5　文書の統計モデル

これまで, 文字・単語・文の順にテキストの単位と, その統計的なモデル化について考えてきました. 実際に私たちが出会う場面では, 単語だけ, 文だけが問題となることは少なく, 文が集まった**文書**を扱うことが多いでしょう. ここでいう文書とはいわゆる書類だけでなく, 普段触れる Web ページや電子メール, アンケートの自由回答, SNS 上の記事なども含まれます. 小説や論説, 法案といったものも, 時に非常に長くなりますが, 文書の一つといっていいでしょう.

文書としてのテキストの特徴は, **意味的まとまり**を持っているということです [*1]. たとえば, 図 5.1(a) のように完全に無関係な言葉の集まった文書は, まず存在しないでしょう [*2]. 実際の文書は図 5.1(b) のように, 何かのテーマ (ここでは「クラシック音楽」でしょう) に沿った内容が書かれているはずです. これはテキストが, 何かの意図を持って情報を伝える媒体であることから明らかでしょう. それでは, 文章からその「意味的まとまり」をどうやって数学的に取り出したらいいのでしょうか.

5.1　ナイーブベイズ法と単語集合表現

文書の意味的まとまりとして, 最もわかりやすいのは文書の「ジャンル」[*3], または分野を表すカテゴリといったものでしょう. 新聞やニュース記事には「社会面」「家庭欄」といった区別があり, それぞれ特定の言葉を使って, 一定のカテゴリの内容が書かれているのが普通です. 身近なところでは, 毎日届く迷惑メー

∗1　これには本書で扱うもの以外に, 文体 (スタイル) 上のまとまりも含まれます. 一つの文書の中では一般にスタイルが統一されていることを利用して, 単語の埋め込みベクトルを意味を表す成分とスタイルを表す成分に分けて学習する興味深い研究に, 東北大の赤間らによる[131]があります.

∗2　3 章での, 意味を考えない n グラムモデルからの出力を思い出してみましょう.

∗3　言語学では, レジスター (言語使用域) ともよばれています.

ルも一種のカテゴリで, この場合, メールは「通常のメール」と「迷惑メール」の二種類のカテゴリに分けられることになります. また, Amazon のレビューなども付けられた星の数によって, 「肯定的」および「否定的」なものに分けることができるでしょう.

Amazon レビューの 5 億個を超える多言語データセット *4 は公開されており [132], 日本語版だけ抜き出したものも Hugging Face で公開されています *5. 一方で, 新聞記事のようにカテゴリの付与された文書は有料のことが多く *6, 特に日本語で自由に使用できるコーパスは少ないのですが, [133]で公開されている Livedoor ニュースコーパスは, 表 5.1 に示した 9 個のカテゴリの記事からなる 7,367 文書の公開データです. たとえば「家電チャンネル」の記事には “売れ筋”, “加湿” といった言葉が, 「MOVIE ENTER」の記事には “公開”, “ヒーロー” といった言葉が多く出現するため, テキストを見れば, 9 つのカテゴリのどれに属する文書なのかが判定できそうです. なお以下では, 3.1 節で紹介した方法などで, 日本語の文書もあらかじめ単語に分けられていると仮定します.

単語集合表現 こうした文書の意味内容を表す最も簡単な方法は, 文書の中で単語の順番を考えず, 頻度だけを数えることです. というのは, 言語には一般に非常に多くの語彙があるため, どんな単語が何回出現したのかを見れば, 文書の大まかな意味はほぼ特定できるからです *7. 図 5.1 のように, これは文書を単語の

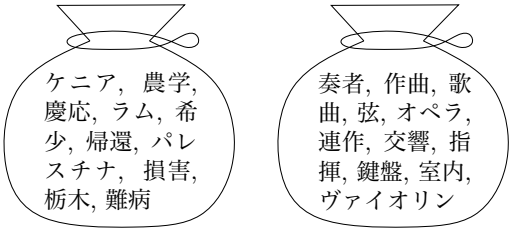
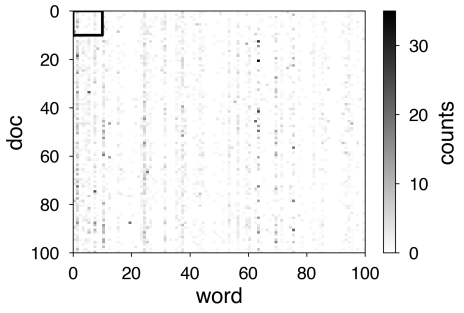


図 5.1: 単語集合表現 (Bag of Words) と意味的まとまりの様子. (a) 単語がまったくランダムに選ばれた文書, および (b) 内容に意味的まとまりがある文書の場合.

*4 https://amazon-reviews-2023.github.io/
*5 https://huggingface.co/datasets/SetFit/amazon_reviews_multi_ja
*6 新聞記事コーパスの入手については, 78 ページを参照してください.
*7 これに対して, たとえば DNA では文字が ATGC の 4 種類しかありませんから, 各文字の出現頻度よりも, その並び方の方がはるかに重要になります.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
d_1	0	8	1	0	1	1	0	2	0	0
d_2	0	2	0	2	0	0	0	4	0	0
d_3	0	2	0	0	0	1	0	3	0	0
d_4	0	6	1	0	0	2	0	1	0	0
d_5	0	10	1	0	0	0	0	2	0	0
d_6	0	3	0	0	0	2	0	4	0	0
d_7	2	10	0	0	2	0	0	1	0	0
d_8	0	2	0	0	0	0	0	3	0	0
d_9	13	1	0	0	0	1	0	0	0	0
d_{10}	0	1	0	0	2	0	0	1	0	0

(a) 文書 1～10, 単語 1～10 の様子.



(b) 文書 1～100, 単語 1～100 の様子.

図 5.2: Livedoor コーパスでの文書–単語行列の一部. 横軸の単語は, コーパス全体での頻度順に並んでいます. この行列のうちほとんどは 0 (99.1%=123,721,815/124,833,815) で, 非常に疎な行列になっています. (b) の黒枠の中を拡大したものが (a) の行列です.

集合 (袋詰め) にしたものと考えることができるため, **単語集合** (Bag of Words) 表現とよばれています. 2 章で学んだ用語を使えば, これは単語が (文書ごとに異なる) ユニグラム分布から生成された, と仮定していることになります. [*8]

この場合, データは図 5.2 のように, 文書番号を縦に, 単語を横にとった**文書–単語行列**で表すことができます. この行列の (d, w) 要素が, d 番目の文書に単語 w が出現した回数 $n(d, w)$ になっています. 図からもわかるように, この行列はほとんどが 0 で, 非常に疎 (スパース) な行列になっています. よって, 実際に表す際にはゼロでない要素だけを使って,

```
1:5 2:7 7:1 12:4 ...
```

のように, 3.4.4 節でも使った SVMlight 形式で表すとよいでしょう. `w:c` のような表記は, 単語 w が c 回出現したことを表しています. つまりこの文書には, 単語 1 が 5 回, 単語 2 が 7 回, 単語 7 が 1 回, …出現したことになります. これは単語集合表現ですから, 順番には特に意味はありません.

*8　「テキストを精密にモデル化する」という観点からは, これはずいぶん簡略化された表現です. 現在の深層学習を用いれば, 次の単語をより正確に予測することは可能ですが, テキストが何の意味を表しているかは完全にブラックボックスで, 分からなくなってしまいます. モデル化とは現実をそのまま再現することではなく, 「ある観点で見た」切り口を考えることで, 現象に関する理解を深め, 制御を可能にすることです[40]. これは無闇に複雑なモデルを使えばよいわけではない, ということの一つの現れといえるでしょう.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ラベル	y
d_1	1		1		2		1	通常メール	0
d_2		1	2		1	1			
d_3	1	1		1		2		迷惑メール	1

図 5.3: ナイーブベイズ法での文書–単語行列の例. 空欄の部分は頻度が 0 であることを表しています.

たとえば, 最も簡単な例として, 文書–単語行列が図 5.3 のようになっているとしましょう. 文書 d_1 と d_2 は通常のメール ($y=0$), 文書 d_3 は迷惑メール ($y=1$) というラベル y がわかっていたとします. 単語 w_1 や w_3 は day や work などの普通の単語, w_4 や w_6 は cash や dollar といった迷惑メールによく現れる単語を表していると考えてください.

このとき, ラベル $y=0$ (通常メール) での単語の確率分布は, 最も簡単には d_1 と d_2 での頻度を合計して総和で割ればよく,

$$p(w|y=0) = \left(\frac{1}{10}, \frac{1}{10}, \frac{1+2}{10}, \frac{0}{10}, \frac{2+1}{10}, \frac{1}{10}, \frac{1}{10}\right) = (0.1, 0.1, 0.3, 0, 0.3, 0.1, 0.1) \tag{5.1}$$

となります. 一方, $y=1$ (迷惑メール) の場合は文書は d_3 の 1 つだけなので,

$$p(w|y=1) = \left(\frac{1}{5}, \frac{1}{5}, \frac{0}{5}, \frac{1}{5}, \frac{0}{5}, \frac{2}{5}, \frac{0}{5}\right) = (0.2, 0.2, 0, 0.2, 0, 0.4, 0)$$

です. また, $y=0$ の文書は d_1 と d_2 の 2 個, $y=1$ は d_3 の 1 個ですから,

$$p(y) = \left(\frac{2}{3}, \frac{1}{3}\right) = (0.67, 0.33) \tag{5.2}$$

になります.

実際のテキストでは, $p(y)$ や $p(w|y)$ はどうなっているでしょうか. サポートサイトにある `livedoor.py` を, Livedoor コーパス `ldcc-20140209.tar.gz` を展開したフォルダの下にある `text` フォルダに適用して

```
% livedoor.py 展開したフォルダ/text livedoor.txt
```

を実行すると, 内部で MeCab で単語分割を行って URL 等を除き, 結果を

```
dokujo-tsushin 友人 代表 の スピーチ 、 独 女 は どう こなし て い
る ? もうすぐ ・ と 呼ば れる 6 月 。 独 女 の 中 に は 自
分 の 式 は まだ な のに ...
topic-news　　園山 真希 絵 が 経営 する 料理 店 を 閉店 、 その 経
緯 に 非難 の 声 29 日 、 料理 研究 家 の 園山 真希 絵 が 、 自
身 が 経営 する 家庭 料理 「 園山 」 を ...
```

のように, [ラベル]<TAB>単語列.. の形で1文書が1行のテキスト livedoor.txt に書き出します. この livedoor.txt に対して, $p(y)$ を求めるスクリプト nblabels.py を実行してみると,

```
% nblabels.py livedoor.txt
⇒ dokujo-tsushin  0.1181
  it-life-hack    0.1181
  kaden-channel   0.1173
  :
  sports-watch    0.1222
  topic-news      0.1045
```

のようになりました. また, 式(5.1)のように $p(w|y)$ を計算する nbprob.py を使って, “Peachy” カテゴリ $(y=6)$ での単語分布 $p(w|y=6)$ を求めると,

```
% nbprob.py livedoor.txt peachy
⇒ 、        -> 0.047520
  の        -> 0.046050
  に        -> 0.029309
  :
  私        -> 0.000718
```

表 5.1: Livedoor コーパスのカテゴリ名と本書で用いるカテゴリ番号, および文書数.

y	カテゴリ	カテゴリ名	文書数
1	MOVIE ENTER	movie-enter	870
2	独女通信	dokujo-tsushin	870
3	Sports Watch	sports-watch	900
4	IT ライフハック	it-life-hack	870
5	livedoor HOMME	livedoor-homme	511
6	Peachy	peachy	842
7	家電チャンネル	kaden-channel	864
8	エスマックス	smax	870
9	トピックニュース	topic-news	770

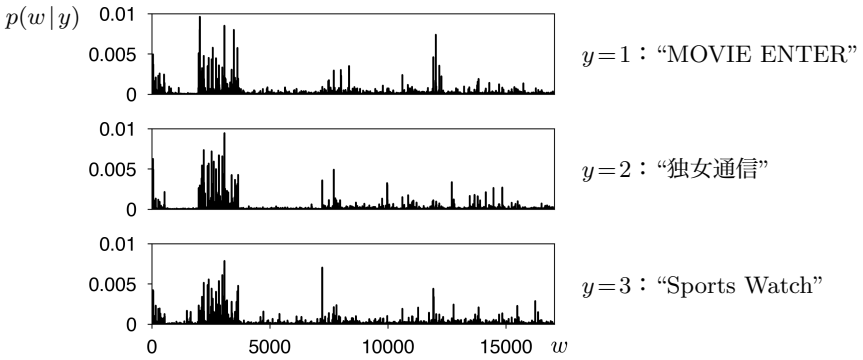


図 5.4: Livedoor コーパスにおけるカテゴリ別の単語確率分布 $p(w|y)$ の様子. 横軸の単語は, 辞書順に並んでいます. “が”, “の” など極端に確率の高い語があるため, 確率が 0.01 以上の単語はプロットから除外しています.

```
クリスマス -> 0.000716
恋愛     -> 0.000709
:
乗れ     -> 0.000002
考慮     -> 0.000002
```

のように計算することができます. このカテゴリー別の単語確率分布 $p(w|y)$ の様子を, 図 5.4 に示しました.

ノート：単語のカテゴリ所属確率の計算

上で計算したのはカテゴリ y の事前確率 $p(y)$ と, y から単語 w が出力される確率 $p(w|y)$ ですが, 実はこれから, w がカテゴリに所属する確率 $p(y|w)$ を求めることができます. というのは, 式(2.36)のベイズの定理から,

$$p(y|w) \propto p(y|w)p(y) \tag{5.3}$$

となるからです. 式(5.3)の右辺を和が 1 になるように正規化すれば, 単語 w の各カテゴリ y への所属分布 $p(y|w)$ が得られます.
Livedoor コーパスで実際に計算してみると, 表 5.2 のようになりました. 確かに, 各カテゴリの特徴がよく表れていることがわかります. 「劇場」のように, ほぼ特定のカテゴリにしか所属しない単語にどんなものがあるか, 調べてみると面白いでしょう (→演習 12).

カテゴリー	劇場	恋愛	Mac	携帯	ゴルフ	肌
movie-enter	87.9	8.4	0.0	1.5	0.3	1.5
dokujo-tsushin	1.0	47.5	0.2	10.5	0.6	13.7
sports-watch	0.0	2.3	0.0	1.9	11.7	1.7
it-life-hack	0.0	0.3	83.4	15.4	0.9	0.3
livedoor-homme	0.8	2.3	4.0	7.0	81.4	2.9
peachy	3.6	29.9	0.0	3.1	1.2	75.0
kaden-channel	1.7	2.4	8.9	32.1	2.1	4.2
smax	1.5	0.2	3.5	22.3	1.1	0.1
topic-news	3.5	6.7	0.0	6.2	0.8	0.6

(%)

表 5.2: Livedoor コーパスのナイーブベイズ法で, 単語がカテゴリーに所属する事後確率 $p(y|w)$ の例. わかりやすいよう, 確率を 100 倍して%で示しています.

5.1.1 文書の分類確率

これまでに求めた確率を使うと, 文書の確率を計算することができます. カテゴリ y のラベルを持つ文書 $d = w_1 w_2 \cdots w_L$ の確率とは, d と y の同時確率のことですから, 式(2.20)の確率の連鎖則から

$$p(d, y) = p(y)p(d|y) \tag{5.4}$$

$$= p(y) \prod_{w \in d} p(w|y) \qquad (\textbf{ナイーブベイズ法の文書確率})$$

となります. $p(d|y)$ がユニグラム確率の積 $\prod_{w \in d} p(w|y)$ に分解されることが, 単語集合の仮定に対応しています. たとえば図 5.3 のデータで, 新しい文書 $d = w_2 w_2 w_5 w_7$ がカテゴリ $y=0$ から生成される確率は,

$$\begin{aligned} p(d, y=0) &= p(y=0) \prod_{w \in \{2,2,5,7\}} p(w|y=0) \\ &= 0.67 \times (0.1 \times 0.1 \times 0.3 \times 0.1) = 0.0002 \end{aligned}$$

になります.

このように文書の確率が計算できると, 新しい文書 d がどのカテゴリに属するかを, 確率的に予測できるようになります. これはつまり, 確率分布

$$p(y|d) \qquad (y \in \{1, \cdots, K\}) \tag{5.5}$$

が知りたいということですから, 式(2.30)のベイズの定理を用いれば

$$p(y|d) \propto p(y)p(d|y) = p(y) \prod_{w \in d} p(w|y) \tag{5.6}$$

となり, この確率は $y \in \{1, \cdots, K\}$ について式(5.4)から求めることができるからです.

実際に計算してみましょう. 図 5.3 のデータで $d = w_1 w_2 w_6$ のとき, $p(y|d)$ は

$$p(y|d) \propto p(y)p(d|y) = \begin{cases} 0.67 \times 0.1 \times 0.1 \times 0.1 \\ 0.33 \times 0.2 \times 0.2 \times 0.4 \end{cases} = \begin{cases} 0.00067 & (y=0) \\ 0.00528 & (y=1) \end{cases} \tag{5.7}$$

となります. よって, これを和が 1 になるように正規化して,

$$p(y|d) = \left(\frac{0.00067}{0.00067 + 0.00528}, \ \frac{0.00528}{0.00067 + 0.00528} \right) = (0.113, \ 0.887) \tag{5.8}$$

が求める分布となります. つまり $d = w_1 w_2 w_6$ は $y=1$, すなわち迷惑メールである確率が 0.887 と非常に高い, ということがわかります.

このように, 式(5.4)の文書確率をもとにベイズの定理から, 式(5.6)を用いて

文書を属するカテゴリに確率的に分類する手法を, **ナイーブベイズ法**といいます. ナイーブ (素朴) とは, テキストの単語の順番を考えない単語集合の仮定のことをさしていますが, この単純な仮定にもかかわらず, この後でみるように, 分類については高い性能を持つことがわかっており, 文書分類の最も基本的なモデルになっています.

なお, 式(5.7)のような確率は一般に単語数が増えると, 確率が指数的に小さくなり, そのままでは計算機で表現できなくなってしまいます. よってこうした場合は, 対数をとって

$$\begin{aligned}(5.9)\qquad \log p(y=0|d) &\propto \log p(y) + \log p(d|y) \\ &= \log(0.67) + \log(0.1) + \log(0.1) + \log(0.1) \\ &= -0.405 - 2.303 - 2.303 - 2.303 = -7.314\end{aligned}$$

と計算するのがよいでしょう. このとき, $\log p(y)$ および $\log p(w|y)$ のそれぞれを「スコア」とみなせば, 式(5.9)は文書 d が $y=0$ に属する「スコア」を, ラベル y および d に含まれる各単語 w ごとに足し込んでいることになります. この合計スコアが最も大きいカテゴリが, 分類の結果になります. [*9]

ナイーブベイズ法の実験

実際に, Livedoor コーパスで実験してみましょう. 先に作成したデータファイル `livedoor.txt` を, 2 章の 2.6 節で説明したように行をシャッフルして, ランダムに 80%の学習データ, 10%の開発データ, 10%のテストデータに分割します.

```
% split.py livedoor.txt livedoor
```

これにより, `livedoor.train` (5888 行), `livedoor.dev` (736 行), `livedoor.test` (743 行) が作成されます. ランダムに分割するため, 以下の数値は人によって多少異なることに注意してください.

同じ場所にある `nb.py` を使って, 学習データからナイーブベイズ法のモデルを計算します. この計算は数秒で終了します.

```
% nb.py livedoor.train nb.livedoor
N = 5888 docs, V = 17053 vocabs, K = 9 classes.
alpha = 0.01, threshold = 10.
saving model to nb.livedoor.. done.
```

*9　一般に, 確率的でないアルゴリズムで天下りに定義された「スコア」は, 確率的に考え直すと, 確率の対数 (=情報量) とみなせることが非常に多くみられます.

nb.livedoor はナイーブベイズ法のパラメータ, つまり $p(y)$ および $p(w|y)$ が格納された gzip 圧縮済みの pickle ファイルです. 紙面の都合でスクリプトは載せませんので, 必ず中身を読んでから使ってください.

