本語 HTML（$|A| = 256$）

の4つと仮定して事前分布1，2および3の設定を行い，その圧縮率および必要メモリ量の性能をシミュレーション実験を用いて評価する．

4.1 実験条件

学習データについて逐次型ベイズ符号化を行い，3.1 節の手順に従ってファイルクラスごとにモデルの事前分布，カウンタの事前値の設定を行う．これを下に，事前分布 1，2 および 3 の設定を行う．なお，事前分布 2 の c の値は，$10, 20, 50, 100$ とし，事前分布 3 の重み付けの比率は，4 つのテキストファイルのクラスで全て等重みとする．比較対象として，従来法として Jeffrey's prior と $q(s) = 0.5$ を事前分布として用いた逐次型ベイズ符号，既存のデータ圧縮ソフトウェアとして bzip2 を用いた場合を比較する．なお，学習データ，圧縮対象データは，[6]-[8] の引用元からそれぞれ 100 ファイル，20 ファイルのランダムサンプリングを行った．

表 1: 各ファイルクラスの圧縮結果（圧縮率の平均値）

使用事前分布		英文テキスト	英文HTML	日本語テキスト	日本語HTML
Jeffrey's prior		0.3391	0.3163	0.4829	0.4219
事前分布1		0.2634	0.2447	0.3507	0.2479
事前分布2	c＝10	0.2518	0.2277	0.3388	0.2419
	c＝20	0.2500	0.2266	0.3403	0.2415
	c＝50	0.2506	0.2281	0.3451	0.2445
	c＝100	0.2525	0.2308	0.3499	0.2482
事前分布3		0.2611	0.2433	0.3706	0.2651
bzip2		0.3199	0.2959	0.4947	0.3856

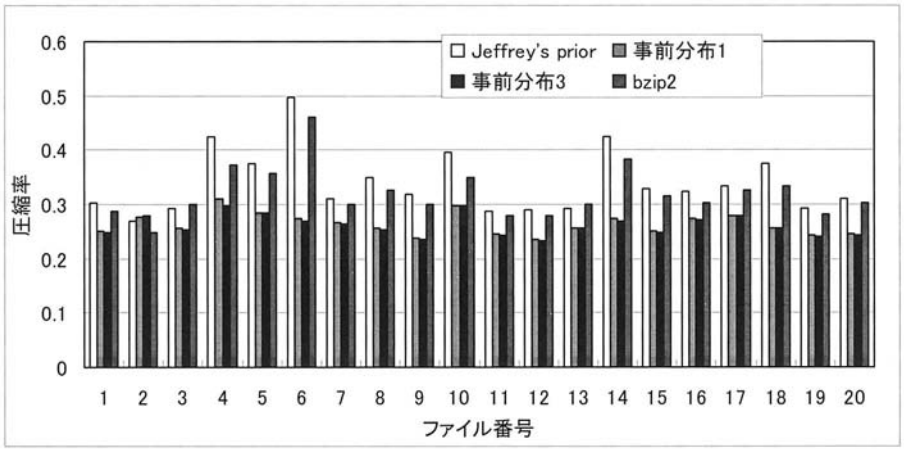

図 1: 英文テキストの圧縮結果

・学習データの合計ファイルサイズ
　英文テキスト [6]　… 計約 40MB
　英文 HTML[6]　… 計約 35MB
　日本語テキスト [7] … 約 5MB
　日本語 HTML[8]　… 約 3.5MB

・圧縮対象データの合計ファイルサイズ
　英文テキスト [6]　… 計約 6MB
　英文 HTML[6]　… 計約 5.5MB
　日本語テキスト [7] … 約 1.2MB
　日本語 HTML[8]　… 約 0.6MB

4.2 圧縮率

各圧縮対象ファイルを圧縮した際の，ファイルクラスごとの圧縮率の平均値を表 1 に示す．また，事前分布 1 および事前分布 3 を用いた場合の英文テキストデータに対するファイルごとの圧縮率を図 1 に示す．なお，その他のファイルクラスに関する結果も英文テキストデータの場合とほぼ同様である．

