大規模日本語ブログコーパスにおける言語モデルの構築と評価

奥野 陽　颯々野 学
ヤフー株式会社
{yookuno, msassano}@yahoo-corp.jp

1　はじめに

自然言語処理の分野において，確率的言語モデル（以後，言語モデル）はコーパスに基づく統計的手法の一つとして90年代から盛んに研究されてきた [1]. 言語モデルは音声認識や機械翻訳に応用されており，言語の自然さをモデル化するために有用である．また，日本語処理に関しては形態素解析や仮名漢字変換において大きな役割を果たしている [2].

近年，Webやブログの普及により大規模な日本語コーパスを手に入れることができるようになってきた. 言語モデルをはじめとする統計的手法では，コーパスが大規模であるほどパラメータ推定の信頼性が上がったり，自由度の高いモデルを利用できたりするため，大規模コーパスの恩恵は大きい.

しかしながら，そのような大規模コーパスによる言語モデルには以下に挙げる2つの問題点がある.

構築時の問題点 大規模なコーパスからN-gramを集計する処理には，莫大な計算とメモリを必要とする．また，そもそも大規模コーパスを1台のコンピュータに保存することは難しい.

利用時の問題点 言語モデルを実際に利用するには，検索などの必要な操作をリアルタイムに行える必要がある．そのためには，言語モデルをメモリ上に読み込めることが望ましいが，大規模な言語モデルでは現在のコンピュータのメモリ搭載量を上回ることが多い.

これらの問題はデータ量と性能とのトレードオフであり，完全に解決する方法は見つかっていない．そのため，言語モデルの応用を考える上でそのトレードオフについて知ることは重要である.

また，大規模なコーパスを扱える環境は現在では限られており，特に日本語のコーパスにおける研究は非常に少ない．そのため，大規模な日本語コーパスで言語モデルの振る舞いを定量的に確認することには意義がある.

そこで，我々は大規模な日本語コーパスの1つとしてブログコーパスを選択し，言語モデルの構築を行った．ブログコーパスを選択した理由は，入手や扱いが容易であることと，一般の人々によって書かれていることの2つがある．後者の性質によって，一般の人々が使う仮名漢字変換や音声認識などの入力システムにおいてはWeb全体を使うよりも有利に働く可能性がある.

本論文では，ブログをテストコーパスとするクロスエントロピーによって言語モデルの評価を行う．また，大規模な言語モデルを応用する上で重要なモデルサイズと性能との関係について調査する.

2　関連研究

言語モデルにはいろいろな種類があるが，本論文では基本的な単語N-gramモデル [3] を扱う．大規模な言語モデルの構築に関する研究としては，低頻度語の切り捨てについての検討 [4] や，MapReduceを用いて言語モデルを構築する手法 [5]，ストリーミングによる集計 [6] などがある．大規模な言語モデルの利用に関しては，簡潔データ構造の一種であるLOUDSの利用 [7] や，N-gramの分散検索を提案した研究 [8] がある．本論文では議論を単純化するため，特別な圧縮などは用いずシンプルな方法で言語モデルを利用する場合のデータ量や性能の振る舞いについて報告する.

3　言語モデル

3.1　単語N-gramモデル

言語モデルの役割は，与えられた単語列 $w_1^n = w_1, ...w_n$ に対し，その生成確率 $P(w_1^n)$ を計算することである.