学習したモデルを使って新しいテキストのカテゴリを予測するには, スクリプト nbinf.py を使って, 次のように実行します.

```
% cat nb.txt
誕生 日 に ディナー で プレゼント を もらっ た ！
% nbinf.py nb.livedoor nb.txt
[ 0.086  0.028  0.000  0.000  0.012  0.872  0.001  0.000  0.001]
```

このテキストは, 表 5.1 から 6 番目の “Peachy” カテゴリの確率が高い (0.872) と予測されることがわかります. なお, 次に確率の大きい (0.086) カテゴリは “MOVIE ENTER” でした.

特定のテキストではなく, モデル全体の性能を評価するには, スクリプト nbeval.py を使って, livedoor.test のカテゴリの予測精度を次のようにして測ります [*10].

```
% nbeval.py nb.livedoor livedoor.test
accuracy = 91.92%
```

このテストデータでの文書カテゴリの予測精度は, 91.9%になりました.

結果の検証と「正解」ラベル ということは, 8.2% (60/743 文書) はカテゴリの予測を「間違った」ということになります. ただし, この 8.2%がどんな場合なのかには注意が必要です. 実際に予測を「間違った」60 件の文書を調べてみると, 表 5.3 のようになっており, 少なくとも人間の基準では, それほど間違いとは言えないことがわかります. 一方で, BERT のようなブラックボックスのニューラル手法は, このデータについて 97%程度の正解率を持っていますが[], これらは同じサッカーの記事でも, 「一般ニュース」と分類しているわけです. これはもしかするとニューラル手法が, Livedoor コーパスの各分野のライターの書き癖を学習しており, 内容に関係なく文体で分類しているのかもしれません (少なくとも, その可能性があります). したがって, 「誤り」とされたものが本当に誤りなのかは, こうして結果を確認して検討した方がよいでしょう. この

*10 ここでは開発データ livedoor.dev は使いません.

表 5.3: Livedoor コーパスでナイーブベイズ法が「間違えた」文書の例.

テキスト	予測ラベル	正解ラベル
川面を流れるの美しさに感動!散った桜が川面を埋め尽くす光景のあまりの美しさが話題に先週、花見客のマナーの悪さを紹介した「これからお花見の..	peachy	it-life-hack
五輪サッカー英韓戦を前に韓国では「最悪のシナリオ」の声 2 日、韓国のニュースサイト「」は、五輪サッカー男子で韓国がとの予選リーグ最終試合を..	sports-watch	topic-news
オトナ女子たちの圧倒的支持をうけ、ドラマ 10『』の一挙再放送が決定!昨年を出産した女優の木村佳乃さんが 2 年ぶりに連ドラ主演を務め、現在 NHK ..	movie-enter	dokujo-tsushin

結果は, 人手による「正解」ラベルが本当に常に正しいのか, を教えてくれる例ともいえます.

テキストの感情極性分類　もう一つ, ナイーブベイズ法のわかりやすい応用に**感情分析**があります. 感情分析には, テキストを肯定的・否定的などの**極性**に分類する感情極性分類 (sentiment analysis) と, “期待”, “驚き” などの基本感情を付与する基本感情分類 (emotion detection) があり, 愛媛大学の梶原らは, 日本語の SNS テキスト (ツイート) に対して上記の感情極性と基本感情の強度を付与したコーパス WRIME [134][135]を公開しています [*11].

ここでは, より簡単な感情極性分類を行ってみることにしましょう. WRIME のデータをダウンロードし, 含まれる `wrime-ver2.tsv` にサポートサイトにある `wrime.py` を実行すると, SNS のテキストを 86 ページの `neologdn` で正規化した後で MeCab で単語に分割し, `livedoor.txt` と同じ形式のデータ (8740 行) を作ることができます. `neologdn` をインストールしていない方は, 先に `pip install neologdn` などでインストールしておいてください.

```
% git clone https://github.com/ids-cv/wrime
% cd wrime
```

[*11] `https://github.com/ids-cv/wrime`

表 5.4: WRIME コーパスのツイート極性から計算した, positive と negative それぞれのカテゴリ y での単語確率 $p(w|y)$ の上位語.

(a) y＝positive の場合				(b) y＝negative の場合			
!	0.1060	♡	0.0730	ない	0.0930	怒ら	0.0800
♪	0.0890	好き	0.0720	“	0.0930	憂鬱	0.0800
最高	0.0850	フィンランド	0.0720	つらい	0.0880	無視	0.0800
楽しみ	0.0820	アニ	0.0710	嫌	0.0850	しんどい	0.0790
嬉しい	0.0810	ww	0.0710	しろ	0.0850	下痢	0.0790
曲	0.0800	^)/	0.0710	イライラ	0.0850	怒り	0.0790
かわいい	0.0770	Saint	0.0710	悪い	0.0850	寂しい	0.0790
サマ	0.0750	Snow	0.0710	くさい	0.0830	—	0.0790
おいしい	0.0740	さん	0.0710	吐き	0.0820	注意	0.0780
歌	0.0740	楽しかっ	0.0710	迷惑	0.0820	生理	0.0780
o	0.0740	きれい	0.0700	むり	0.0820	やらかし	0.0780
可愛	0.0730	*	0.0700	無理	0.0810	地獄	0.0780
可愛い	0.0730	＼(^	0.0700	なくなる	0.0810	吐き気	0.0780
良かっ	0.0730	しあわせ	0.0700	悲しい	0.0810	リスク	0.0780
ケーキ	0.0730	DVD	0.0700	めんど	0.0800	起き	0.0780

```
% wrime.py wrime-ver2.tsv > wrime.txt
% shuf wrime.txt | head -3
negative        掃除 機 が 動か ない (^ q ^) なんて こっ た ..
positive        おはよう ござい ます ! 今日 は 学び に 使い ..
negative        22 時 に 閉まる すき家 って はじめ て み た
```

このうち, ランダムに選んだ 740 ツイートをテストデータ, 残りの 8000 ツイートを学習データとして, 225 ページと同様に `nb.py` でナイーブベイズ法のモデルを計算してみましょう. この計算は 1 秒未満で終わります.

```
% shuf wrime.txt > wrime.shuffled.txt
% head -8000 wrime.shuffled.txt > wrime.train
% tail -740 wrime.shuffled.txt > wrime.test
% nb.py wrime.train nb.wrime
N = 8000 docs, V = 1690 vocabs, K = 2 classes.
alpha = 0.01, threshold = 10.
saving model to nb.wrime.. done.
```

計算したナイーブベイズ法の単語確率 $p(w|y)$ (式(5.4)) の上位語を, y が positive と negative の場合のそれぞれについて表 5.4 に示しました. 単語の極性が, きわめて自然な形で学習されていることがわかります. このモデルをもとに, テストデータのツイートの感情極性を予測してみましょう.

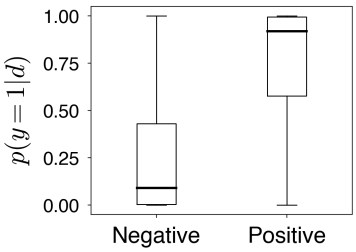


図 5.5: WRIME コーパスのテストデータのツイート極性の予測結果. 縦軸は, 予測された極性が Positive である確率を表します. 正解が Positive な場合には確率は 1 に近く, Negative な場合には 0 に近くなっており, 正しい学習が行えていることがわかります.

```
% nbinf.py nb.wrime wrime.test
⇒ 'negative': 0, 'positive': 1
negative [ 0.589  0.411] 手の痺れ一生治らん
positive [ 0.001  0.999] 先輩の息子が、まじ一瞬だけどつかまら..
positive [ 0.474  0.526] 私は、恐らく予定を組むのが好きな人な..
positive [ 0.464  0.536] チロルチョコにダイブしたい。
negative [ 1.000  0.000] わたしの伝え方がおかしいのか?なぜ、..
positive [ 0.739  0.261] 「落ちる」場面や、走っても走っても前..
positive [ 0.224  0.776] Shift+win キー+S キーで範囲指定の画面..
positive [ 0.091  0.909] おはようございます。
:
```

この場合, 数字は順に negative, positive の確率を表します. 短いツイートでは情報が少なく判断が難しくなりますが, ほぼ最初のカラムに示した「正解」のラベルに沿った予測ができていることがわかります. 図 5.5 に, 正解ラベルの Positive/Negative に分けた場合の Positive 確率の予測結果を示しました. 次のように実行して, ランダムな 20 ツイートの極性を予測値でソートしてみると, **非常に簡単なモデルにもかかわらず**, 確かにツイートの感情極性順に並んだ結果が得られることがわかります.

```
% nbinf.py nb.wrime wrime.test | shuf | head -20 | sort -k3 -n
⇒ positive [ 0.000  1.000] CD 工場のやつ今見た!「どももっちーで..
positive [ 0.000  1.000] EXIT の DVD やっすぅぅぅ神谷浩史小野大..
positive [ 0.000  1.000] サスケとサクラちゃんが最高!って、pi..
positive [ 0.000  1.000] 石川智晶姉さんきたあああ。大好きい..
positive [ 0.010  0.990] 飯塚昌明 ANNIVERSARYLIVE“e-XPO2020..
positive [ 0.025  0.975] やる気出てきたぞー＼(' ω')/
```

```
positive [ 0.026  0.974] 最近、肉蝮伝説という作品にハマって..
positive [ 0.029  0.971] 今期のロボットも二つとも面白いの?..
positive [ 0.071  0.929] RBC 入場!!
negative [ 0.445  0.555] お金無駄遣いなおしたい
negative [ 0.471  0.529] はーん
positive [ 0.519  0.481] はー今日のおもち本当に美味しかった..
positive [ 0.584  0.416] 大学会館前もある…
negative [ 0.611  0.389] パソコンがどんどんこわれてくひらが..
negative [ 0.743  0.257] 食費にいくら残る?
positive [ 0.752  0.248] 「京」の 1000 個の CPU の配布って最終回..
negative [ 0.824  0.176] 壊されてたまるかわたしは
negative [ 0.901  0.099] 君僕新刊もまだ買ってない (´Д`) 天使も
negative [ 0.942  0.058] リボ払いコワイ (使ってないとは言って..
negative [ 0.943  0.057] スペースもだけど夫のいびきうるさい..
```

5.2 ユニグラム混合モデル (UM)

前節で, ナイーブベイズ法が非常に簡単なモデル化と計算にもかかわらず, 文書を高い性能で確率的に分類できること, また, 人の付与した「正解」ラベルが必ずしも意味的な内容とは一致しないことをみてきました.

そもそも, テキストの内容が “一般ニュース” や “Peachy” といったラベルだけで語れるはずがなく, それぞれのカテゴリの中には, 様々な話題が含まれているはずです. しかし, それらの「話題」が何なのかがテキストに書かれているわけではありません. また一般のテキストには, そもそもラベルすらないことがほとんどです. 図 5.6 に示したのは, サポートサイトの本章のフォルダにある `jawiki.txt` の一部で, これは日本語 Wikipedia の記事全体 (約 62 万記事) からランダムに 10,000 記事を筆者が抜き出したテキストですが, これらの記事にはもちろんラベルはありません.

それでは, こうしたラベルのないテキストを扱うにはどうすればいいのでしょうか. 基本的な戦略は, 「**ラベルがないのだから, 自分で推定してしまえ**」ということです.

コラム：対数確率と logsumexp

式(5.8)の確率は式(5.7)の事後確率の和が 1 になるように正規化して求めたものですが, 文書の長さが大きくなると, この計算はそのまま行うことができません. というのは, 一定以上小さな確率は計算機では表すことができず, 0 になってしまうからです.[*12] よって, 式(5.9)のように対数をとって計算するのがよいでしょう.

K 個のカテゴリについて, 非常に小さな確率 $(p_1, p_2, \cdots, p_K)$ の対数 $(\ell_1, \ell_2, \cdots, \ell_K)$ がわかっていたとき, p を正規化して和を 1 にするには,

$$p'_k = \frac{p_k}{\sum_{k=1}^{K} p_k} = \exp\left(\ell_k - \log \sum_{k=1}^{K} \exp(\ell_k)\right) \tag{5.10}$$

で計算することができます. ただし上の式の $\log \sum_{k=1}^{K} \exp(\ell_k)$ は, そのまま計算すると, $\exp(\ell_k)$ (たとえば $\exp(-1000)$) がすべて 0 になってしまい, 求めることができなくなってしまう場合が多くあります.

このとき, $\ell_1, \ell_2, \cdots, \ell_K$ の中で最大のものを m とおけば,

$$\begin{aligned} \log \sum_{k=1}^{K} \exp(\ell_k) &= \log \left(e^{\ell_1} + e^{\ell_2} + \cdots e^{\ell_K}\right) \\ &= \log e^m \left(e^{\ell_1 - m} + e^{\ell_2 - m} + \cdots e^{\ell_K - m}\right) \\ &= m + \log \sum_{k=1}^{K} \exp(\ell_k - m) \qquad (\textbf{logsumexp}) \end{aligned} \tag{5.11}$$

となります. こうすると ℓ_k の間の差だけが exp の中に残るため, 値がすべて 0 になるのを防ぐことができます. 式(5.11)で $\log \sum \exp()$ の値を計算する関数を, logsumexp といいます [136, §2.5.4]. Python では `scipy.special.logsumexp()` で使えるほか,

```
from numpy import exp,log
def logsumexp (x):
    y = max (x)
    return y + log(sum(exp(x - y)))
```

で定義すれば計算することができます.

*12 執筆時の手元の環境 (64bit) では, 倍精度実数の最小値は 2.385×10^{-277} でした.

```
<doc>
Mozilla Application Suite（モジラ・アプリケーション・スイート）または Mozilla Suite（モジラ・スイート）は Mozilla Foundation によりプロジェクトを組んでオープンソースで開発されていたインターネットスイートであり、...
</doc>
<doc>
カート・ヴォネガット（Kurt Vonnegut、1922 年 11 月 11 日 - 2007 年 4 月 11 日）は、アメリカの小説家、エッセイスト、劇作家。1976 年の作品『スラップスティック』より以前の作品はカート・ヴォネガット・ジュニア（"Kurt Vonnegut Jr."）の名で...
</doc>
<doc>
世界の放送方式（せかいのほうそうほうしき）高精細度テレビジョン放送（HDTV："High Definition Television"）に対して従来のテレビ放送の画質は標準テレビジョン放送（SDTV："Standard Definition Televisi on"）とも言われ、ここでは主に標準テレビに分類される方式について記述している。...
</doc>
```

図 5.6: 日本語 Wikipedia からランダムに抽出した記事を集めた `jawiki.txt` の一部.

いま図 5.7 のように, 図 5.3 と同じデータでラベル y が未知だったとしましょう. この場合, 推定するのは人がつけたラベル y ではなく, 文書の潜在的なカテゴリを表す**潜在変数**ですので, 以後は y と区別して, z と書くことにします. 各文書 d_1, d_2, d_3 についてカテゴリ $z \in \{0, 1\}$ がわからないのですから, (完全に対称だと以下の計算が進まないため) $(0.5, 0.5)$ から少しずらした確率分布

$$p(z|d_1) = (0.4,\ 0.6), \quad p(z|d_2) = (0.6,\ 0.4), \quad p(z|d_3) = (0.4,\ 0.6)$$

を初期値としてみましょう [*13]. このとき $p(z)$ は, $z \in \{0, 1\}$ について各文書がもつ確率の総和として

$$p(z) \propto \sum_{n=1}^{3} p(z|d_n) = \begin{matrix} d_1 \quad d_2 \quad d_3 \\ \left\{ \begin{array}{l} 0.4 + 0.6 + 0.4 = 1.4 \\ 0.6 + 0.4 + 0.6 = 1.6 \end{array} \right. \end{matrix} \propto \left\{ \begin{array}{ll} 0.467 & (z{=}0) \\ 0.533 & (z{=}1) \end{array} \right. \tag{5.12}$$

と求められます. これは, 式(5.2)で文書がどちらかのカテゴリに所属するとして 0/1 で数えていた頻度を, 確率に拡張してソフトな和をとったと言っていいでしょう. また $p(w|z)$ も, 同様に式(5.1)で頻度を z ごとに確率 $p(z|d_n)$ で重

*13 この確率をクラスタリングでは, 文書の各クラスタへの**負担率** (responsibility) といいます. ここでは, 負担率がほぼ等しい状態を初期値にして計算を始めています.

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 \end{array} \\ \left(\begin{array}{ccccccc} 1 & & 1 & & 2 & & 1 \\ & 1 & 2 & & 1 & 1 & \\ 1 & 1 & & 1 & & 2 & \end{array} \right) \end{array} \begin{array}{c} z \\ ? \\ ? \\ ? \end{array}$$

図 5.7: ユニグラム混合モデルでの文書–単語行列の例. 図 5.3 と同じデータですが, ここでは各文書のラベル z は未知で, 各テキストでの単語の頻度だけが与えられています. このとき, 文書 d_1,d_2,d_3 について何がいえるでしょうか?

みづけて数え,

$$p(w|z) \propto \sum_{n=1}^{N} p(z|d_n)\, n(d_n, w) \tag{5.13}$$

で計算することができます.[*14] 実際に求めてみると, $z=0$ の場合は, 図 5.7 の行列を縦に読んで

$$\begin{cases} p(w_1|z=0) \propto 0.4\times 1 & +0.4\times 1 = 0.8 \\ p(w_2|z=0) \propto & 0.6\times 1 + 0.4\times 1 = 1.0 \\ p(w_3|z=0) \propto 0.4\times 1 + 0.6\times 2 & = 1.6 \\ \quad\vdots \qquad\qquad \vdots & \\ p(w_7|z=0) \propto 0.4\times 1 & = 0.4 \end{cases}$$

となりました. これを総和が 1 になるように正規化し, $z=1$ の場合も同様に計算すると,

$$\begin{cases} p(w|z=0) = (0.114,\ 0.143,\ 0.229,\ 0.057,\ 0.200,\ 0.200,\ 0.057) \\ p(w|z=1) = (0.150,\ 0.125,\ 0.175,\ 0.075,\ 0.200,\ 0.200,\ 0.075) \end{cases}$$

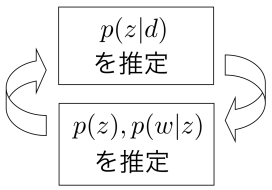
が得られます.

　すると, いま求めた $p(z), p(w|z)$ から, 式(5.4) すなわち

$$p(z|d) \propto p(z) \prod_{w\in d} p(w|z) \tag{5.14}$$

---

*14　ここでは直感的な説明をしていますが, なぜこの計算をしてよいのかは, この後の 5.2.2 節で EM アルゴリズムを導入した際に説明します.



アルゴリズム 3: UM の EM アルゴリズム

パラメータ $p(z)$, $p(w|z)$ を適当に初期化する.
while 収束するまで **do**
  (* E ステップ *)
  for $n = 1 \cdots, N$ **do**
    $p(z_n|d_n)$ を式(5.14)で計算する.
  (* M ステップ *)
  $p(z_n|d_n)$ に基づいて, パラメータ $p(z)$, $p(w|z)$ を式(5.12)および式(5.13)で更新する.

図 5.8: ユニグラム混合モデルの推定法の概要と EM アルゴリズム.