4.3 必要メモリ量

次に，必要メモリ量の評価を行う．事前分布 1〜3 を用いて 3.3 節で述べたアルゴリズムによる圧縮を行うのに必要メモリ量は，文脈木を保持するためのメモリ量に上限を設けたため学習データにのみ依存し，圧縮対象データのファイルサイズには依存しない．これに対して Jeffrey's prior を用いた改良アルゴリズムでは，文脈木を保持するメモリ量に制約がないため，必要メモリ量は圧縮対象データのファイルサイズに依存する．なお，実験におけるアルゴ

リズムで圧縮を行うのに必要なメモリ量は，事前分布1および事前分布2を用いた場合は以下の通りとなる．

英文テキスト：約200MB
英文HTML：約200MB
日本語テキスト：約600MB
日本語HTML：約300MB

これは，改良アルゴリズムで200〜300KB程度のファイルの圧縮を行う際に必要なメモリ量に相当する．また，事前分布3を用いた場合は1.5GB程度のメモリ量を必要とした．

5 考察

実験結果から事前分布1〜3を用いたベイズ符号の圧縮率は，ほぼ全てのファイルで，Jeffrey's priorを用いたベイズ符号やbzip2を用いた場合よりも改善が見られ，ファイルクラスごとに平均を取ると20〜35%ファイルによっては50%以上の改善が見られた．これは，テキストデータのモデルやパラメータの分布に，学習データを用いて特徴を反映させた事前分布が適合しているためである．特に，事前分布2で$c = 10, 20, 50$の場合には，実験を行った全てのファイルでbzip2やJeffrey's priorを用いたベイズ符号化よりも効率のよい圧縮ができた．このcの値が大きすぎると圧縮を行う際に事前分布が与える影響が強く，ファイル個別のモデルに適合するのに時間がかかり，またcの値が小さすぎると無情報事前分布に近い事前分布となってしまうためであると考えられる．また，重み付けを行った事前分布3に関しては，圧縮率の観点からは事前分布1や2と比較して大差はないが，必要メモリ量の観点からみると事前分布3を用いた場合は莫大なメモリ量が必要となり，テキストのクラスを増やした場合にメモリオーバーフローの恐れがある．また，事前分布1や2を用いたとしても，文脈木の$P(s)$の小さい状態を削減する等の手法により，実用化に向けてさらに必要メモリ量を削減する必要がある．

6 まとめ

本研究では，逐次型ベイズ符号化アルゴリズムを用いたテキストデータ圧縮に適した事前分布の設定を行い，これらを用いた際のテキストデータ圧縮に関する考察を行った．その結果，必要メモリ量の最大値を制約した下で，圧縮性能の改善が見られ，bzip2と比較しても良好な結果となった．しかし，必要メモリ量は依然としてbzip2と比べて非常に大きい．今後の課題として，必要メモリ量により厳しい制約を加えた下での圧縮アルゴリズムの提案等が挙げられる．

謝辞

本研究を行うにあたり，数多くのご助言，ご支援を頂いた早稲田大学松嶋研究室内各氏に心より感謝申し上げます．なお，本研究の一部は日本学術振興会科学研究費基盤(C)一般（No.18560391）の援助による．

参考文献

[1] T. Matsushima, H. Inazumi and S. Hirasawa, "A Class of Distortionless Codes Designed by Bayes Decision Theory," *IEEE Trans. Inform. Theory*, Vol. 37, No. 5, pp. 1293–1298, 1991.

[2] F. M. J. Willems, "Context-Tree Weighting Method: Extentions," *IEEE Trans. Inform. Theory*, Vol. 44, No. 2, pp. 792–798, 1998.

[3] T. Matsushima and S. Hirasawa, "A Bayes Coding Algorithm for Markov Models," *IEICE Technical Report*, Vol. IT-95, pp. 1–6, 1995.

[4] 中野晶，小林直人，松嶋敏泰，"FSMX情報源に対するベイズ符号のメモリ量削減アルゴリズム，"電子情報通信学会技術研究報告，Vol. IT-2005-50，pp. 47–52，2005.

[5] B. S. Clarke, "Jeffrey's Prior is Asymptotically Least Favorable under Entropy Risk," *J. of Stat. Planning and Influence*, Vol. 41, pp. 37–60, 1994.

[6] Project Gutenberg, http://www.gutenberg.org/wiki/Main_Page

[7] Japanese Text Initiative, http://etext.lib.virginia.edu/japanese/index.euc.html, University Virginia Library.

[8] 青空文庫, http://www.aozora.gr.jp/