単語 N-gram モデルは単語列に $N-1$ 次のマルコフ性を仮定し，以下のようにモデル化する [1].

$$P(w_1^n) = \prod_{i=1}^{n} P(w_i|w_1^{i-1}) = \prod_{i=1}^{n} P(w_i|w_{i-N+1}^{i-1}) \quad (1)$$

十分なデータが利用可能ならば，$P(w_i|w_{i-N+1}^{i-1})$ を学習データ中の単語列の相対頻度によって推定することができる．

$$P(w_i|w_{i-N+1}^{i-1}) = \frac{C(w_{i-N+1}^{i})}{C(w_{i-N+1}^{i-1})} \quad (2)$$

ここで $C(w_i^j)$ は単語列 w_i^j が学習コーパス中に出現した回数（頻度）を表す．式 (2) は，文脈 w_{i-N+1}^{i-1} が与えられたときの単語 w_i の分布が多項分布に従うと仮定した場合の最尤推定に相当する．

しかしながら，N が大きくなるほど指数的に信頼できるパラメータ推定に必要なデータ量は増えていくため，最尤推定のアプローチは現実的ではない．N に対して十分なデータ量がない状態で最尤推定を行うと，コーパス中に出現しない単語列の確率が 0 になってしまう．このような問題はゼロ頻度問題あるいはスパースネス問題と呼ばれている．

3.2 Dirichlet スムージング

ゼロ頻度問題に対処するための単純な方法として，N-gram の分布 $P(w_i|w_{i-N+1}^{i-1})$ の事前分布として Dirichlet 分布を導入し，そのハイパーパラメータが (N-1)-gram 確率に比例すると仮定することで，以下の形式を得る [9].

$$P(w_i|w_{i-N+1}^{i-1}) = \frac{C(w_{i-N+1}^{i}) + \alpha P(w_i|w_{i-N+2}^{i-1})}{C(w_{i-N+1}^{i-1}) + \alpha} \quad (3)$$

式 (3) を Dirichlet スムージングと呼ぶ．(N-1)-gram 確率 $P(w_i|w_{i-N+2}^{i-1})$ を推定するには再帰的に Dirichlet スムージングを適用し，1-gram 確率 $P(w)$ にたどり着いたら最尤推定 $P(w) = \frac{C(w)}{C}$ によって求める．ここで C は全単語数である．

3.3 Absolute ディスカウンティング

簡単のためここから [4] にならって単語列 w_i^j を abc と書く．ここで，a は単語列の中の最も左の単語，b は中間の単語列，c は最も右の単語を表す．b は可変長の単語列であり，空文字列を許す．

コーパス中に出現した頻度をそのまま使うと，低頻度の単語が不当に高確率になりやすいという問題が生じる．このため観測された頻度から定数 D を差し引き，修正された頻度を使う方法が Absolute ディスカウンティングである．

$$P(c|ab) = \frac{\max(0, C(abc) - D) + DN(ab*)P(c|b)}{C(ab*)} \quad (4)$$

ここで $N(ab*)$ はコーパス中で単語列 ab の後ろに続く単語の種類数である．

3.4 Kneser-Ney スムージング

Absolute ディスカウンティングにおいて最も高次の N-gram はそのままに，低次の N-gram をより滑らかにするために単語列の異なり数を用いたモデルが Kneser-Ney スムージングである [10].

$$P(c|ab) = \frac{\max(0, N(*bc) - D) + DR(*b*)P(c|b)}{N(*b*)} \quad (5)$$

ここで $R(*b*) = \left|c : N(*bc) > 0\right|$ であり，$*b*$ というパターンに当てはまる N-gram の右側の単語の種類数を表す．

3.5 クロスエントロピー

訓練コーパスから構築した言語モデルの性能を評価するために，テストコーパス w_1^n に対するクロスエントロピーを用いる．

$$H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 P(w_i|w_1^{i-1}) \quad (6)$$

H の単位は bit である．クロスエントロピーの指数 $PP = 2^H$ はパープレキシティと呼ばれる．クロスエントロピーとパープレキシティは，値が小さいほどモデルがテストコーパスによく適合していることを示す．

3.6 MapReduce による N-gram 集計

大規模コーパスから単語 N-gram 言語モデルを推定するには，単語 N-gram の頻度 $C(w_{i-N+1}^{i})$ を集計する必要がある．このために，並列分散処理のための

```
Map(int id, string doc):
    string[] words = MorphologicalAnalyze(doc)
    for i = 1 to size(words)-N+1
        Emit(words[i..i+N-1], 1)

Reduce(string[] words, int[] counts):
    sum = 0
    for each count in counts
        sum += count
    Emit(words, sum)
```