を使って, $p(z|d)$ を更新することができます. たとえば, 文書 d_2 については

$$\begin{array}{cccccc} & z & & w_2 & w_3 & w_5 & w_6 \\ \left\{ \begin{array}{l} p(z=0|d_2) \propto 0.467 \times 0.143 \times (0.229)^2 \times 0.200 \times 0.200 = 0.00014008 \\ p(z=1|d_2) \propto 0.533 \times 0.125 \times (0.175)^2 \times 0.200 \times 0.200 = 0.00008162 \end{array} \right. \end{array}$$

から,

$$p(z|d_2) = (0.632, 0.368)$$

となり, $p(z|d_2)$ が $(0.6, 0.4) \rightarrow (0.632, 0.368)$ に更新されることがわかります. このように, 適当な初期値から始めて

$$\left\{ \begin{array}{l} p(z|d) \text{ を推定する} \\ p(z), p(w|z) \text{ を推定する} \end{array} \right. \tag{5.15}$$

ことを図 5.8 のアルゴリズムに従って繰り返すと, 表 5.5 に示したようにこの計算は 7 回程度の繰り返しで収束し, 結果として

$$p(z|d_1) = (1.000,\ 0.000),\ p(z|d_2) = (1.000,\ 0.000),\ p(z|d_3) = (0.000,\ 1.000)$$

が得られます. **何も教えていないのに, 教師データ y があったときと同じ分類ができてしまいました!** $p(z|d_1)$ は最初, $(0.4,\ 0.6)$ と $z=1$ の方が確率が高かったにもかかわらず, 式(5.15)の計算の繰り返しの中で逆転し, 正しい確率分布が得られていることに注意してください.

このように, 式(5.15)の計算を繰り返すことで, 教師なしでテキストをクラス

表 5.5: ユニグラム混合モデルの学習過程. ここでは 7 ステップで学習が収束し, 教師なしで文書を 2 つの種類に分類できています.

文書	d_1		d_2		d_3	
繰り返し	$z=0$	$z=1$	$z=0$	$z=1$	$z=0$	$z=1$
0(初期値)	0.400	0.600	0.600	0.400	0.400	0.600
1	0.399	0.601	0.630	0.370	0.367	0.633
2	0.423	0.577	0.668	0.332	0.303	0.697
3	0.519	0.481	0.715	0.285	0.183	0.817
4	0.769	0.231	0.776	0.224	0.047	0.953
5	0.982	0.018	0.873	0.127	0.002	0.998
6	1.000	0.000	0.988	0.012	0.000	1.000
7	**1.000**	**0.000**	**1.000**	**0.000**	**0.000**	**1.000**

タリングする方法を, **ユニグラム混合モデル** (Unigram Mixtures, UM) [137] といいます. これまでの説明でわかるように, UM は「教師なしナイーブベイズ法」ともいえ, 機械学習で K 平均法[20]として知られるクラスタリングを, テキストの多項分布の場合に適用したものになっています.

ユニグラム混合モデルの実験

実際の文書でも計算してみることにしましょう. 図 5.6 に示したテキストは, 日本語版 Wikipedia の全記事 (執筆時点で約 62 万記事) からランダムに 1 万記事の概要部分を筆者が抽出したもので, サポートサイトに `jawiki.txt` として置いてあります. これを SVMlight 形式のデータにするには, 同じフォルダにある `text2data.py` を次のように実行します. 最後の数字は, 単語頻度の閾値です. スクリプトの使い方は, 中身を読んでみてください.

```
   % text2data.py jawiki.txt jawiki 10
⇒  writing dic to jawiki.lex.. done.
   writing data to jawiki.dat.. done.
```

これにより, 10000 行のデータ `jawiki.dat` と, 7509 個の語彙と ID の対照表 `jawiki.lex` が作られます.[*15] このデータについて, UM の実装 `um.py` を実行してみましょう.

*15　UM の場合は「が」「の」のような機能語を入れてしまうと, それに引っぱられてクラスタリングがうまく働かないため, 平仮名だけからなる単語を除くといった前処理を行っています.

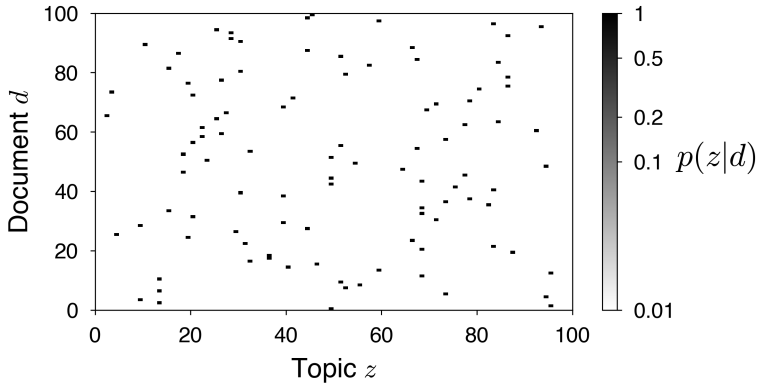


図 5.9: 日本語 Wikipedia コーパス `jawiki.txt` から学習された UM による, 各文書のトピックへの所属確率 $p(z|d)$ (最初の 100 文書). トピック数は $K=100$ としました. ほとんどの $p(z|d)$ はほぼ 1 になっており, 文書ごとに異なるトピックに所属していることがわかります. このプロットは `umplot.py` で作ることができます.

```
% um.py -K 100 jawiki um.jawiki.K100
⇒ UM: 10000 documents, 7509 words in vocabulary.
iter[ 1] : PPL = 2580.88
iter[ 2] : PPL = 1541.67
iter[ 3] : PPL = 684.78
iter[ 4] : PPL = 625.30
iter[ 5] : PPL = 617.85
iter[ 6] : PPL = 615.83
iter[ 7] : PPL = 614.67
iter[ 8] : PPL = 614.36
converged.
saving model to um.jawiki.K100.. done.
```

スクリプトの使い方は, `% um.py` をそのまま実行すれば表示されます. ここでは, $K=100$ 個のカテゴリを仮定しました. 図 5.9 に示したように, 学習されたモデルで $p(z|d)$ を表示してみると, 日本語 Wikipedia の文章がさまざまなカテゴリに所属する様子が, 自動的に学習されていることがわかります.

5.2.1　トピックの解釈と自己相互情報量

ラベルのない生テキストについて UM を学習することで, K 個のカテゴリ z ごとの単語の出力分布 $p(w|z)$ と, カテゴリの事前確率 $p(z)$ を学習することができました. この場合の z はナイーブベイズ法のように人が与えたカテゴリではなく, データから統計的に学習されたものですから, 一般にこれらを**トピック** (話題) とよびます. 教師なしナイーブベイズ法である UM は, 最も簡単な**トピックモデル**の一つです.

それぞれのトピック z が何を表しているかは, 単語分布 $p(w|z)$ の様子を見てみればわかるはずです. ただし, この際には注意が必要です. 各トピック z について単に確率 $p(w|z)$ の大きい単語を表示すると, 表 5.6(a) に示したように “年” や “日” といった語が上位に来てしまい, トピックの意味がほとんどわからなくなってしまいます. これは, 考えると当然の現象です. UM では, 文書に含まれる語がすべて, あるトピック z から $p(w|z)$ に従ってサンプリングされたと考えています. すると, どの話題でも共通して現れる語は, どのトピックでも高い確率にならなければ, データをよく説明できないからです. ここでは “の” のような平仮名だけからなる語を前処理で除いていますが, 除かない場合はこうした語がすべてのトピックで出現確率の上位を占めることになります. [*16]

したがって重要なのは, 単語 w のトピック z での生起確率 $p(w|z)$ 自体の大きさではなく, w の平均的な出現確率

$$p(w) = \sum_{z=1}^{K} p(w,z) = \sum_{z=1}^{K} p(w|z=k)p(z=k) \tag{5.16}$$

との比をとった,

$$\frac{p(w|z)}{p(w)} \tag{5.17}$$

の値でしょう. 式(5.17)は, 単語 w がトピック z で「通常より何倍高い確率で

*16　この例からわかるように, 言語では固定された「ストップワード」のリストを指定して, それらを除外すれば問題が解決するわけではありません. 必ずこのように, リストには含まれない, 同様の意味が薄い語が現れるからです. ある語がストップワードであるか否かは, ルールではなく, 単語の振る舞いから統計的に判断すべき問題です.

表 5.6: 日本語 Wikipedia コーパスでの各トピックを表す特徴語の計算.

(a) 生成確率 $p(w|z)$ を使った上位語

Topic 1		Topic 10		Topic 20		Topic 30	
年	0.056	年	0.037	年	0.066	年	0.059
日	0.025	月	0.025	月	0.019	大学	0.027
月	0.024	日	0.017	日	0.019	月	0.026
音楽	0.022	システム	0.009	世	0.016	日本	0.021
作曲	0.019	社	0.009	王	0.012	日	0.019
曲	0.017	的	0.007	伯	0.010	会	0.013
家	0.016	法律	0.007	前	0.010	部	0.012
者	0.015	法	0.006	公	0.009	者	0.011
指揮	0.011	日本	0.006	紀元	0.008	長	0.010
作品	0.009	使用	0.006	語	0.007	委員	0.009

(b) PMI (w, z) を使った上位語

Topic 1		Topic 10		Topic 20		Topic 30	
大刀	4.502	inotify	4.538	ロジスティック	4.528	法科	4.079
査証	4.343	カーネル	4.418	τ	4.516	学長	3.625
ジュリアン	4.313	ボトル	4.292	ネブカドネザル	4.508	ラトビア	3.612
グレコ	4.282	ボランティア	4.055	マウイ	4.499	民兵	3.545
管弦	4.170	上杉	3.989	巨星	4.244	医科	3.495
奏	4.066	宝石	3.787	諸侯	4.211	土木	3.491
石巻	4.051	長尾	3.778	写像	4.199	商科	3.416
ヴァイオリニスト	4.038	帯域	3.775	伯	4.126	法学	3.413
長調	4.029	GNU	3.753	司馬	4.068	志願	3.353
初演	3.913	ペット	3.684	ブランデンブルク	4.055	黒川	3.306

(c) NPMI (w, z) を使った上位語

Topic 1		Topic 10		Topic 20		Topic 30	
作曲	0.462	カーネル	0.501	伯	0.476	工学	0.373
管弦	0.458	inotify	0.495	ロジスティック	0.457	大学	0.371
指揮	0.437	ボトル	0.490	τ	0.449	委員	0.368
ピアノ	0.437	ボランティア	0.463	ネブカドネザル	0.445	会長	0.353
交響	0.434	ペット	0.444	マウイ	0.441	理事	0.351
楽団	0.434	サーバ	0.438	ブランデンブルク	0.434	学会	0.347
大刀	0.434	帯域	0.430	諸侯	0.430	法学	0.345
グレコ	0.422	上杉	0.429	司馬	0.430	土木	0.332
音楽	0.422	Linux	0.427	辺境	0.428	長	0.327
協奏	0.421	宝石	0.425	写像	0.423	学長	0.327

現れるのか」を表しています. “帯域” のように, 全体的な確率は低くても, 特定のトピックでは高い確率で現れる語は, $p(w|z)/p(w)$ の値は大きくなります. 一方で “日” のように, 出現確率は高くても, どのトピックでもほぼ同じ確率で現れる場合は, $p(w|z)/p(w)$ はほぼ 1 になると考えられます. よって, 式(5.17) の対数をとって,

$$\mathrm{PMI}\,(w,z) = \log \frac{p(w|z)}{p(w)} \tag{5.18}$$

を計算すれば, “的” のような語ではほぼ $\log 1 = 0$, “帯域” のような語では > 0 になるでしょう. つまり, この形で単語の統計的な「重み」が得られます. 表 5.6(b) に, `jawiki` コーパスについて式(5.18)で計算した値が大きい単語を各トピックについて示しました. これで, だいぶトピックの差がみえてきました.

式(5.18)を, w と z の**自己相互情報量** (Pointwise Mutual Information, PMI) といいます. というのは, 式(2.30)のベイズの定理を用いれば, 式(5.18)は

$$\log \frac{p(w|z)}{p(w)} = \log \frac{p(w,z)}{p(w)p(z)} \tag{5.19}$$

と変形することができ, この値は 2 章で学んだ, 確率変数 X と Y の間の相互情報量

$$I(X,Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{5.20}$$

で, Σ の中で期待値をとる対象の各要素 (pointwise) になっているからです.

正規化自己相互情報量　PMI を使うことで, 各トピックの意味はかなり明らかになりました. ただ, PMI の上位語にはまだ, 「ジュリアン」「ラトビア」などの特殊な単語が含まれてしまっています. これは PMI に一般にみられる現象で, 式(5.18)で分母に $p(w)$ があるため, $p(w)$ が小さい, すなわち稀な語であればあるほど, PMI の値が大きくなってしまいます.

この欠点を補うために, **正規化自己相互情報量** (Normalized PMI, NPMI) が 2009 年に提案されました［63］. 式(5.18)の PMI は, $p(w|z) = 1$, すなわち $p(w) = p(z)$ のときに最大値 $-\log p(w)$ をとることに注意しましょう.[*17]

---

*17　これから, w と z が完全に相関している, すなわち単語 w がカテゴリ z でしか現れない場

よって, PMI をその最大値で割った

$$\log \frac{p(w\,|\,z)}{p(w)} \Big/ \big(-\log p(w)\big) \tag{5.21}$$

を考えることができ, これを正規化自己相互情報量 (NPMI) といいます. NPMI は $-1 \leq \mathrm{NPMI} \leq 1$ の値をとり,

- w と z が完全に相関しているとき 1,
- w と z が独立なとき 0,
- w と z が完全に逆相関しているとき -1

となり, w と z の相関を表すために理想的な量となっています.

なお, 式(5.21)は

$$\frac{\log p(w\,|\,z) - \log p(w)}{-\log p(w)} = 1 - \frac{\log p(w\,|\,z)}{\log p(w)} \tag{5.22}$$

とも書き直すことができ, 式(2.63)で学んだトピック z での単語 w の自己情報量 $-\log p(w\,|\,z)$ の, w の平均的な情報量 $-\log p(w)$ に対する比を最大値 1 から引いた値になっています. NPMI は一般には, 式(5.19)で最大値 $-\log p(w,z)$ $(= -\log p(w) = -\log p(z))$ で割って,

$$\mathrm{NPMI}\,(w,z) = \log \frac{p(w,z)}{p(w)p(z)} \Big/ \big(-\log p(w,z)\big) \tag{5.23}$$

(**正規化自己相互情報量**)

と定義されています[63].

表 5.6(c) に, NPMI で計算した UM の各トピックの特徴語を示しました. PMI と比べ, “太刀” や “ラトビア” といった低頻度語の順位が下がり, 各トピックの意味をより適切に表していることがわかります. こうした特徴から, NPMI はトピックモデルにおいて, トピックを説明する標準的な指標として広く使われています[]. なお, 表 5.6 ではまだ 1 つのトピックに複数の話題が混入してしまっていますが, この後で説明するように, これは UM をベイズ学習することで, 大きく改善することができます (表 5.7).

合は, PMI は $p(w)$ が小さい稀な語ほど大きな値をとってしまう, ということがわかります.

(a) 通常の統計モデル　(b) 潜在変数 z がある統計モデル

図 5.10: 統計モデルとパラメータの推定. $\mathcal{D}$ はデータを, θ はパラメータを表します.

5.2.2　EM アルゴリズムによる学習

ところで, 図 5.8 の学習アルゴリズムはなぜ収束し, 教師なしで UM の学習ができるのでしょうか? それは, このアルゴリズムが **EM アルゴリズム**という一般的な学習法の一種になっているからです.

EM アルゴリズムは, 因果推論でも有名な Rubin らによって 1970 年代に提案され[138], 1990 年代には最先端の手法として機械学習で多く使われました[20]. EM アルゴリズムは, 最近のニューラルネットワークの VAE (変分自己符号化器) [139]の学習法の基礎ともなっている重要な方法ですので, 以下でみていくことにしましょう.

最尤推定とベイズ推定　まず, 統計モデルとは図 5.10 のように, データ $\mathcal{D}$ の背後にパラメータ θ があり, $\mathcal{D}$ を最もよく説明する θ を求めることです. このとき, θ から $\mathcal{D}$ が生成される確率 (尤度) を最大にする $\theta = \widehat{\theta}$ を求める, すなわち

$$\widehat{\theta} = \operatorname*{argmax}_{\theta} p(\mathcal{D}|\theta) \tag{5.24}$$

を点推定するのが**最尤推定**です. いっぽう, 有限のデータから $\widehat{\theta}$ が一点で正確に求まるはずがありませんから, $\mathcal{D}$ が与えられた下での θ の確率分布

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta) \tag{5.25}$$

を求めるのが**ベイズ推定**です. 式(5.24)と式(5.25)を見比べると, 最尤推定とは θ の事前分布に一様分布 $p(\theta) \propto 1$ を仮定した上で, $p(\theta|\mathcal{D})$ をデルタ関数 $\delta(\theta = \widehat{\theta})$ で一点近似することに対応することがわかります. EM アルゴリズムは最尤推定を行う方法ですので, ここではしばらく式(5.24)で考えることにしましょう.

EM アルゴリズム　モデルにパラメータ θ だけでなく, UM の各文書のクラス

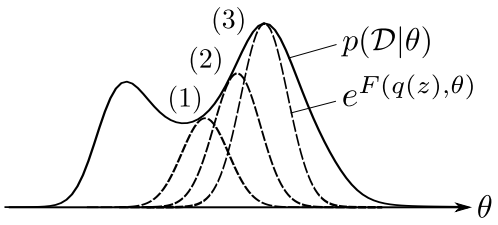


図 5.11: EM アルゴリズムによる学習. 実際には対数尤度 $\log p(\mathcal{D}|\theta)$ を最大化しますが, わかりやすさのため, もとの確率密度 $p(\mathcal{D}|\theta)$ の空間で示しています. EM アルゴリズムでは $\log p(\mathcal{D}|\theta)$ の下界 $F(q(z),\theta)$ を, 適当な初期値から始めて, E ステップと M ステップを繰り返すことで次々と最大化します. 初期値によっては, 左側の局所解に陥ることもあることに注意してください.

タ番号のように**潜在変数** z があるとき, 式(5.24)の尤度は

$$p(\mathcal{D}|\theta) = \int p(\mathcal{D}, z|\theta) dz \tag{5.26}$$

と書き換えることができます. z が離散の場合は上の積分は和になりますが, ここでは一般的に積分で表しました. 実際には, きわめて小さな値になる式(5.26)の確率の代わりに, その対数をとった

$$\log p(\mathcal{D}|\theta) = \log \int p(\mathcal{D}, z|\theta) dz \tag{5.27}$$

を最大化したいのですが, この式は log の中に積ではなく和が入っているため, $\log p()$ の形に分解することができません. そこで, 一つ工夫をします. 分母・分子に同じ値 (補助分布) $q(z)$ をかけて, **??**ページの Jensen の不等式を使えば, 式(5.27)の対数尤度は

$$\begin{aligned} \log p(\mathcal{D}|\theta) &= \log \int p(\mathcal{D}, z|\theta) dz = \log \int q(z) \frac{p(\mathcal{D}, z|\theta)}{q(z)} dz \\ &\geq \int q(z) \log \frac{p(\mathcal{D}, z|\theta)}{q(z)} dz \ \equiv F(q(z), \theta) \end{aligned} \tag{5.28}$$

と, 不等式で下から抑えることができます. 補助分布 $q(z)$ を用いることで, 無事, $\log p(\mathcal{D}, z|\theta)$ の形にすることができました！

式(5.28)は真の対数尤度 $\log p(\mathcal{D}|\theta)$ の下界ですから, 図 5.11 に示したよう

に式(5.28)を θ および $q(z)$ について最大化することで, 真の対数尤度に下から近づくことができます.[*18] これを行うのが EM アルゴリズムです. 具体的には,

E ステップ ：θ を固定して, $F(q(z),\theta)$ を $q(z)$ について最大化
M ステップ：$q(z)$ を固定して, $F(q(z),\theta)$ を θ について最大化

を交互に繰り返すことで下限を逐次最大化します. E は Expectation (期待値), M は Maximization (最大化) を意味します. それぞれ, どんな計算になるでしょうか. やや抽象的ですが, 先に一般論をみてみましょう. UM の場合の具体例は, その後で説明します.