図 1: MapReduce による N-gram 集計

フレームワーク MapReduce[11] を用いて図 1 の疑似コードのように Map 関数と Reduce 関数を用いて集計する [5]．本論文で扱うコーパスは日本語コーパスであるため単語分割には形態素解析を用いた．

大規模コーパスは分散ファイルシステムによって多数のコンピュータに配置され，それぞれのコンピュータ内で Map 関数が実行される．Map 関数が終了すると，同じキーのデータが同じコンピュータに行くよう Shuffle フェーズがフレームワーク側によって自動的に実行され，そのコンピュータの中で Reduce 関数が実行される．本論文ではオープンソースの MapReduce 実装である Hadoop を用いる．

4 実験

4.1 予備実験

予備実験として，ウェブ日本語 N グラム [12] の全データを用いたクロスエントロピーの評価を行った．テストコーパスとして，Wikipedia とブログからそれぞれ 1000 文のテキストをサンプリングし mecab 0.98 で分かち書きした．未知語については特別な種類の 1 つの単語として扱った．パラメータ α と D は 1 から 10000 の間で 10 倍おきに試し最良の値を用いた．実験の結果を表 1 に示す．

予備実験の結果から，以下のことが分かる．

- ウェブ日本語 N グラムは Wikipedia のような硬い文章にはよく適合するが，ブログのようなくだけた文章には必ずしもよく適合するとは限らない．
- スムージング方式は，既存研究と同じく Wikipedia・ブログともに Kneser-Ney が優れている．

表 1: ウェブ日本語 N グラムの実験結果 (bit)

	Wikipedia		Blog	
N	Dirichlet	Kneser-Ney	Dirichlet	Kneser-Ney
1	10.65	10.65	10.77	10.77
2	8.71	8.52	9.63	9.44
3	7.72	5.15	9.21	6.87
4	7.09	5.23	9.35	7.70
5	6.64	5.69	9.43	8.73
6	6.73	6.25	9.48	9.33
7	6.47	6.23	9.49	9.62

4.2 実験設定

本実験に用いたデータは，Yahoo! ブログ検索のクローラが収集したブログの本文テキストである．クロール期間は 2009 年 10 月から 2010 年 10 月までの 1 年間，データサイズは LZO 圧縮状態で合計約 2TB である．

集計に用いた Hadoop クラスタは，スペックが 1CPU/12GB Memory/1TB*4 HDD のサーバ 20 台（マスター 1 台+スレーブ 19 台）で構成されている．形態素解析には Yahoo! 形態素解析 API と同等のライブラリを用いた．

4.3 実験結果

コーパスサイズ（LZO 圧縮状態）と N を変えて集計に必要な時間を測定した結果を表 2 に示す．

表 2: 集計時間（時間:分）

処理	860GB	2TB
形態素解析	9:50	28:16
1-gram	2:14	7:42
2-gram	3:34	13:45
3-gram	5:02	20:43
4-gram	8:58	×
5-gram	11:12	×
6-gram	13:00	×
7-gram	14:48	×

ここで，2TB の 4-gram 以降の「×」はクラスタの性能不足により集計が行えなかったことを示す．

次に，モデルデータのサイズを変えてクロスエントロピーの評価を行った結果を 表 3 に示す．モデルデー

タには 860GB のコーパスから構築した 1〜7-gram を使用し，テストコーパスには学習コーパスと同様の形態素解析を行ったブログテキスト 1000 文を用いた．スムージングは Dirichlet スムージングであり，パラメータの決め方は予備実験と同じである．