E ステップ　$q(z)$ については, 下限 $F(q(z),\theta)$ は

(5.29)
$$\begin{aligned} F(q(z),\theta) &= \int q(z)\log\frac{p(\mathcal{D},z|\theta)}{q(z)}dz = \int q(z)\log\frac{p(z|\mathcal{D},\theta)p(\mathcal{D}|\theta)}{q(z)}dz \\ &= \int q(z)\log\frac{p(z|\mathcal{D},\theta)}{q(z)}dz + \log p(\mathcal{D}|\theta) \\ &= \underbrace{-D(q(z)||p(z|\mathcal{D},\theta))}_{(*)} + \log p(\mathcal{D}|\theta) \end{aligned}$$

となることに注意しましょう. $*$ の部分は式(2.69)で学んだ, $q(z)$ と $p(z|\mathcal{D},\theta)$ の KL ダイバージェンスですから, この値は

(5.30)
$$q(z) = p(z|\mathcal{D},\theta)$$

のとき最小になり, よって F が最大になります. すなわち, $q(z)$ としては実際には, 現在のモデル (パラメータ θ) での z の予測分布 $p(z|\mathcal{D},\theta)$ をとればよい, ということがわかります.

M ステップ　θ の方はどうでしょうか. 式(5.28)を θ について変形すると,

(5.31)
$$\begin{aligned} F(q(z),\theta) &= \int q(z)\log\frac{p(\mathcal{D},z|\theta)}{q(z)}dz = \int q(z)\log p(\mathcal{D},z|\theta)dz \\ &\quad - \int q(z)\log q(z)dz = \Big\langle \log p(\mathcal{D},z|\theta)\Big\rangle_{q(z)} + H(q(z)) \end{aligned}$$

*18　この $F(q(z),\theta)$ を, 物理学とのアナロジーで自由エネルギーともいいます.

です. $H()$ は式(2.65)のエントロピー関数で, $\langle\cdots\rangle_p$ は, $\cdots$ を p について期待値を取ることを表しています. この式はこれ以上簡単になりませんので, F を θ について最大化するためには,

$$Q(\theta) = \Big\langle \log p(\mathcal{D}, z|\theta) \Big\rangle_{q(z)} \tag{5.32}$$

に対して, $\partial Q/\partial\theta = 0$ とおくなどして θ について最大化することになります. 式(5.32)は, 一般に **Q 関数**とよばれています. Q 関数は, 現在のモデルを使った潜在変数 z とデータの同時確率を, 未知の z について期待値をとったものであることに注意してください.

UM の EM アルゴリズムの導出

抽象的な話が続いたので, 具体的に計算してみましょう. UM で文書 d が生成される確率モデルは, 式(5.4) で y (ここでは z) について和をとった

$$\begin{aligned} p(d) &= \sum_{z=1}^{K} p(d,z) = \sum_{z=1}^{K} p(d|z)p(z) \\ &= \sum_{z=1}^{K} p(z) \prod_{w\in d} p(w|z) \qquad\qquad (\textbf{UM の文書確率}) \end{aligned} \tag{5.33}$$

となります. よって N 個の文書 $\mathcal{D} = \{d_1, d_2, \cdots, d_N\}$ 全体の確率は,

$$p(\mathcal{D}) = \prod_{n=1}^{N} \left(\sum_{z=1}^{K} p(z) \prod_{w\in d_n} p(w|z) \right) \tag{5.34}$$

になり, 対数をとれば

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log \left(\sum_{z=1}^{K} p(z) \prod_{w\in d_n} p(w|z) \right) \tag{5.35}$$

です. パラメータは $\theta = \{\, p(z), p(w|z) \,\}_{z=1}^{K}$ です.

E ステップ $p(z|\mathcal{D},\theta)$ とはこの場合, 各文書 $d \in \mathcal{D}$ について, その負担率 $p(z|d)$ のことです. これは, 式(5.14)で計算できるのでした.

M ステップ Q 関数の中身である $\log p(\mathcal{D}, z|\theta)$ は, この場合は

$$\log p(\mathcal{D}, z\,|\,\theta) = \log\left(p(z)\prod_{w\in d} p(w\,|\,z)\right) = \log p(z) + \sum_{w\in d}\log p(w\,|\,z)$$

になります. Q 関数はこの z に関する期待値なので, $q(z)=p(z\,|\,d)$ でしたから,

$$\left\langle \log p(\mathcal{D}, z\,|\,\theta)\right\rangle_{q(z)} = \sum_{z=1}^{K} p(z\,|\,d)\log p(z) + \sum_{w\in d}\sum_{z=1}^{K} p(z\,|\,d)\log p(w\,|\,z) \tag{5.36}$$

となります. 文書は N 個あるので, 全体の Q 関数は

$$Q(\theta) = \sum_{n=1}^{N}\left[\sum_{z=1}^{K} p(z\,|\,d_n)\log p(z) + \sum_{w\in d_n}\sum_{z=1}^{K} p(z\,|\,d)\log p(w\,|\,z)\right] \tag{5.37}$$

となります.

$p(z)$ について最大化： $Q(\theta)$ を $p(z)$ に関して微分すると [*19], $(\log x)' = \dfrac{1}{x}$ ですから

$$\frac{\delta Q(\theta)}{\delta p(z)} = \sum_{n=1}^{N}\frac{p(z\,|\,d_n)}{p(z)} \tag{5.38}$$

です. $\sum_z p(z)=1$ なので, 制約項 $\left(\sum_z p(z)-1\right)$ を追加すると, ラグランジュの未定乗数法 [*20] により, 未定乗数を λ として

$$\frac{\delta}{\delta p(z)}\left(Q - \lambda\left(\sum_z p(z)-1\right)\right) = 0 \tag{5.39}$$

を解けばよいことになります. よって式(5.38)から

$$\sum_{n=1}^{N}\frac{p(z\,|\,d_n)}{p(z)} - \lambda = 0$$

となり, これを解いて,

---

*19 $p(z)$ は関数なので, 形式的に変分 $\delta Q/\delta p(z)$ を用いていますが, Q の中に $p(z)$ の微分に関する項はありませんので, これは $p(z)$ に関する通常の偏微分になります[140]. $p(z=k)=\mu_k$ のようにおいて, 変数 μ_k について最大化すると考えてもよいでしょう.

*20 ラグランジュの未定乗数法について知らない方は, 紙幅の問題で本書では解説しませんので, [7, §1.2.3]などを参照してください.

(5.40) $$p(z) = \frac{1}{\lambda} \sum_{n=1}^{N} p(z|d_n)$$

となります. ここで λ についても微分すれば, 当たり前ですが

$$\sum_{z} p(z) = 1$$

ですから, 式(5.40)で λ は和がちょうど 1 になるように決めればよいことがわかり, 式(5.12)が得られます.

$p(w|z)$ について最大化： $Q(\theta)$ を $p(w|z)$ について微分すると, 同様に

(5.41) $$\frac{\delta Q(\theta)}{\delta p(w|z)} = \sum_{n=1}^{N} \sum_{v \in d_n} \mathbb{I}(v=w) \frac{p(z|d_n)}{p(w|z)} = \frac{1}{p(w|z)} \sum_{n=1}^{N} p(z|d_n)\, n(d_n, w)$$

となります. よって, 同様にラグランジュの未定乗数法により

(5.42) $$\frac{1}{p(w|z)} \sum_{n=1}^{N} p(z|d_n)\, n(d_n, w) - \lambda = 0$$

から,

(5.43) $$p(w|z) = \frac{1}{\lambda} \sum_{n=1}^{N} p(z|d_n)\, n(d_n, w)$$

となり, 同様にして式(5.13)が得られました. □

EM アルゴリズムでは, E ステップと M ステップの繰り返しによって下限 $F(q(z), \theta)$ が単調に増加します. よって, 235 ページの実装 `um.py` のように学習データのパープレキシティを計算して, 減少が一定の閾値以下になったかどうかで収束を判定するとよいでしょう.

5.2.3* UM のベイズ学習

前節で, 図 5.8 のアルゴリズムが EM アルゴリズムになっていること, およびその導出について理解することができました. EM アルゴリズムでは, E ステップと M ステップを交互に行うことで, 真の対数尤度を図 5.11 のように下から近似して最大化します. これにより, 式(5.28)の $F(q(z), \theta)$ はつねに増加して極

大値に近づきます. しかし, EM アルゴリズムは常に山登りを行うため, HMM の学習について図 4.20 でみたように, 初期値に依存して一番近くの山 (局所解) につかまってしまうという大きな欠点があります. 複雑な問題で対数尤度の曲面に凸凹が大きいほど, この問題は顕著になります.

これを解消するもっとも簡単な方法は, EM アルゴリズムの E ステップで負担率を計算して期待値をとるのではなく, この確率に従ってランダムにサンプリングを行うことでしょう. 常に山を登るのではなく, 時には下ることも許せば, より広い空間を探索できます. 具体的には, UM の場合は E ステップでは負担率 $p(z|d_n)$ に従ってトピック z_n を文書ごとに毎回サンプリングし, M ステップではその z_n をナイーブベイズ法における「正解」ラベルと同様にみなして, パラメータ $\theta=\{p(z),p(w|z)\}$ を更新します. E ステップをサンプリングで置き換えるこの方法を, **モンテカルロ EM アルゴリズム** [141]といいます [*21].

モンテカルロ EM アルゴリズムは局所解から脱出できる有用な方法ですが [*22], M ステップでは θ について最大化を行っているため, 全体にみると EM アルゴリズムの逐次最大化の一種となっており, 真の最大値を探索することはできません. より正しくは, θ についても最尤推定ではなく, 事後分布全体からサンプリングを行うベイズ学習を行う必要があります.

幸いにして, UM の場合は 4 章で学習した **Gibbs サンプリング**を容易に行うことができます. 各文書 $d_1,\cdots,d_N$ について, その潜在トピック $\mathbf{z}=z_1,\cdots,z_N$ を $p(z_n|d_n)$ に従ってサンプリングしたとしましょう. このとき, $\mathbf{z}$ の中でトピックが k となった文書の数を $n(k)=\sum_{n=1}^{N}\mathbb{I}(z_n=k)$ とおけば, $\boldsymbol{\mu}=\{p(z)\}$ $(z=1,\cdots,K)$ が事前分布

$$\boldsymbol{\mu}\sim \mathrm{Dir}(\alpha,\cdots,\alpha) \tag{5.44}$$

に従っているとしたとき, その事後分布は式(4.38)と同様に

$$\boldsymbol{\mu}|\mathbf{z}\sim \mathrm{Dir}(\alpha+n(1),\cdots,\alpha+n(K)) \tag{5.45}$$

---

*21 モンテカルロとはカジノで有名な地中海の都市で, 乱数を使って期待値を求める方法をこの名前にしたのは, 第二次世界大戦中に原爆開発のために作られたアメリカのロスアラモス研究所で, 数学者のウラムとフォン・ノイマンによるものです[142].

*22 モンテカルロ EM アルゴリズムは, 潜在変数 z に指数的な組み合わせの可能性があるなど, すべての場合を計算して期待値をとることが実質的に不可能な場合にも有益なアルゴリズムです.

アルゴリズム 4: UM の周辺化 Gibbs サンプリング

```
for n in 1,...,N do                    (* 初期化 *)
    z_n = randint(K)                   (* 1...K の乱数で初期化 *)
    n'(z_n)++
    for w in d_n do
        m'(z_n, w)++
while 収束するまで do                   (* Gibbs サンプリング *)
    for n in randperm(N) do            (* 1...N をシャッフル *)
        n'(z_n)--                      (* カウントを減らす *)
        for w in d_n do
            m'(z_n, w)--
        z_n を式(5.14)および式(5.49)に従ってサンプリング
        n'(z_n)++                      (* カウントを増やす *)
        for w in d_n do
            m'(z_n, w)++
式(5.49)を用いて, p(z), p(w|z) の期待値を求める
```

図 5.12: 周辺化 Gibbs サンプリングによる UM のベイズ学習のアルゴリズム.

となります. $\boldsymbol{\phi}_k = \{p(w|k)\}$ についても, トピック $z_n = k$ となった文書だけを集めて, その中での各単語 w の頻度 $m(k,w)$ を計算すれば, $\boldsymbol{\phi}_k$ が事前分布

$$\boldsymbol{\phi}_k \sim \mathrm{Dir}(\beta, \cdots, \beta) \tag{5.46}$$

に従っているとき, 事後分布は式(4.49)とまったく同様に,

$$\boldsymbol{\phi}_k | \mathbf{z} \sim \mathrm{Dir}(\beta + m(k,1), \cdots, \beta + m(k,V)) \tag{5.47}$$

となります. $\mathbf{z}$ をサンプリングした後で, $\boldsymbol{\mu}$ と $\boldsymbol{\phi}$ を式(5.45)および式(5.47)からサンプリングし, これを繰り返せば, 正しい事後分布からサンプリングを行って学習することができます.

これでもよいのですが, 4.4.2 節で学習したように, **周辺化 Gibbs サンプリング**を行えば $\boldsymbol{\mu}$ および $\boldsymbol{\phi}$ を積分消去し, より効率的に $\mathbf{z}$ だけをサンプリングして学習することができます. いま, n 番目の文書 d_n のトピック z_n をサンプリングしたいとしましょう. 周辺化 Gibbs サンプリングでは, $\mathbf{z}$ をまずランダムに初期化し, n 番目以外の文書のトピック $\mathbf{z}_{-n} = z_1, \cdots, z_{n-1}, z_{n+1}, \cdots, z_N$ がわかっているとしたときに, z_n を事後分布 $p(z_n | d_n, \mathbf{z}_{-n})$ からサンプリングします. $\mathbf{z}_{-n}$

表 5.7: UM のベイズ学習で得られたトピックの特徴語 (NPMI). ここではトピック数 $K=100$ として学習しました. EM アルゴリズムによる学習と比べて, 局所解に陥らず, はるかによいトピックが学習されていることがわかります.

Topic 1		Topic 10		Topic 20		Topic 30	
アキレウス	0.643	南極	0.686	党	0.431	オリンピック	0.663
アステロパイオス	0.638	度	0.651	議員	0.387	団	0.635
ヒッポトオーン	0.633	西経	0.648	民主	0.385	北京	0.583
母音	0.619	北極	0.647	選挙	0.380	アテネ	0.582
ギリシア	0.615	東経	0.645	内閣	0.375	ロンドン	0.581
トロイア	0.607	線	0.623	大臣	0.368	バルセロナ	0.573
省略	0.598	成す	0.621	政治	0.365	結果	0.530
神話	0.574	角度	0.617	衆議	0.356	名簿	0.523
ゼウス	0.560	点	0.591	議会	0.344	競技	0.521
槍	0.548	通過	0.585	政党	0.343	選手	0.519

がわかっているのですから, 式(5.45)および式(5.47)において, 文書 d_n の分を除いた頻度を $m'(k,v)$ および $n'(k)$ とおけば, それぞれの事後分布は, 同様に

$$\begin{cases} \boldsymbol{\mu}\,|\,\mathbf{z}_{-n} \sim \mathrm{Dir}\,(\alpha+n'(1),\cdots,\alpha+n'(K)) \\ \boldsymbol{\phi}_k\,|\,\mathbf{z}_{-n} \sim \mathrm{Dir}\,(\beta+m'(k,1),\cdots,\beta+m'(k,V)) \end{cases} \tag{5.48}$$

となります. この期待値は, ディリクレ分布の期待値の公式(3.32)から

$$\begin{cases} \mathbb{E}[\mu_k\,|\,\mathbf{z}_{-n}] \;= \dfrac{\alpha+n'(k)}{\sum_{k=1}^{K}(\alpha+n'(k))} & \big(\,\mu_k = p(z_n=k)\,\big) \\ \mathbb{E}[\phi_{kv}\,|\,\mathbf{z}_{-n}] = \dfrac{\beta+m'(k,v)}{\sum_{v=1}^{V}(\beta+m'(k,v))} & \big(\,\phi_{kv} = p(v\,|\,z_n=k)\,\big) \end{cases} \tag{5.49}$$

となります. 式(5.49)を式(5.14)の $p(z_n\,|\,d_n)$ の計算に用いれば, 文書 d_n のトピック z_n をサンプリングすることができます. これを, すべての $n=1,\cdots,N$ について繰り返します. 以上を行う UM のベイズ学習 (周辺化 Gibbs サンプリング) のアルゴリズムを, 図 5.12 に示しました.

式(5.49)で期待値をとっているため, パラメータ $\theta=\{\,p(z),p(v\,|\,z)\,\}$ の点推定値はもはや存在しないことに注意してください. $\mathbf{z}$ だけをサンプリングすれば, θ の分布は式(5.47)および式(5.45)の形でディリクレ事後分布として表されますので, その期待値を式(5.49)のように求めることができます. この方法は, パ

ラメータの事後分布からサンプリングを行う完全な MCMC 法となっているため, 局所解の危険がなく, 理論的には十分長くサンプリングを行えば, 図 4.20 のように真の最適解 (を含む事後分布全体) からのサンプルを得ることができます.

表 5.7 に, サポートサイトにある UM のベイズ学習の実装 `bum.py` を用いて, 同じ日本語 Wikipedia コーパスについて $K=100$ のトピック数で学習した結果を示しました. ここでは, 100 回の Gibbs サンプリング (MCMC) の繰り返しでモデルを学習しています. EM アルゴリズムと比べて, さまざまな可能性を確率的に試すことで, 意味的にはるかにまとまりの高い潜在トピックが学習されていることがわかります.

5.3 ディリクレ混合モデル (DM)

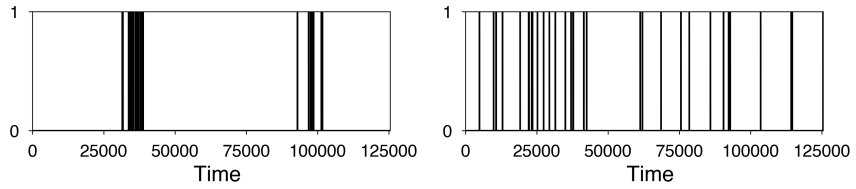
ここまで, テキストの単語が式(5.4)のように多項分布に従って発生すると仮定してきました. この多項モデルでは, 確率 $1/100=0.01$ の単語が 2 回発生する確率は $(1/100)^2=1/10000=0.0001$ になります. しかし実際には, これはあまり正しくありません. 図 5.13 に, 『シャーロック・ホームズの冒険』の本文の中で特定の単語が出現した場所を示しました. これを見ると, “lestrade” (レストレード警部) は一度出現すると続けて何度も出現しており, これはそんなに低い確率ではなさそうです. 言葉の出現のこうした性質を, **バースト性**といいます. ただし, バースト性の度合いは単語によって異なり, 同じ頻度 (=確率) の “interest” は, ほぼ一様に出現していることがわかります. [*23]

バースト性の度合いは, 次のようにしても調べることができます[143]. いま, コーパスの各文書を前半と後半に分け, 2 章で行ったように前半を ‘train’ データ, 後半を ‘test’ データとよぶことにしましょう. 表 5.8 に, Livedoor コーパスの 7367 文書の中で, “ソフトバンク” が前半と後半に 1 回以上出現した文書の数をまとめました.[*24] a は両方出現した, b は前半のみ出現し後半にはなかった, c は前半にはなかったが後半に出現した, d は前半と後半どちらにも出現しなかった文書の数をそれぞれ表しています.

---

*23 バースト性のモデル化について.

*24 これは, サポートサイトにある `recur.py` で調べることができます.



(a) “lestrade” (b) “interest”

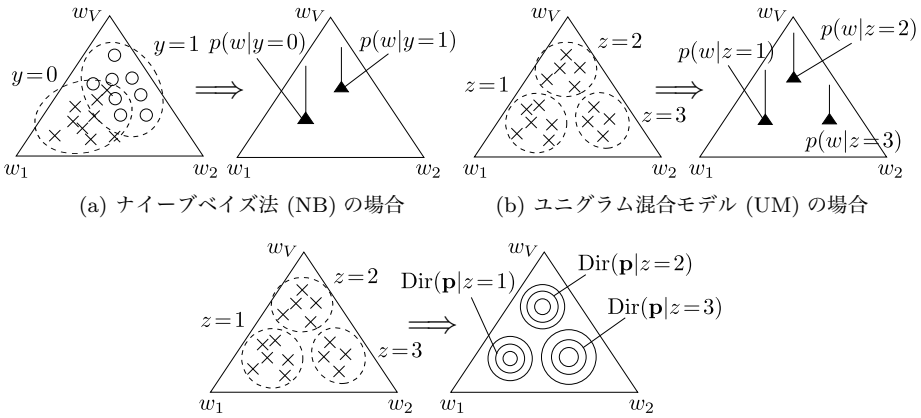
図 5.13: 『シャーロック・ホームズの冒険』での “lestrade” と “interest” の出現位置. この 2 つの単語の出現頻度はどちらも同じ 38 回ですが, “lestrade” (レストレード警部) のような言葉は, テキストの中で一部にバースト的に出現しています.

表 5.8: Livedoor コーパスの中で「ソフトバンク」が出現した文書の数. x はそこで単語が出現したことを, $\overline{x}$ は出現しなかったことを表します.

	test	$\overline{\text{test}}$
train	$a = 108$	$b = 93$
$\overline{\text{train}}$	$c = 93$	$d = 7073$

“ソフトバンク” の文書レベルでの出現確率は $(a+c)/(a+b+c+d) = 0.027$ ですので, 前半に出現した中 $(a+b)$ で, 後半にも出現する a の確率は, 単語の出現が独立ならば 0.03 程度になるはずです. しかし, この場合 $a/(a+b) = 0.537$ となっており, これは非常に高い確率です. つまり, “ソフトバンク” は 1 回出現すれば, 同じ文書でもう 1 回以上出現する確率は 0.537 もあるということです. これは明らかに, 単語のバースト性を示しています. [143]で (当時) ベル研究所の Church は, 出現確率 p の低い “Noriega” を例にとって 「“Noriega” が 2 回出現する (Two Noriegas) 確率は, p^2 ではなく $p/2$ だ」という言葉で, この現象を経験的に指摘しました. われわれの例でも, “ソフトバンク” が 2 回出現する確率は確かに, $p \cdot 0.537 \simeq p/2$ になっていることがわかります.

もちろん, バースト性は単語によって異なり, たとえば “順調” では $a/(a+b)$ は 0.065 と低い確率でした. テキストの確率モデルは, こうしたバースト性を表現できると望ましいでしょう.



(a) ナイーブベイズ法 (NB) の場合　(b) ユニグラム混合モデル (UM) の場合

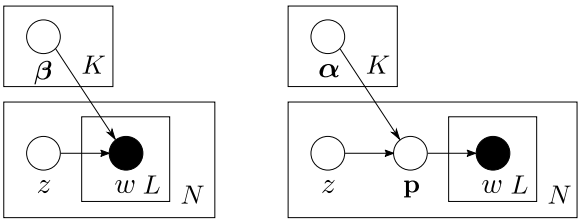
(c) ディリクレ混合モデル (DM) の場合

図 5.14: 文書モデルの幾何的表現. すべての多項分布は, $w_1 \cdots w_V$ を頂点とする単語単体の中に存在しています. NB や UM では, それらをトピックの代表点 (▲) でに針の立った δ 関数で表しますが, DM ではそれぞれのトピックをディリクレ分布で表現します.

5.3.1 単語単体と幾何的解釈

ここで, 少し違う視点からナイーブベイズ法と UM について振り返ってみましょう. 5.1 節の単語集合の仮定から, 各文書には含まれる単語を生成した V 次元の確率分布 $\mathbf{p} = (p_1, p_2, \cdots, p_V)$ があることになります. 3.4.1 節でみたように, この $\mathbf{p}$ は図 5.14 の各図の左側のような, 各単語 $w_1, w_2, \cdots, w_V$ を頂点とする V 次元の単体の中に存在しています. これを**単語単体**といいます. ナイーブベイズ法や UM は, この単語単体上に点として, トピック分布 $\mathbf{p} = \{p(w|y)\}$ や $\mathbf{p} = \{p(w|z)\}$ を図 5.14(a) や (b) のように学習するものだといえるでしょう. ナイーブベイズ法は UM の教師あり版ですから, 以下では UM に統一して考えていくことにします.

しかし, よく考えると UM のモデルは, 制限が強すぎるのではないでしょうか. 文書を生成したもともとの $\mathbf{p}$ には様々な可能性があるにもかかわらず, UM では, トピック z ごとに特定の $\mathbf{p} = \{p(w|z)\}$ で代表してしまっています. 「単語の出やすさ」$\mathbf{p}$ のタイプが完全に固定されてしまっているので, むしろ, 図 5.14(c) のようにトピックごとに**分布**を考え, この分布から $\mathbf{p}$ が生まれたと考える方が



(a) ユニグラム混合モデル (UM)　(b) ディリクレ混合モデル (DM)

図 5.15: UM と DM のグラフィカルモデル. ●は観測値, ○は確率変数を表します. 四角いプレートは繰り返しを, 右隅は繰り返しの数を表しています. DM では, 文書ごとに異なる $\mathbf{p}$ があるのが特徴です. 実際には, この $\mathbf{p}$ は積分消去することができます.

適切ではないでしょうか. $\mathbf{p}$ 自体が多項分布なので, これは**確率分布の確率分布**となります.

この分布として最も簡単なのは, 3.4.1 節のディリクレ分布を考えることでしょう. 図 5.14(c) のように, 単語単体上にトピックを表現する複数のディリクレ分布を考えるこのモデルを, **ディリクレ混合モデル** (Dirichlet Mixtures, **DM**) といいます.[*25] DM は最初, バイオインフォマティクスの分野でアミノ酸配列の解析のために 1996 年に提案されました[144]. 筑波大学の山本らは 2003 年に, これがテキストにも適用でき, 5.4 節で説明する LDA よりも高い文書確率を与えることを示しました[145].[*26]

ディリクレ混合モデルの生成モデル　UM では, 文書 $d = w_1 w_2 \cdots w_L$ は次のようにして生成されたと考えました.

1. $z \sim \boldsymbol{\lambda}$ でトピック z を選択.
2. For $i = 1, \cdots, L$,
   a. $w_i \sim p(w|z)$ から単語 w_t を生成.

すなわち, 各単語はトピック z で決まるユニグラム分布 $\boldsymbol{\beta}_z = \{p(w|z)\}$ から生成されると考えています. これをグラフィカルモデルで表すと, 図 5.15(a) のようになります. これに対して, DM では文書は

---

*25　これは, 通常のユークリッド空間において K 平均法の代わりに, ガウス混合モデルを考えることと似ています. ディリクレ分布は, 単体上のガウス分布のようなものと考えてよいでしょう.

*26　東京大学の佐藤らの研究[146]を用いれば, Pitman-Yor 過程を用いる洗練された方法で DM とこの後の LDA は統合でき, さらに高い文書確率を与えることが確かめられています.

アルゴリズム 5: DM の EM-Newton アルゴリズム.

```
λ, α_1, ..., α_K を適当に初期化する
while 収束するまで do
    for n in 1, ..., N do                    (* E ステップ *)
        式(5.51) で p(z|d_n) を計算
    式(5.52) で λ, α_1, ..., α_K を更新       (* M ステップ *)
```

図 5.16: DM の EM アルゴリズムによる学習.

1. $z \sim \boldsymbol{\lambda}$ でトピック z を選択.
2. $\mathbf{p} \sim \mathrm{Dir}(\boldsymbol{\alpha}_z)$ でユニグラム分布 $\mathbf{p}$ を生成.
3. For $i = 1, \cdots, L$,
   a. $w_i \sim \mathbf{p}$ で単語 w_t を生成.

のようにして生成されたと考えます. グラフィカルモデルで表すと, 図 5.15(b) のようになります. UM では決まったユニグラム分布 $\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ の中から選ぶのに対し, DM では文書ごとに異なるユニグラム分布 $\mathbf{p}$ が, トピックに従って毎回生成されるのが特徴です. これにより, DM ではたとえば同じ車の話題を扱う文書でも, 「トヨタ」が多い場合と「日産」が多い場合で異なる $\mathbf{p}$ が使われていると考えて区別することができます. 後で数学的に示すように, このことから単語のバースト性を説明することができます.

幸いにして $\mathbf{p}$ は直接求める必要はなく, 3 章で説明したように, 期待値をとって積分消去することができます. いま, 文書 d のトピックが $z=k$ とわかっていたとしましょう. すると文書の確率は, $\mathbf{p}$ が k 番目のディリクレ分布 $\mathrm{Dir}(\boldsymbol{\alpha}_k)$ $(\boldsymbol{\alpha}_k = (\alpha_{k1}, \alpha_{k2}, \cdots, \alpha_{kV}))$ から生成されたということなので, 期待値をとれば

$$
\begin{aligned}
(5.50) \qquad p(d|z=k) &= \int p(d, \mathbf{p}|z=k)d\mathbf{p} = \int p(d|\mathbf{p})p(\mathbf{p}|z=k)d\mathbf{p} \\
&= \int \prod_{i=1}^{L} \prod_{v=1}^{V} p_v^{\mathbb{I}(w_i=v)} \cdot \mathrm{Dir}(\mathbf{p}|\boldsymbol{\alpha}_k)d\mathbf{p} \\
&= \int \prod_{v=1}^{V} p_v^{n_v} \cdot \frac{\Gamma(\sum_v \alpha_{kv})}{\prod_v \Gamma(\alpha_{kv})} \prod_{v=1}^{V} p_v^{\alpha_{kv}-1} d\mathbf{p} \\
&= \frac{\Gamma(\sum_v \alpha_{kv})}{\Gamma(\sum_v \alpha_{kv} + L)} \prod_{v=1}^{V} \frac{\Gamma(\alpha_{kv} + n_v)}{\Gamma(\alpha_{kv})}
\end{aligned}
$$

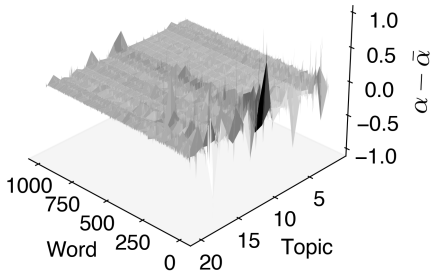


図 5.17: Livedoor コーパスで学習された DM のパラメータ $\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_K$ の様子. 全体の平均 $\overline{\boldsymbol{\alpha}}$ との差分をプロットしています. (単語 1〜1000, トピック数 $K=20$)

となり, 式(3.58)の**ポリア分布**で表すことができます. ここで, 1 行目では同時確率の周辺化 (公式(2.10)) および確率の連鎖則 (公式(2.20)) を用いました. $n_v = \sum_{i=1}^{L} \mathbb{I}(w_i = v)$ は, d の中で単語 v が何回出たかを表します. 実際には z は未知ですから, DM での文書の確率は, z についても期待値をとり,

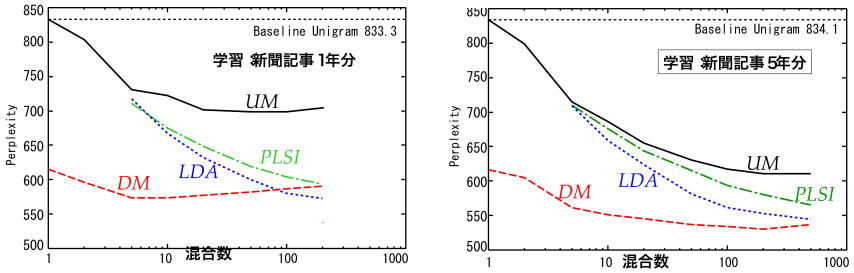
$$
\begin{aligned}
(5.51)\quad p(d) &= \sum_{k=1}^{K} p(d, z=k) = \sum_{k=1}^{K} p(d \,|\, z=k) p(z=k) \\
&= \sum_{k=1}^{K} \lambda_k \frac{\Gamma(\sum_v \alpha_{kv})}{\Gamma(\sum_v \alpha_{kv} + L)} \prod_{v=1}^{V} \frac{\Gamma(\alpha_{kv} + n_v)}{\Gamma(\alpha_{kv})} \quad (\textbf{DM の文書確率})
\end{aligned}
$$

となります.[*27]

ディリクレ混合モデルの学習　DM のパラメータは, $p(z)$ を表す $\boldsymbol{\lambda} = \{\lambda_1, \cdots, \lambda_K\}$ およびトピック別のディリクレ分布のハイパーパラメータ $\{\alpha_{kv} \,|\, k = 1, \cdots, K, v = 1, \cdots, V\}$ です. 他のトピックモデルも同様ですが, 一般に $K = 100$, 語彙の大きさ $V = 10000$ と少なめのときでも, α_{kv} は 100 万個程度のパラメータ数となることに注意してください. α_{kv} は EM アルゴリズムの中で式(3.60)のニュートン法で最適化することもできますが, これは特殊関数であるダイガンマ関数 $\Psi(x) = d/dx \log \Gamma(x)$ が含まれているため, テキストデータについては非常に重い計算になります.

---

*27　すなわち, 「ディリクレ分布の混合モデル」となっているのは $\mathbf{p}$ に対してのもので, $\mathbf{p}$ を積分消去すると, DM は観測値については「混合ポリア分布」になっています.



(a) 学習データが少ない場合 (1 年分) (b) 学習データが多い場合 (5 年分)

図 5.18: 各トピックモデルの性能 ([147]から引用). 縦軸はテストデータに対するパープレキシティです. いずれの場合も, 単語のバースト性を捉える DM の性能が最も高くなっており, 混合数を増やすと LDA の性能と近づいてくることがわかります.

山本ら[145]はこれに代わり, [67]で示された高速な LOO (Leave-One-Out) 近似を使って, 図 5.16 のアルゴリズムおよび次式で DM のパラメータを高速かつ安定に学習できることを示しました.

$$\text{(5.52)} \qquad \begin{cases} \lambda_k \propto \sum_{n=1}^{N} p(z_n = k \,|\, d_n) \\ \alpha'_{kv} = \alpha_{kv} \cdot \dfrac{\sum_{i=1}^{N} p(z_i = k \,|\, d_i)\, n_{iv} / (n_{iv} - 1 + \alpha_{kv})}{\sum_{i=1}^{N} p(z_i = k \,|\, d_i)\, n_i / (n_i - 1 + \sum_v \alpha_{kv})} \end{cases}$$

ここで n_{iv} は文書 i における単語 v の頻度, $n_i = \sum_v n_{iv}$ は文書 i の長さです. この式は特殊関数を含まないため計算が速く, 筆者の公開している C 言語による実装 [*28] を用いた場合, Livedoor コーパスで $K = 100$ の場合は, 執筆時の環境では 3 分程度で EM アルゴリズムが収束します. Python では非常に遅くなるため, 注意してください. なお, この推定法は最尤推定のため過学習しやすく, 山本らはさらに, 階層ベイズの考え方で推定を安定化する Smoothed DM と, 変分ベイズ法の一種による解法を示し[148], こちらの方が安定して高性能であることが報告されています.[*29] 図 5.18 に, UM, DM および次節で説明する LDA を Livedoor コーパスで学習した場合の, テストデータのパープレキシティをト

*28 `http://chasen.org/~daiti-m/dist/dm/`にある `dm-0.2.tar.gz` で C 言語による実装を公開しています.

*29 `http://chasen.org/~daiti-m/dist/dm/`にある `sdm-0.2.tar.gz` で C 言語による実装を公開しています.

ピック数 K ごとに示しました. これからわかるように, パープレキシティ(文書確率) の面では, DM は LDA よりも優れたモデルです. ただしこれは主に, 文書内での単語のバースト性のモデル化から生じるもので, LDA の方が柔軟なモデルであることに注意してください.

なお, DM の各トピックを表すディリクレ分布 $\mathrm{Dir}(\boldsymbol{\alpha}_k)$ の期待値は, 公式(3.32)で示したように $\boldsymbol{\alpha}_k$ の和を 1 に正規化すると求めることができ, これは UM におけるトピック分布 $p(w|z)$ と等価だとみなすことができます. DM ではどのようなトピックが推定されているか, 5.2.1 節の UM の場合と同様にして調べてみると面白いでしょう. (→演習 5)

5.3.2　ポリア分布と単語のバースト性

いま, 文書を生成したディリクレ分布 $\mathrm{Dir}(\boldsymbol{\alpha})$ のパラメータを $\boldsymbol{\alpha}=(\alpha_1,\cdots,\alpha_V)$ とすると, 式(5.51) に示したように, 文書 d の確率はポリア分布

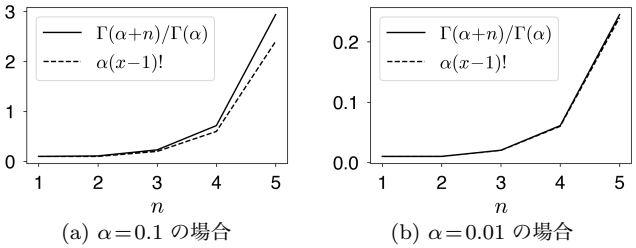
$$p(d)=\frac{L!}{\prod_{v\in d} n_v!}\frac{\Gamma(\sum_v \alpha_v)}{\Gamma(\sum_v \alpha_v+L)}\prod_{v\in d}\frac{\Gamma(\alpha_v+n_v)}{\Gamma(\alpha_v)} \tag{5.53}$$

となります. 長さ L の文書で各単語の頻度が n_v になる組み合わせの数は, 多項係数 $L!/\prod_v n_v!$ だけありますから, 上では明示的に加えました.

ここで, 3.4.4 節の図 3.17 で述べたように, 言語の場合は α_v は非常に小さく, ほとんどの場合は 0.01 あるいは 0.001 未満の値となることを思い出してください. このとき, 式(5.53)で各単語に依存するファクター $\Gamma(\alpha_v+n_v)/\Gamma(\alpha_v)$ は, 式(3.39) の Γ 関数の性質から

$$\begin{aligned}\frac{\Gamma(\alpha+n)}{\Gamma(\alpha)} &= \frac{(n+\alpha-1)(n+\alpha-2)\cdots\alpha\cancel{\Gamma(\alpha)}}{\cancel{\Gamma(\alpha)}}\\ &= \alpha(\alpha+1)\cdots(n+\alpha-1)\\ &\simeq \alpha(n-1)! \qquad (\because \alpha \ll 1)\end{aligned} \tag{5.54}$$

と近似することができます. 図 5.19 に, この近似が実際に非常に正確であることを示しました. よって, この近似を式(5.53)に代入すれば



(a) $\alpha=0.1$ の場合 (b) $\alpha=0.01$ の場合

図 5.19: Pochhammer 関数 $\Gamma(\alpha+n)/\Gamma(\alpha)$ の近似. α が小さい場合には, この値は近似 $\alpha(x-1)!$ と, ほとんど差がないことがわかります.

$$p(d) \simeq \frac{L!}{\prod_v n_v!}\frac{\Gamma(\sum_v \alpha_v)}{\Gamma(\sum_v \alpha_v + L)}\prod_{v\in d}\alpha_v(n_v-1)! \tag{5.55}$$

$$= L!\,\frac{\Gamma(\sum_v \alpha_v)}{\Gamma(\sum_v \alpha_v + L)}\prod_{v\in d}\frac{\alpha_v}{n_v} \qquad (\textbf{指数分布族 DCM 分布})$$

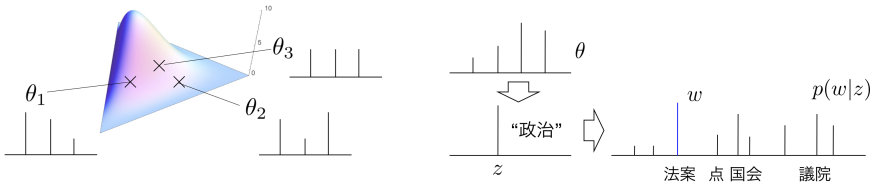
となります. n_v の項をくくり出せば, これは統計学で標準的な指数分布族の形で表すことができ, Elkan [149] はこれを指数分布族 DCM (EDCM) 分布と呼んでいます.

n	1	2	3	⋯
p	α_v	$\frac{\alpha_v}{2}$	$\frac{\alpha_v}{3}$	⋯

式(5.55)で, 文書中の単語 v に依存するファクターは α_v/n_v になっており, これは右の表のように, $n_v=1$ のときは確率が α_v 倍, $n_v=2$ のときは $\alpha_v/2$ 倍, ⋯ となることを意味しています. つまり, (頻度の低い) 単語 v が 2 回出現しても, その確率は α_v^2 ではなく, $\alpha_v/2$ 倍にしかならないということです. これはまさに, 251 ページの “Two Noriegas” の発見そのものです. 単語が $n+1$ 回現れる確率は, n 回現れる確率の $n/(n+1)$ 倍になっており, これから単語は一度現れれば, バースト的に出現することになります. ポリア分布を使うことで, われわれは実際にこの現象が成り立つことを数学的に説明することができました.