モデルデータのサイズを変えるために，閾値 100 から 10000 の間で切り捨てを行い，モデルサイズとクロスエントロピーの関係を調べた．ここでモデルサイズとは各 N の値ごとに N-gram データのテキストファイルとしてのサイズを調べたものである．スムージングを行うためには単純にはすべての低次の N-gram のデータが必要となる．

表 3: クロスエントロピー (bit) とモデルサイズ (byte)

	クロスエントロピー			モデルサイズ		
N	10000	1000	100	10000	1000	100
1	16.25	17.21	17.80	2.8M	9.1M	40M
2	7.71	6.48	7.66	21M	127M	683M
3	8.88	6.41	6.51	30M	293M	2.5G
4	8.93	6.71	6.18	23M	201M	3.6G
5	8.66	6.20	5.97	15M	232M	3.5G
6	8.28	5.98	5.74	8.2M	160M	1.6G
7	7.81	5.68	5.65	5.2M	113M	1.1G

モデルサイズと性能の間にはトレードオフの関係があり，どうバランスを取るかはアプリケーションによって異なる．例えば言語モデルを多くのサーバに分散して格納できる場合は，モデルサイズを気にせず性能を追求することができる．しかし，モバイルや組み込み機器では小規模なモデルしか利用できないだろう．表 3 の結果から，それぞれのユースケースにおけるモデルサイズと性能の目安を知ることができる．

例えば 1 台の PC で言語モデルを利用する場合，現在の PC の性能を考えるとモデルサイズは 1GB 以内程度に収めることが望ましい．表 3 の結果から，現実的なモデルサイズに収めるために閾値 1000 で切り捨てた場合でもモデルサイズは 1.1GB 程度でクロスエントロピーが 5.68bit という性能を得られることがわかる．言語モデルを実際に適用する際はデータの圧縮やアプリケーションに特化した最適化が必要となるが，シンプルな方法で性能を確かめることができた．

5 結論

本論文では，大規模な日本語ブログのコーパスから単語 N-gram 言語モデルを構築し，その評価を報告した．ウェブ日本語 N グラムを用いた実験ではブログコーパスの特殊性を示した．モデルサイズと性能のトレードオフについての実験では，アプリケーションによって取るべきバランスにおけるモデルサイズと性能の目安を明らかにした．

参考文献

[1] 北 研二, 辻井 潤一. 確率的言語モデル. 東京大学出版会, 1999.

[2] 森 信介, 土屋 雅稔, 山地 治, 長尾 真. 確率的モデルによる仮名漢字変換. 情報処理学会論文誌, Vol.40, No.7, pp.2946-2953, 1999.

[3] Stanley Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. TR-10-09, Computer Science Group, Harvard University, 1998.

[4] Deniz Yuret. Smoothing a Tera-word Language Model. ACL-08: HLT, pp.141-144, June 2008.

[5] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean. Large Language Models in Machine Translation. EMNLP-ACL, pp.858-867, June 2007.

[6] Graham Cormode, Marios Hadjieleftheriou. Methods for Finding Frequent Items in Data Streams. VLDB, vol.1 Issue 2, August 2008.

[7] Taro Watanabe, Hajime Tsukada, Hideki Isozaki. A Succinct N-gram Language Model. ACL-IJCNLP, pp.341-344, August 2009.

[8] Ahmad Emami, Kishore Papineni, Jeffrey Sorensen. Large-Scale Distributed Language Model. ICASSP, IV-37-IV-40, April 2007.

[9] David J. C. MacKay, Linda C. Bauman Peto. A hierarchical Dirichlet language model. Natural Language Engineering, vol.1 Issue 03, pp.289-308, 1995.

[10] Kneser R., Ney H.. Improved backing-off for M-gram language modeling. ICASSP, pp.181-184, vol.1, 1995.

[11] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI, December, 2004.

[12] 工藤拓, 賀沢秀人, Web 日本語 N グラム第 1 版, 言語資源協会発行, 2007.