5.4 潜在ディリクレ配分法 (LDA)

これまで, UM や DM では各文書が 1 つのトピックから生成されたと仮定してきましたが, これには明らかに限界があると考えられます. たとえば, 「劇場



(a) ディリクレ分布 $\mathrm{Dir}(\boldsymbol{\alpha})$ から生成される各文書のトピック分布 $\boldsymbol{\theta}$.

(b) 1 単語ごとに $\boldsymbol{\theta}$ からトピック z を選び, z ごとのトピック–単語分布 $p(w|z)$ から単語 w が生成されます.

図 5.20: LDA の文書生成モデル. LDA では, 文書の各単語ごとに異なる潜在トピック z が存在します.

の改修」を伝えるニュースでは, 演劇についての言葉と建築についての言葉が両方現れるでしょう. また, 数理経済学の論文では, 数学の話と経済学の話が混じっていると考えられます. これらを単一のトピックで表現しようとすると, すべての組み合わせに別々のトピックを用意しなければならなくなってしまいます. 組み合わせは 2 つとは限りませんから, これには厖大なトピック数が必要になります. また, 「数理経済学」と「開発経済学」のトピックが経済学という話題を共有していることもわからなくなってしまうでしょう.

よって, 各文書にトピックが 1 つだけあるのではなく, 図 5.20(a) のようにトピックの**確率分布** $\boldsymbol{\theta}$ があると考えた方がよいでしょう. たとえばトピックが $K=4$ 種類あり, それぞれ「演劇」「経済」「建築」「数学」だったとき, 上の劇場の改修のニュースは $\boldsymbol{\theta}=(0.7,0,0.3,0)$, 数理経済学の論文は $\boldsymbol{\theta}=(0,0.4,0,0.6)$ のようになると考えられます. 数学的には, 各文書 d についてトピック z の確率分布 $\boldsymbol{\theta}=\{p(z|d)\}$ を考えることになります. UM や DM は, これをあるトピック k についてだけ 1 をとるデルタ関数 $p(z|d)=\delta(z=k)$ で近似してしまうモデルとみなすことができます [*30].

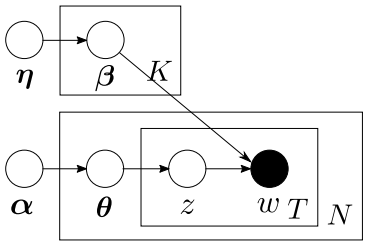
この $\boldsymbol{\theta}$ は文書ごとに異なり, 最も簡単には 3 章で学習したディリクレ分布

$$\boldsymbol{\theta} \sim \mathrm{Dir}(\boldsymbol{\alpha}) \quad (\boldsymbol{\alpha}=(\alpha_1,\cdots,\alpha_K)) \tag{5.56}$$

*30　なお, 単一トピックの UM や DM でも, 学習の際には $\delta(z=k)$ の推定値として図 5.9 のように $p(z|d)$ を考えることになりますが, 真の値はあくまで単一のトピックで, それを確率的に推定しているにすぎません. それに対して, ここでは真の値 $p(z|d)$ 自体が多項分布であり, その推定値として, 多項分布の分布であるディリクレ分布が事後分布として得られることになります.

- For $k = 1, \cdots, K$,
  - Draw $\boldsymbol{\beta}_k \sim \mathrm{Dir}(\boldsymbol{\eta})$.
- For $i = 1, \cdots, N$,
  - Draw $\boldsymbol{\theta}_i \sim \mathrm{Dir}(\boldsymbol{\alpha})$.
  - For $t = 1, \cdots, T$,
    - Draw $z_{it} \sim \boldsymbol{\theta}_i$.
    - Draw $w_{it} \sim \boldsymbol{\beta}_{z_{it}}$.

(a) LDA の生成モデル



(b) LDA のグラフィカルモデル

図 5.21: LDA の生成モデルと, 対応するグラフィカルモデル. ●は観測された変数を, ○は潜在変数を表しています.

から生成されたと考えるといいでしょう. 図 5.20(b) に示したように, この $\boldsymbol{\theta}$ から, 文書の各単語ごとに

(1) まずトピック $z \sim \boldsymbol{\theta}$ をランダムに選び,
(2) z ごとのトピック–単語分布 $p(w|z)$ から単語 w を生成する

ことを繰り返して, 文書が生成されたとするモデルを考えることができます. このトピック–単語分布 $p(w|z)$ も, z ごとにディリクレ事前分布 $\mathrm{Dir}(\boldsymbol{\eta})$ から生成されたとしましょう. 以上の生成モデルを, アルゴリズムの形で図 5.21(a) に示しました. 学習の際には, 文書ごとに潜在変数 $\boldsymbol{\theta}$ のディリクレ事後分布を割り当てて推定するため, このモデルを**潜在ディリクレ配分法** (Latent Dirichlet Allocation, **LDA**) [137]といいます. LDA は後で説明する PLSI のベイズ化として, Blei らにより 2001 年の論文[150]で最初に提案されました.

LDA のグラフィカルモデルを図 5.21(b) に示しました. 図 5.15 の UM のグラフィカルモデルと比べると, z が内側の箱の中に入っており, 文書に含まれる 1 つ 1 つの単語 w ごとに潜在トピック z があることを示しています. 100 万単語のコーパスがあれば, 100 万個の z があるわけです. よって LDA は, UM よりずっと複雑, かつ柔軟な確率モデルになっています.

5.4.1 Gibbs サンプリングによる学習

LDA の学習法には (1) 最初に提案された変分ベイズ法[137], (2) 周辺化の考え方を取り入れた周辺化変分ベイズ法[151], (3) Gibbs サンプリング[152]

[153] などがあります [*31]. この中で, 5.2 節で説明した EM アルゴリズムのベイズ版である変分ベイズ法は数学的導出の難しさにもかかわらず, EM アルゴリズムの一種のため局所解の問題を逃れることができず, 図 5.22 に示したように, Gibbs サンプリング (MCMC 法) による学習が最終的に, 最も性能が高いことがわかっています. 導出や実装も容易なため, 本書では Gibbs サンプリングによる学習について説明します. 変分ベイズ法による学習について知りたい方は, 教科書[155]を参照してください.

上の生成モデルによれば, LDA で文書 $\mathbf{w}=w_1w_2\cdots w_T$ と背後にある潜在変数

(1) 各単語の潜在トピック $\mathbf{z}=z_1z_2\cdots z_T$,
(2) 文書の潜在トピック分布 $\boldsymbol{\theta}$,
(3) トピック–単語分布 $\boldsymbol{\beta}=\{\beta_{kv}=p(v|k)\}$

の同時確率は,

$$p(\mathbf{w},\mathbf{z},\theta,\boldsymbol{\beta}|\boldsymbol{\alpha},\boldsymbol{\eta}) = p(\mathbf{w}|\mathbf{z},\boldsymbol{\beta})\,p(\mathbf{z}|\theta)\,p(\theta|\boldsymbol{\alpha})\,p(\boldsymbol{\beta}|\boldsymbol{\eta}) \tag{5.57}$$

と分解することができます. 具体的には, 199 ページで導入した指示関数 $\mathbb{I}()$ による記法を使うと,

$$\begin{aligned} &p(\mathbf{w}|\mathbf{z},\boldsymbol{\beta})\,p(\mathbf{z}|\boldsymbol{\theta})\,p(\boldsymbol{\theta}|\boldsymbol{\alpha})\,p(\boldsymbol{\beta}|\boldsymbol{\eta}) \\ &= \left(\prod_{t=1}^{T} p(w_t|z_t,\boldsymbol{\beta})p(z_t|\boldsymbol{\theta})\right)\cdot \mathrm{Dir}\,(\boldsymbol{\theta}|\boldsymbol{\alpha})\cdot\prod_{k=1}^{K}\mathrm{Dir}\,(\boldsymbol{\beta}_k|\boldsymbol{\eta}) \\ &\propto \underbrace{\prod_{t=1}^{T}\prod_{v=1}^{V}\prod_{k=1}^{K}\beta_{kv}^{\mathbb{I}(w_t=v)\,\mathbb{I}(z_t=k)}}_{=\beta_{z_tw_t}}\cdot\underbrace{\prod_{t=1}^{T}\prod_{k=1}^{K}\theta_k^{\mathbb{I}(z_t=k)}}_{=\theta_{z_t}}\cdot\prod_{k=1}^{K}\theta_k^{\alpha_k-1}\cdot\prod_{k=1}^{K}\prod_{v=1}^{V}\beta_{kv}^{\eta-1} \end{aligned} \tag{5.58}$$

(**LDA の文書同時確率**)

と表すことができます. 4 章 (199 ページ) で説明したように, パラメータ $\beta_{kv}=p(v|k)$ を表に出すために, $p(w_t|z_t,\boldsymbol{\beta})=\beta_{kv}$ (ただし $k=z_t$, $v=w_t$) を指示関数 $\mathbb{I}()$ を使って, 「ただし」の部分を数学的に

---

*31　期待値伝搬法 (EP) による学習も提案されていますが[154], この場合はすべての単語についてそのトピック事後分布を保持しなければならないため, 学習がスケールしないことが問題です.

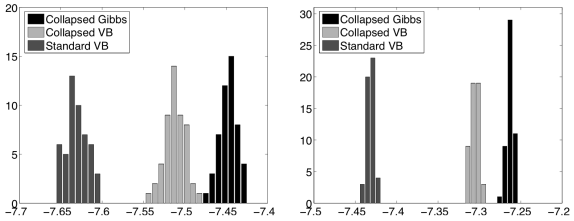


図 5.22: LDA の学習アルゴリズムの違いによる性能の比較 ([151]より引用). 左側は DailyKOS のニュース記事, 右側は NIPS の論文コーパスの結果で, 横軸はテストデータの 1 単語あたりの対数尤度を表します. いずれも, 変分ベイズ法に比べて周辺化 Gibbs サンプリング (黒棒) がもっとも高い対数尤度を与えていることがわかります.

$$p(w_t \,|\, z_t, \boldsymbol{\beta}) = \prod_{v=1}^{V} \prod_{k=1}^{K} \beta_{kv}^{\mathbb{I}(z_t=k)\mathbb{I}(w_t=v)} \tag{5.59}$$

と表していることに注意してください. θ_k についても同様です.

実際には, 文書は $\mathbf{w}_1^N = \mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_N$ の N 個ありますから, データ全体の同時確率は, 対応する潜在トピックを $\mathbf{z}_1^N = \mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_N$, 各文書のトピック分布を $\boldsymbol{\theta}_1^N = \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_N$ として

$$\begin{aligned} &p(\mathbf{w}_1^N, \mathbf{z}_1^N, \boldsymbol{\theta}_1^N, \boldsymbol{\beta} \,|\, \boldsymbol{\alpha}, \boldsymbol{\eta}) \\ &\propto \prod_{i=1}^{N} \left[\prod_{t=1}^{T} \prod_{v=1}^{V} \prod_{k=1}^{K} \beta_{kv}^{\mathbb{I}(w_{it}=v)\,\mathbb{I}(z_{it}=k)} \cdot \prod_{t=1}^{T} \prod_{k=1}^{K} \theta_{nk}^{\mathbb{I}(z_{it}=k)} \prod_{k=1}^{K} \theta_{nk}^{\alpha_k - 1} \right] \prod_{k=1}^{K} \prod_{v=1}^{V} \beta_{kv}^{\eta - 1} \end{aligned} \tag{5.60}$$

と表されます. 記法を簡単にするために, 文書の長さ T を上では同じにしていますが, 実際は文書によって異なる値であることに注意してください.

LDA の Gibbs サンプリング

このうち, わかっているのは観測された文書 $\mathbf{w}$ だけで, ほかの潜在変数はすべて未知の潜在変数で, $\mathbf{w}$ から推定します. LDA のパラメータ $\mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\beta}$ は, 4 章で学んだ Gibbs サンプリングを使って, 順番にサンプリングすることを繰り返せば推定することができます.

$\mathbf{z}$ のサンプリング 文書 n の t 番目の単語 w_{it} のトピックを表す潜在変数 z_{it} は, 式(5.57)の 1 行目で z に関連する項は $p(\mathbf{w}|\mathbf{z}, \boldsymbol{\beta})p(\mathbf{z}|\theta)$ だけですから, 式(5.60)

から対応する項を抜き出せば,

$$p(z_{it}|\mathbf{z}_{-it},\boldsymbol{\theta},\boldsymbol{\beta}) \propto p(w_{it}|z_{it},\boldsymbol{\beta})p(z_{it}|\mathbf{z}_{-it},\boldsymbol{\theta}_i) = \beta_{kv}\cdot\theta_{ik} \quad (v=w_{it}) \tag{5.61}$$

となります. これは, 意味的には現在の文書を $\mathbf{w}_i=d$, 単語を $w_{it}=v$, サンプリングしたい $z_{it}=k$ とおくと,

$$p(k|v,d) \propto p(v,k|d) = \underbrace{p(v|k)}_{\beta_{kv}}\underbrace{p(k|d)}_{\theta_{ik}} \tag{5.62}$$

を計算している, ということです. したがって, z_{it} は確率

$$p(z_{it}=k|\mathbf{z}_i\backslash z_{it},\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\theta_{ik}\beta_{kv}}{\sum_{k=1}^{K}\theta_{ik}\beta_{kv}} \tag{5.63}$$

に従ってサンプリングすればよいことがわかります.

$\boldsymbol{\theta}$ のサンプリング　同様に, 式(5.57)で θ について注目すると

$$p(\boldsymbol{\theta}|\mathbf{z},\boldsymbol{\alpha}) \propto p(\mathbf{z}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \tag{5.64}$$

ですから, 式(5.60)から対応する項を抜き出して

$$p(\boldsymbol{\theta}_i|\mathbf{w}_i,\mathbf{z}_i) \propto \prod_{k=1}^{K}\theta_{ik}^{\alpha_k-1}\cdot\prod_{t=1}^{T}\prod_{k=1}^{K}\theta_{nk}^{\mathbb{I}(z_{it}=k)} = \prod_{k=1}^{K}\theta_{nk}^{\alpha_k-1+\sum_{t=1}^{T}\mathbb{I}(z_{it}=k)} \tag{5.65}$$

となり, $n(i,k)=\alpha_k+\sum_{t=1}^{T}\mathbb{I}(z_{it}=k)$ とおけば, θ_i はディリクレ事後分布

$$p(\boldsymbol{\theta}_i|\mathbf{w}_i,\mathbf{z}_i) = \mathrm{Dir}(\alpha_1+n(i,1),\cdots,\alpha_K+n(i,K)) \tag{5.66}$$

からサンプリングすればよいことがわかります.

$\boldsymbol{\beta}$ のサンプリング　最後に, 最も重要なトピック–単語分布 $\boldsymbol{\beta}$ のサンプリングを考えましょう. 式(5.57)で $\boldsymbol{\beta}$ に対応する項は

$$p(\boldsymbol{\beta}|\mathbf{w},\mathbf{z},\boldsymbol{\theta},\boldsymbol{\alpha}) \propto p(\mathbf{w}|\mathbf{z},\boldsymbol{\beta})p(\boldsymbol{\beta}) \tag{5.67}$$

ですから, $\boldsymbol{\beta}$ がトピック k ごとにディリクレ事前分布

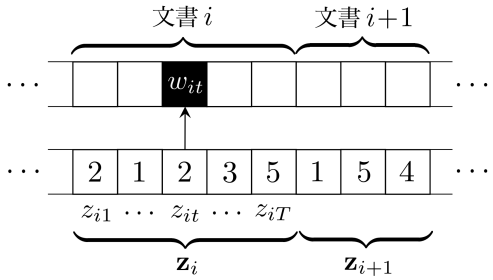


図 5.23: LDA の Gibbs サンプリングの様子. 各単語 w_{it} を生成した潜在トピック z_{it} を, 事後分布からランダムにサンプリングして更新することを繰り返します.

$$p(\boldsymbol{\beta}_k) = \mathrm{Dir}\,(\eta, \cdots, \eta) \propto \prod_{v=1}^{V} \beta_{kv}^{\eta-1} \tag{5.68}$$

に従うとすれば, $\boldsymbol{\beta}_k$ の事後分布は式(5.60)から

$$\begin{aligned} p(\boldsymbol{\beta}_k \,|\, \mathbf{w}_1^N, \mathbf{z}_1^N) &\propto \prod_{i=1}^{N} \prod_{t=1}^{T} \prod_{v=1}^{V} \beta_{kv}^{\mathbb{I}(w_{it}=v)\,\mathbb{I}(z_{it}=k)} \cdot \prod_{v=1}^{V} \beta_{kv}^{\eta-1} \\ &= \prod_{v=1}^{V} \beta_{kv}^{\eta-1+\sum_{i=1}^{N}\sum_{t=1}^{T}\mathbb{I}(w_{it}=v)\,\mathbb{I}(z_{it}=k)} \end{aligned} \tag{5.69}$$

となり, $m(k,v) = \eta + \sum_{i=1}^{N}\sum_{t=1}^{T}\mathbb{I}(w_{it}=v)\,\mathbb{I}(z_{it}=k)$ とおけば, こちらもディリクレ事後分布

$$p(\boldsymbol{\beta}_k \,|\, \mathbf{w}_1^N, \mathbf{z}_1^N) = \mathrm{Dir}\,(\eta + m(k,1), \cdots, \eta + m(k,V)) \tag{5.70}$$

からサンプリングすることができます.

式(5.63), 式(5.66), 式(5.70) に従って $\mathbf{z}, \boldsymbol{\theta}, \boldsymbol{\beta}$ を次々とサンプリングすることを繰り返せば, LDA のパラメータを求めることができます. 実は, データ量やトピック数は圧倒的に少ないものの, 本質的に同じモデルが LDA より 1 年前にバイオインフォマティクスの分野で Pritchard ら[156]によって提案されており, 推論にはこの方法が使われていました.

アルゴリズム 6: LDA の周辺化 Gibbs サンプリング

for i in $1, \cdots, N$ **do** (* 初期化 *)
  for t in $1, \cdots, T$ **do**
    $z_{it} = \text{randint}(K)$ (* $1 \cdots K$ の乱数で初期化 *)
    $n'(i, z_{it})$++
    $m'(w_{it}, z_{it})$++
while 収束するまで **do** (* Gibbs サンプリング *)
  for i in randperm(N) **do** (* 文書 i をランダムに選ぶ *)
    for t in $1, \cdots, T$ **do**
      $n'(i, z_{it})$−− (* カウントを減らす *)
      $m'(w_{it}, z_{it})$−−
      z_{it} を式(5.75)に従ってサンプリング
      $n'(i, z_{it})$++ (* カウントを増やす *)
      $m'(w_{it}, z_{it})$++
式(5.76)から $\boldsymbol{\theta}_1^N$, $\boldsymbol{\beta}$ の期待値を求める

図 5.24: 周辺化 Gibbs サンプリングによる LDA の学習アルゴリズム.

LDA の周辺化 Gibbs サンプリング

しかし, パラメータ数が桁違いに大きい自然言語の場合は, HMM (205 ページ) や UM (246 ページ) でもわれわれが行ったように, パラメータについて期待値をとり, 可能な限り積分消去することで, より効率的なサンプリングを行うことができます. 実際には, こちらの方法で学習を行うのがよいでしょう.

式(5.61)で z_{it} のサンプリングを行う際に, θ_{ik} が必要になります. ただし, z_{it} 以外の $\mathbf{z}_i \backslash z_{it}$ はわかっているのですから, $\boldsymbol{\theta}_i$ の事後分布は式(5.66)から, $\mathbf{z}_i \backslash z_{it}$ の中でトピック k に割り当てられた単語の数を $n'(i,k)$ とおくと,

$$p(\boldsymbol{\theta}_i \,|\, \mathbf{z}_i \backslash z_{it}) = \text{Dir}\,(\alpha_1 + n'(i,1), \cdots, \alpha_K + n'(i,K)) \tag{5.71}$$

です. よって, その期待値

$$\mathbb{E}[\theta_{ik} \,|\, \mathbf{z}_i \backslash z_{it}] = \frac{\alpha_k + n'(i,k)}{\sum_k (\alpha_k + n'(i,k))} \tag{5.72}$$

を使うことができるでしょう.

同様にして β_{kv} についても, その期待値で置き換えることができます. $\boldsymbol{\beta}_k$ の

事後分布は式(5.70)から,

$$p(\boldsymbol{\beta}_k \,|\, \mathbf{w}_1^N, \mathbf{z}_1^N \backslash z_{it}) = \mathrm{Dir}\,(\eta + m'(k,1), \cdots, \eta + c'(k,V)) \tag{5.73}$$

となることに注意しましょう. ここで $m'(k,v)$ は z_{it} を除いたすべての潜在変数 $\mathbf{z}_1^N \backslash z_{it}$ の中で, 単語 v にトピック k が割り当てられた回数を表しています. よって, 式(5.63)の β_{kv} の部分は

$$\mathbb{E}[\beta_{kv} \,|\, \mathbf{z}_1^N \backslash z_{it}] = \frac{\eta + m'(k,v)}{\sum_{v=1}^{V} (\eta + m'(k,v))} \tag{5.74}$$

となります.

以上より, 式(5.74)と式(5.72)から, z_{it} のサンプリングは式(5.63)に代えて, 次の式で行うことができます. [*32]

$$p(z_{it} = k \,|\, w_{it} = v, \mathbf{z}_i \backslash z_{it}) \propto \frac{\alpha_k + n'(i,k)}{\sum_{k=1}^{K} (\alpha_k + n'(i,k))} \cdot \frac{\eta + m'(k,v)}{\sum_{v=1}^{V} (\eta + m'(k,v))} \tag{5.75}$$

(**LDA の周辺化 Gibbs サンプリングの公式**)

この場合, $\boldsymbol{\theta}$ **や** $\boldsymbol{\beta}$ **のサンプリングは不要で,** $\mathbf{z}$ **だけをサンプルすれば学習できる**ことに注意してください. $\mathbf{z}$ のサンプリングが収束すれば, $\boldsymbol{\theta}$ と $\boldsymbol{\beta}$ は式(5.66)および式(5.70)の期待値をとることで,

$$\mathbb{E}[\theta_{nk} \,|\, \mathbf{z}_i] = \frac{\alpha_k + n(i,k)}{\sum_{k=1}^{K} (\alpha_k + n(i,k))}, \quad \mathbb{E}[\boldsymbol{\beta}_{kv} \,|\, \mathbf{z}_1^N] = \frac{\eta + m(k,v)}{\sum_{v=1}^{V} (\eta + m(k,v))} \tag{5.76}$$

と求めることができます. この学習アルゴリズムを図 5.24 に示しました.

周辺化尤度の計算 なお, この場合は周辺化 Gibbs サンプリングの目的関数は, 式(5.60)で $\boldsymbol{\theta}_1^N$ と $\boldsymbol{\beta}$ を周辺化した

$$p(\mathbf{w}_1^N, \mathbf{z}_1^N \,|\, \boldsymbol{\alpha}, \boldsymbol{\eta}) = p(\mathbf{w}_1^N \,|\, \mathbf{z}_1^N, \boldsymbol{\eta})\, p(\mathbf{z}_1^N \,|\, \boldsymbol{\alpha}) \tag{5.77}$$

---

*32 実装の際には, 式(5.75)の第 1 項の分母の $\Sigma_{k=1}^{K}(\alpha_k + n'(i,k))$ は k に依存しませんので, 比例式としては不要です. また, 第 2 項の分母の $\Sigma_{v=1}^{V}(\eta + m'(k,v))$ も実際に V 個の和を毎回計算する必要はなく, $m'(k) = \Sigma_{v=1}^{V} m'(k,v)$ を保持しておいて更新すれば, $V\eta + m'(k)$ で求められることに注意してください.

になります[153]. この各項は, 図 5.25 に示したように $n(i,k)$ および $m(k,v)$ を行列で表せば, 各行にそれぞれ潜在的な多項分布 $\boldsymbol{\theta}_i$, $\boldsymbol{\beta}_k$ が存在してディリクレ分布 $\mathrm{Dir}(\boldsymbol{\alpha})$, $\mathrm{Dir}(\boldsymbol{\eta})$ に従い, これから各行の頻度 $n(i,:)$ および $m(k,:)$ が生まれたことを意味します. したがって, この尤度は, 3 章の式(3.58)で学習したポリア分布になり, 式(5.77)の右辺の各項は $m(k)=\sum_v m(k,v)$ とおくと,

$$
\left\{
\begin{array}{ll}
p(\mathbf{z}_1^N \,|\, \boldsymbol{\alpha}) & = \displaystyle\prod_{i=1}^{N} \left[\frac{\Gamma(\sum_k \alpha_k)}{\Gamma(T_i + \sum_k \alpha_k)} \prod_{k=1}^{K} \frac{\Gamma(\alpha_k + n(i,k))}{\Gamma(\alpha_k)} \right] \\
p(\mathbf{w}_1^N \,|\, \mathbf{z}_1^N, \boldsymbol{\eta}) & = \displaystyle\prod_{k=1}^{K} \left[\frac{\Gamma(V\eta)}{\Gamma(V\eta + m(k))} \prod_{v=1}^{V} \frac{\Gamma(\eta + m(k,v))}{\Gamma(\eta)} \right]
\end{array}
\right. \tag{5.78}
$$

で計算できることに注意してください. これから式(2.68)を使って, 学習データの 1 単語あたりのパープレキシティを求めることができます.

LDA の実験

LDA は単語ごとに潜在トピックが存在する確率モデルのため, Python のみで実装すると, 非常に計算が遅くなります. モジュールを C 言語にコンパイルして高速化する Cython を使った実装を筆者が公開していますので, 公開ページ [*33], または本章のサポートページから `lda.py-0.2.tar.gz` をダウンロードした後,

```
% tar xvfz lda.py-0.2.tar.gz
% cd lda.py-0.2/
% make
```

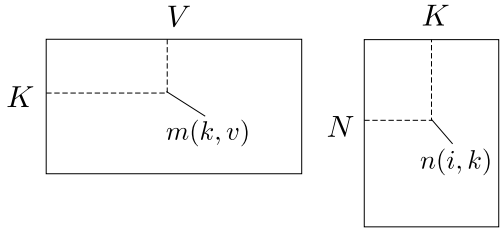


図 5.25: LDA の周辺化 Gibbs サンプリングで保持する統計量 $m(k,v)$, $n(i,k)$ の行列. 行列の各行の頻度を生成した多項分布がそれぞれディリクレ分布に従っており, 全体の確率は, ポリア分布の積になります.

*33 `http://chasen.org/~daiti-m/dist/lda-python/`

表 5.9: LDA ($K=50$) で教師なし学習された Livedoor コーパスのトピックの例. 表 5.1 の教師ラベル (カテゴリ) では表されない, 細かい意味的トピックが学習されていることがわかります.

Topic 1		Topic 7		Topic 43		Topic 48	
仕事	0.0357	応募	0.0435	料理	0.0276	結婚	0.0464
転職	0.0297	名	0.0431	食べ	0.0210	女	0.0389
会社	0.0246	プレゼント	0.0370	店	0.0120	女性	0.0367
情報	0.0225	様	0.0338	味	0.0118	彼	0.0334
年収	0.0179	当選	0.0291	野菜	0.0105	男性	0.0303
人	0.0161	月	0.0226	食べる	0.0086	男	0.0258
自分	0.0160	キャンペーン	0.0224	食	0.0080	恋愛	0.0244
万	0.0160	日	0.0193	酒	0.0074	相手	0.0216
livedoor	0.0158	終了	0.0177	ビール	0.0071	彼女	0.0171
求人	0.0131	場合	0.0172	食事	0.0070	自分	0.0140

でモジュールをコンパイルし, 準備することができます. これには Cython が必要ですので, 事前に `% pip install cython` などを実行してインストールしておいてください.

この上で, Livedoor コーパスについて

```
   % lda.py -K 50 -N 400 livedoor.dat model livedoor.lex
⇒ LDA: K = 50, iters = 400, alpha = 1, beta = 0.01
   loading data.. documents = 7367, lexicon = 16946, nwords = 1838414
   initializing..
   Gibbs iteration [400/400] PPL = 4761.2851
   saving model to model ..
   including dictionary.
   done.
```

を実行すると, トピック数 $K=50$ の LDA を学習することができます. この結果の一部を, 表 5.9 に示しました. 執筆時の環境では, この計算には約 6 分かかります. 上では MCMC の繰り返し数を 400 にしていますが, `model.log` に保存される学習データのパープレキシティの様子を見て, 収束するまで適宜増やすといいでしょう. 表 5.9 からわかるように, LDA により, 文書全体を 1 トピックとする UM や教師ラベルでは表現できない, 仕事, プレゼント, 食事, 恋愛などの細かいトピックが自動的に学習されていることがわかります.

表 5.10(a) に, 同様にして `jawiki.txt` について表 5.7 の UM と同じ $K=100$

表 5.10: LDA ($K=100$) で学習した日本語 Wikipedia コーパスのトピック.

(a) 生成確率 $p(w|z)$ を使った場合

Topic 10		Topic 20		Topic 30		Topic 100	
県	0.2955	世紀	0.0832	的	0.2476	月	0.3766
市	0.2018	時代	0.0777	化	0.0267	日	0.3252
福岡	0.0246	歴史	0.0341	基本	0.0255	年	0.2580
埼玉	0.0239	初期	0.0291	人間	0.0243	テン	0.0030
広島	0.0229	頃	0.0286	方法	0.0243	ルー	0.0017
兵庫	0.0229	年代	0.0224	主	0.0233	支え	0.0013
愛知	0.0215	説	0.0217	一般	0.0223	姓	0.0011
千葉	0.0194	当時	0.0215	表現	0.0206	セントラル	0.0011
九州	0.0188	記	0.0172	用い	0.0200	政治	0.0008
名古屋	0.0172	古代	0.0167	手法	0.0176	会議	0.0008

(b) 正規化自己相互情報量 NPMI (w,z) を使った場合

Topic 10		Topic 20		Topic 30		Topic 100	
県	0.7763	世紀	0.6493	的	0.7374	月	0.7545
市	0.6981	時代	0.5917	基本	0.5516	日	0.7281
福岡	0.5526	頃	0.5474	方法	0.5420	年	0.5677
埼玉	0.5511	初期	0.5386	人間	0.5392	テン	0.4269
広島	0.5488	年代	0.5383	手法	0.5311	ルー	0.3910
兵庫	0.5488	歴史	0.5379	主	0.5289	支え	0.3356
愛知	0.5442	説	0.5359	表現	0.5238	モー	0.3236
千葉	0.5350	記	0.5305	分析	0.5201	セントラル	0.3170
名古屋	0.5306	中世	0.5260	図	0.4977	ウイング	0.2552
新潟	0.5306	不明	0.5195	成分	0.4944	カルロス	0.2548

トピックの LDA を学習した場合のトピックの一部を示しました. LDA では, 文書全体を 1 トピックで表現する必要はないため, トピック 30 のように「的」「化」といった機能語に対応するトピックができることが多く [*34], 生成確率 $p(w|z)$ を使っても, 他のトピックで機能語が上位に来ることはあまりありません. 表 5.10(b) に示したように, この場合も NPMI を用いることで, 各トピックに相関する単語をより明確に取り出すことができます. 図 5.26 に, 各文書 d に対する潜在トピック分布 $\theta=\{p(z|d)\}$ を示しました. 同じトピック数の図 5.9 の UM

*34 **??**ページで示すように, ハイパーパラメータ α_k も推定して学習すると, こうした結果になりやすいことが確かめられています[157].

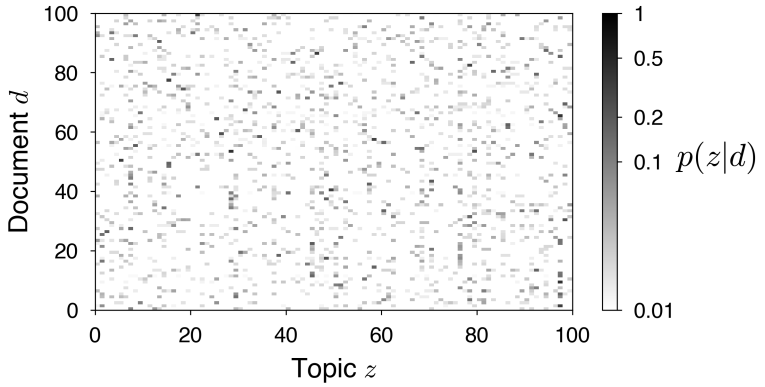


図 5.26: 日本語 Wikipedia コーパス `jawiki.txt` から学習された LDA による, 各文書のトピックへの所属確率 $p(z|d)$ (最初の 100 文書). トピック数は $K=100$ としました. 図 5.9 の UM の場合と比較すると, 文書に対応する各行が, 複数の潜在トピックを確率的に持っていることがわかります.

の場合と比べると, $p(z|d)$ がどれか 1 つのトピックに集中することがなく, 各文書が複数の潜在トピックからなる様子が学習されているのがわかります.

5.4.2 LDA の幾何的解釈

LDA は, 幾何的には何をしていることに相当するのでしょうか. LDA では, トピック分布 $\boldsymbol{\theta}$ が決まると, 文書の各単語は

1. トピック $z \sim \boldsymbol{\theta}$ をサンプルし,
2. 単語 $w \sim p(w|z)$ をサンプルする

という 2 段階で生成されます. 確率で書くと, $\boldsymbol{\theta}$ から w と z を生成する確率は

$$p(w, z|\boldsymbol{\theta}) = p(w|z)p(z|\boldsymbol{\theta}) \tag{5.79}$$

となっているわけです.

しかし, よく考えるとわれわれが観測しているのは w だけなのですから, z については和をとって周辺化してしまってもよいでしょう.[*35] つまり, w は確率分布

*35 これを 211 ページの脚注のように, Rao-Blackwell 化ともいうのでした.

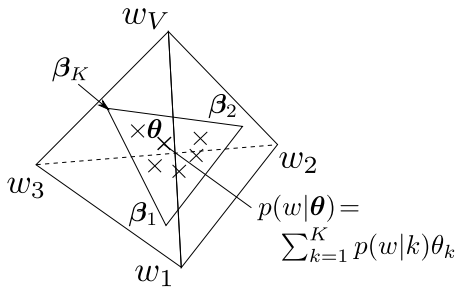


図 5.27: LDA の幾何的表現. LDA は×で示した単語の生成確率をモデル化できるよう, 外側の V 次元の単語単体の中で内側の K 次元のトピック単体 $\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ の位置を最適化し, 次元圧縮を行う統計モデルです.

$$p(w|\boldsymbol{\theta}) = \sum_z p(w, z|\boldsymbol{\theta}) = \sum_z p(w|z)p(z|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(w|k)\theta_k \tag{5.80}$$

に従う 1 段階で生成されたと考えても, 数学的には等価になります.

トピック分布 $p(w|k)$ は, k ごとの単語の確率分布 $\boldsymbol{\beta}_k = (p(w_1|k), \cdots, p(w_V|k))$ のことですから, これは図 5.14 でみたように, 単語単体の中の 1 点になります. 図 5.27 に示したように, 式(5.80)は, テキストが $\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ を $\theta_1, \cdots, \theta_K$ の割合で内分した点 $p(w|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(w|k)\theta_k$ から生成されたことを表しています. このように, V 次元の単語単体の中にあって $\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ によって張られる K 次元の単体のことを, **トピック単体**といいます.

$p(w|\boldsymbol{\theta})$ はこのトピック単体の中にあるのですが, 一般に $K \ll V$ ですから, トピック単体 $\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ をうまく選ばないと, 文書を生成した確率分布 $\mathbf{p}$ を表現することができません. このように,

(1) トピック, すなわちトピック単体の「角」$\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_K$ と,
(2) 文書ごとの $\boldsymbol{\theta}$, すなわち上で定まるトピック単体内の各文書の位置

を同時に最適化しているのが LDA です. このとき $\boldsymbol{\theta}$ にはディリクレ事前分布を仮定して, 各文書にディリクレ事後分布を割り当てる (allocation) ため, **潜在ディリクレ配分法**とよばれているわけです.

空間内に, データをうまく表現する低次元の部分空間を張るという意味では, これは実数値 (ユークリッド空間) における主成分分析 (PCA) に似ています. た

だし PCA と異なり, LDA は離散データのために単体上で定義された統計モデルなのが特徴で, 低頻度のデータでもうまく扱うことができます. Buntine ら [158]は, LDA およびその拡張を統一して離散 PCA (Discrete PCA) とよんでいます.

メモ：ディリクレ分布の $\boldsymbol{\alpha}$ のサンプリング *

LDA のハイパーパラメータはトピック分布およびトピック–単語分布を生成するディリクレ分布のハイパーパラメータ $\boldsymbol{\alpha}$, η ですが, これらはどうやって決めたらいいのでしょうか.

一般的にはトピック数を K として, $\boldsymbol{\alpha}=(50/K,\cdots,50/K)$, $\eta=0.01$ のようにヒューリスティックに決められることが多いのですが, これらのハイパーパラメータの影響を詳しく調べた研究[159]によると, これらもデータから学習した方がよいモデルとなることが示されています. とくに, 非対称な $\boldsymbol{\alpha}=(\alpha_1,\alpha_2,\cdots,\alpha_K)$ を推定することで, α_k が大きいトピックに「が」「の」, 英語なら “of”, “the” といった機能語 *36 が集まり, 他のトピックにこれらが混ざることが少なくなります. また, トピック数を増やしても不要なトピックは α_k が小さく, 自然に使われなくなることも報告されています.

$\boldsymbol{\alpha}$ と η についての尤度関数は式(5.78)ですから, 数学的にはこれは, このポリア分布のハイパーパラメータを推定することと等価です. ただし, このカウント $n(i,k)$, $m(k,v)$ はあくまで学習中の $\mathbf{z}_1^N$ から計算される値ですから, 学習中に式(3.60)を使って $\boldsymbol{\alpha}$ を最適化してしまうと, 局所解に陥る危険があります.*37 よって, きちんと $\boldsymbol{\alpha}$ のベイズ推定を行うのが正しい方法でしょう.

ポリア分布を表す式

$$p(\mathbf{n}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N+\sum_k \alpha_k)} \prod_{k=1}^{K} \frac{\Gamma(\alpha_k+n_k)}{\Gamma(\alpha_k)} \tag{5.81}$$

は, α_k がガンマ関数 $\Gamma()$ の中に含まれているため, そのままではサンプリングできる形になりません. しかし, 巧妙な補助変数 θ, x を導入すると, α_k はガンマ事後分布からサンプリングすることができます. 導出はやや複雑なため, 詳しくは, 付録 E を参照してください.

*36 もしこうした機能語を「ストップワード」として除いても, 237 ページで説明したように, 同様に意味の薄い語が必ず出現し, 機能語とそれ以外は簡単に二分することができません.

*37 すなわち, 学習途中で $\boldsymbol{\alpha}$ を最適化してしまうと, 学習アルゴリズム全体が Gibbs サンプリングではなく, 247 ページのモンテカルロ EM アルゴリズムを行うことになってしまいます.

5.4.3 トピックモデルの評価

こうして学習したトピックモデルが, 文書の「よいモデル」になっているかどうかは, どうやって測ればよいのでしょうか. LDA は先に説明したように, 単語単体上の密度推定を行う統計モデルですから, テストデータに高い確率を与えられるかが, 最も素直な評価方法といえるでしょう. LDA の評価方法について詳しく調べた研究[157]によると, これにはテストデータの各文書について, 前半からトピック分布を学習し, それに基づいて後半の単語の予測パープレキシティを計算する方法が最も良かったことが報告されています.

サポートサイトの本章のフォルダに, この評価スクリプト `ldaeval.py` を示しました. LDA は順番を考慮しない単語集合のモデルですから, 評価の際には各文書で単語をランダムにシャッフルし, 前半 $\mathbf{w} = w_1 \cdots w_{T/2-1}$ から計算したトピック分布 $\boldsymbol{\theta}$ を用いて, 後半 $\mathbf{w}' = w_{T/2} \cdots w_T$ の確率を

(5.82)

$$p(\mathbf{w}'|\boldsymbol{\theta}) = \prod_{t=T/2}^{T} p(w_t|\boldsymbol{\theta}) = \prod_{t=T/2}^{T} \sum_{z_t} p(w_t, z_t|\boldsymbol{\theta}) = \prod_{t=T/2}^{T} \sum_{k=1}^{K} p(w_t|z_t=k) \underbrace{p(z_t=k|\boldsymbol{\theta})}_{=\theta_k}$$

のように計算します. なお, $\mathbf{w}$ からそれを生成した $\boldsymbol{\theta}$ を求めるには式(5.75)を用いることもできますが, パラメータはすでに学習済みですので, 変分ベイズ EM アルゴリズムを用いると高速に行えます. 詳しくは, `ldaeval.py` のスクリプトの中身および[155]を参照してください.

図 5.28 に, Livedoor コーパスのうちランダムな 367 文書をテストデータ, 残りの 7,000 文書を学習データとした場合の予測パープレキシティを, 各トピック数 K について示しました. LDA はベイズ的な手法のため, K を大きくしても過学習して性能が落ちることはなく, パープレキシティはゆっくり減少していくことがわかります. また, 付録 E の方法でハイパーパラメータ $\boldsymbol{\alpha}, \boldsymbol{\eta}$ を推定した場合, 固定の場合と比べて予測精度が上がっていることも読みとれます.

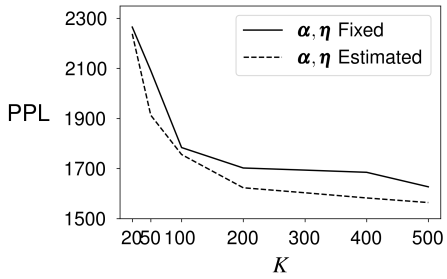


図 5.28: Livedoor コーパスにおけるテストデータの予測パープレキシティとトピック数 K の関係. ハイパーパラメータ $\boldsymbol{\alpha}, \boldsymbol{\eta}$ を固定した場合 (Fixed) に対して, それらもベイズ推定した場合 (Estimated) の方が, 一貫して低いパープレキシティを達成することがわかります.

トピックの直接評価

上ではトピックモデルの性能を予測精度で評価しましたが, これは得られたトピックが意味的まとまりの高い, 「よいトピック」であることを保証しているわけではありません. 「よいトピック」であることは, どうやって測ればいいのでしょうか? トピックモデルは教師なし学習ですので, この問題にはもちろん「正解」はないのですが, いくつかの方法が提案されています[160][161]. 以下では, 各トピック k について, 生起確率 $p(w|k)$ の高い順に N 語 (たとえば $N = 10$) とった単語集合を「トピック語」$t(k)$ とよぶことにします.

Word Intrusion 各トピックが意味的によくまとまっていれば, トピック語に無関係な語 (侵略者, intruder) を混ぜてもたやすく検出できるはずです. 逆にノイズが多いトピックなら, どれが侵略者なのかを判断できないでしょう. たとえば図 5.29 で, A のトピック語に混ぜた “京都” は簡単に発見できますが, B に混ぜた “大阪” はすぐには見分けられません. この判断は人間の被験者に行ってもらうこともできますし, サポートベクトル回帰 (SVR) で自動化する方法も提案されています[161]. *38

Topic Coherence トピックが意味的にまとまっているかを, 直接測ることもできます. 5.2.1 節で議論したように, NPMI を使うとトピックと相関の高い単

*38 `https://github.com/jhlau/topic_interpretability` で, 自動評価のための評価スクリプトが公開されています.

トピック A	酸, 酵素, **京都**, 化合, 合成, 水素, 有機, 触媒
トピック B	湖, 不足, キル, PK, 通勤, **大阪**, テレビジョン

図 5.29: Word intrusion の例. 意味的まとまりの高いトピック A では, 太字で示した侵略者 (intruder) はすぐに見分けることができますが, 意味的まとまりの低いトピック B では, どれが侵略者なのか判断がつきません.

語を取り出すことができますから, よいトピックであれば, それらの単語どうしの NPMI も高くなるはずです. トピック語のすべてのペア (v, w) について, 外部の大きなテキストでの共起から計算した NPMI(v, w) の平均値を Coherence と定義すると, これは上の Word Intrusion および人間の判断と高い相関を持つことが示されています[161].

Topic Uniqueness トピックが意味的にまとまっていても, ほとんど同じトピックが複数あるのは望ましくないでしょう. 各トピックのトピック語を「文書」とみなしたとき, ある単語 w が複数のトピック語に現れていれば, 288 ページで学ぶ文書頻度 (=何個のトピック語に現れたか) $\mathrm{df}(w)$ は 1 より大きくなります. よって, その逆数をとって平均し,

$$\mathrm{TU} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{N}\sum_{w\in t(k)}\frac{1}{\mathrm{df}(w)} \tag{5.83}$$

を計算すれば, トピック語がすべて異なっていれば 1, 重複があると <1 になるはずです. これを Uniqueness と定義して評価する方法が提案されています[162].

ただし, 式(5.83)の指標はトピック数 K の違いについてロバストではありません. K が大きくなれば, ある単語が他のトピック語にも偶然含まれる可能性は, それだけ大きくなるからです. 式(5.83)は本質的に, すべてのトピック語をまとめた集合での各単語 w の出現回数 $\mathrm{df}(w)$ を計算しているだけですから, それをユニグラム分布 $p(w) = \mathrm{df}(w)/(NK)$ に直せば, 2 章で学習した $p(\cdot)$ のエントロピー (式(2.65)) $H(p(\cdot))$ を指標とすればよいでしょう. この最大値は, すべての単語が 1 回ずつ現れる (=トピック語に重複がない) 場合で, $\log(NK)$ になります. よって最大値との比をとり,

$$\mathrm{TU}{+}{+} = H(p(\cdot))/\log(NK) \tag{5.84}$$

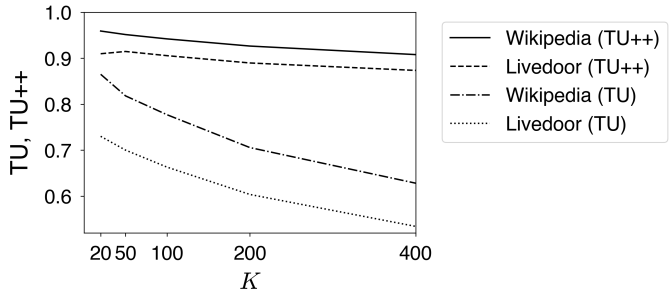


図 5.30: Livedoor コーパスおよび日本語 Wikipedia で計算した TU,TU++の値とトピック数 K. K の選択には使えませんが, 同じトピック数での比較のためには有効です.

を指標とすれば, これは K に依存しません. この指標 (本書のオリジナルです) を TU++とよぶことにしましょう.

こうした評価は, LDA に限らずトピックモデル全般に用いることができます. 5.2 節の UM について, EM アルゴリズムで推定した表 5.6 のトピックと, Gibbs サンプリングで計算した表 5.7 のトピックを比べてみましょう. サポートサイトの `coherence.py` を使って計算すると,

```
   % coherence.py model/um.jawiki.K100 data/ja.text8.txt
⇒  obtaining topic words..
   computing cooccurrences..
   calculating coherence..
   average = -0.0997
```

のようになります. ここでは, 日本語 text8 を共起計算のためのコーパスとして用いました.[*39] EM アルゴリズムの場合は上のように −0.0997, Gibbs サンプリングの場合は −0.0684 となり, 確かに Gibbs サンプリングの方が意味的まとまりの高いトピックが学習されていることが定量化されました.

また Uniqueness については, Livedoor コーパスについて `uniqueness++.py` で TU および TU++をさまざな K について計算すると図 5.30 のようになり, $K = 50$ 程度でやや TU++の値が高くなることがわかります. ただし, Wikipedia の場合は指標は K とともに一様に減少するようです.

∗39 この 10 倍のサイズがある 134 ページの `ja.text9` を使っても, 統計値はほとんど同じになりました.

トピック数 K の選択 トピックモデルにおいて, トピック数 K の選択は重要な問題です. 上でみたように, 予測パープレキシティ, Uniqueness (TU++) のいずれも, それだけでトピック数を適切に決めることはできません.[*40] 理論的には, LDA のトピック数は階層ディリクレ過程 (HDP) を用いた HDP-LDA で推定でき, この問題は解決されています[126]. ただし, HDP の理解と実装には測度論の知識が必要となり, 本書の範囲を超えますので, 簡便には (1) HDP-LDA が実装されている `gensim` などのパッケージ [*41] を用いるか, または (2) 4 章で行ったように K を大きめにとって $\alpha<1$ と小さく設定するか, あるいは (3) $\boldsymbol{\alpha}$ をサンプリングして学習することで, 実質的に必要なトピックだけを残すようにするとよいでしょう. HDP による無限モデルに興味のある方は, 優れた導入である[163]を参照してください.

LDA の拡張

- Supervised
- Embedding
- Tea party

---

*40 LDA で学習データの確率が最も高くなるのは, トピックの数と単語数 V が等しく, 各トピックで β_{kv} がある単語だけ確率 1, 他は 0 となる場合です. しかし, 式(5.76)の $\boldsymbol{\beta}$ の事後分布から, $\eta>0$ のときにそうなる確率は 0 です. したがって η を固定すれば, 式(5.78)のエビデンス $p(\mathbf{w}_1^N|\mathbf{z}_1^N,\boldsymbol{\eta})$ を最大にするトピック数 K を求めることができます[153]. しかしこれは η に依存し, η=0.1 と η=0.01 ではまったく違うトピック数になってしまいます. 式(0.139)から η をサンプリングする場合, K が大きいほど推定される η は小さくなり, この方法で最適な K を選ぶことはできません.

*41 `https://radimrehurek.com/gensim/models/hdpmodel.html`

5.5　ニューラル文書モデルと独立成分分析

LDA は, 各文書にディリクレ分布に従う潜在的なトピック分布 $\boldsymbol{\theta}$ があると考え, $\boldsymbol{\theta}$ とトピックの両方をデータから学習できる, 解釈性に優れたモデルですが, LDA にもいくつかの問題があることには注意が必要です.

- 一つは, 文書の内容を表現する $\boldsymbol{\theta}$ が多項分布に制限されていることです ($\sum_k \theta_k = 1, \theta_k \geq 0$). こうすると, たとえば経済と数学の両方の内容を持つ文書は, 経済トピックが強いほど数学トピックは弱くなり, その逆も成り立ちます. また, あまり内容がなかったり短い文書でも, $\boldsymbol{\theta}$ は内容の濃いテキストと同様に, 和が 1 の確率分布になってしまいます.
- 二つ目は, 単語を離散的にとらえているため, 3 章で学習した単語ベクトルのように単語間の関係を精密にモデル化できないことです. たとえば, “取る” と “取得” のようにほぼ同じ意味の言葉でも, LDA では「同じ潜在トピックから生成される確率が高い」という間接的な形でしか関係をとらえることができません.
- また, LDA は文書に含まれる各単語ごとにトピックを推定するため, 精密なモデル化ができる一方で, 大規模なコーパスでは計算量が大きくなってしまうという問題があります.

ここで, LDA で文書の $\boldsymbol{\theta}$ がわかっている場合, その文書での単語 w の確率 $p(w|\boldsymbol{\theta})$ は, トピック z を考えて周辺化することで, 式(5.80)でもみたように

$$p(w|\boldsymbol{\theta}) = \sum_z p(w, z|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(w|z=k)p(z=k|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(w|k)\theta_k \tag{5.85}$$

と書けることに注意しましょう. 以下ではわかりやすさのため, 文書–単語行列の縦横を入れ替えて縦軸を単語, 横軸を文書として説明します. [*42] 上の確率を V 個の単語について縦に並べれば,

---

*42　Python の Numpy では, 行列の要素は標準では内部的に行方向が先に格納されるため (row-major といいます), これまで説明した文書–単語行列の形が実装上は有利です.

$$(5.86)\qquad \underbrace{\begin{pmatrix} p(w_1|\boldsymbol{\theta}) \\ p(w_2|\boldsymbol{\theta}) \\ p(w_3|\boldsymbol{\theta}) \\ \vdots \\ p(w_V|\boldsymbol{\theta}) \end{pmatrix}}_{\mathbf{p}} = \underbrace{\begin{pmatrix} p(w_1|1) & p(w_1|2) & \cdots & p(w_1|K) \\ p(w_2|1) & p(w_2|2) & \cdots & p(w_2|K) \\ p(w_3|1) & p(w_3|2) & \cdots & p(w_3|K) \\ \vdots & & & \vdots \\ p(w_V|1) & p(w_V|2) & \cdots & p(w_V|K) \end{pmatrix}}_{\mathbf{B}} \underbrace{\begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_K \end{pmatrix}}_{\boldsymbol{\theta}}$$

と書くことができ, $\boldsymbol{\beta}_k$ を並べた行列 $\mathbf{B}=(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_K)$ を使って $\mathbf{p}=\mathbf{B}\boldsymbol{\theta}$ と表すことができます. 式(5.86)を N 個の文書について横に並べると, n 番目の文書の $\boldsymbol{\theta}$ を $\boldsymbol{\theta}_n$ として

(5.87)

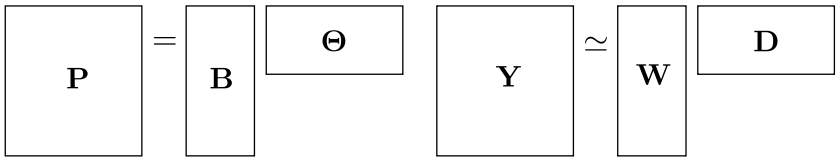
$$\underbrace{\begin{pmatrix} p(w_1|\boldsymbol{\theta}_1) & \cdots & p(w_1|\boldsymbol{\theta}_N) \\ p(w_2|\boldsymbol{\theta}_1) & \cdots & p(w_2|\boldsymbol{\theta}_N) \\ p(w_3|\boldsymbol{\theta}_1) & \cdots & p(w_3|\boldsymbol{\theta}_N) \\ \vdots & & \vdots \\ p(w_V|\boldsymbol{\theta}_1) & \cdots & p(w_V|\boldsymbol{\theta}_N) \end{pmatrix}}_{\mathbf{P}} = \underbrace{\begin{pmatrix} p(w_1|1) & \cdots & p(w_1|K) \\ p(w_2|1) & \cdots & p(w_2|K) \\ p(w_3|1) & \cdots & p(w_3|K) \\ \vdots & & \vdots \\ p(w_V|1) & \cdots & p(w_V|K) \end{pmatrix}}_{\mathbf{B}} \underbrace{\begin{pmatrix} \theta_{11} & \theta_{21} & \cdots & \theta_{N1} \\ \theta_{12} & \theta_{22} & \cdots & \theta_{N2} \\ \vdots & & & \vdots \\ \theta_{1K} & \theta_{2K} & \cdots & \theta_{NK} \end{pmatrix}}_{\boldsymbol{\Theta}}$$

となり, 図 5.31(a) に示したように行列形式で

$$(5.88)\qquad \mathbf{Y} \sim \mathbf{P}, \quad \mathbf{P}=\mathbf{B}\boldsymbol{\Theta}$$

と表すことができます. つまり LDA は, $\mathbf{P}$ と同じ大きさの単語–文書行列 $\mathbf{Y}$ が多項分布に従って $\mathbf{Y}\sim\mathbf{P}$ と生成されたと考え, この $\mathbf{P}$ を式(5.88) のように分解する**行列分解**を教師なし学習している, と考えることができるわけです. ここで $\sim$ とは, 観測値 $\mathbf{Y}$ の各列が $\mathbf{P}$ の同じ列をパラメータとして多項分布で生成されたことを表します. [*43]

*43 $\boldsymbol{\Theta}$ と $\mathbf{B}$ はともに非負ですから, これは非負値行列因子分解 (NMF) の一種とみなすことができます. Lee らが 2000 年に提案した NMF [164]は, 統計的には頻度がポアソン分布に従うことに対応しています. ポアソン分布はスケールが文書の長さに依存してしまい, 正規化することで多項分布となるため, 式(5.88)はその一般化といえます. NMF とテキストの Gamma–Poisson モデル (GaP) [165]については, 筆者のメモ[166]も参照してください.



(a) LDA および PLSI　　(b) LSI

図 5.31: 行列分解による文書モデル. 左辺の行列を, 右辺の二つの行列の積で近似しています. このとき, 右辺の行列の各行と各列が「単語ベクトル」および「文書ベクトル」になります.

式(5.87)で, 確率分布の形で各文書の $\boldsymbol{\theta}_n$ は文書 n の「埋め込み」, $\mathbf{B}$ の各行 $\phi(w) = (p(w|1), p(w|2), \cdots, p(w|K))$ も単語 w の一種の「埋め込み」と考えることができます. しかし, $\boldsymbol{\theta}$ も $\phi(w)$ も負になることができず, 縦横に和が 1 になる必要もあるため, この行列分解には大きな制約が伴います.

それならば, いっそのこと $\mathbf{Y}$ を図 5.31(b) のように直接,

$$\mathbf{Y} \simeq \mathbf{WD} \tag{5.89}$$

と, 一般の実行列 $\mathbf{D}, \mathbf{W}$ に行列分解 (145 ページ) すれば, $\mathbf{W}$ と $\mathbf{D}$ の各行は制約のない単語ベクトル, 文書ベクトルになるのではないでしょうか. この考えに基づくのが, 情報検索の分野で 1990 年に提案された **LSI** (Latent Semantic Indexing) [167]でした. LSI では, 頻度 $\mathbf{Y}$ に含まれる頻度 $Y(w,d)$ をそのまま使うとほとんどを機能語が占めてしまうため, $\widetilde{Y}(w,d) = \mathrm{tfidf}(Y(w,d))$ のように, この後で説明する **tf.idf** のような重みづけを行ってから, 式(5.89)のように行列分解を計算して文書ベクトル・単語ベクトルを求めます.

ただし, LSI では tf.idf のような単語の重みづけ法がアドホックで, 得られるベクトルの数学的な意味が弱いという大きな欠点がありました. そこで, LSI を多項分布による確率モデルとして式(5.88)のようにとらえ直した, **PLSI** (Probabilistic Latent Semantic Indexing) という革新的なモデルが 1999 年に提案され[168], それが 2001 年にベイズ化されて LDA になった[150]という歴史があります. 実験的にも, パープレキシティで測った性能は LDA>PLSI>LSI の順に高いことがわかっており[137], テキストをアドホックでなく, 確率的に考えることの重要性を示しています.

5.5.1 文書ベクトルと Doc2Vec

ただし, 実行列による行列分解自体に意味がないわけではありません. 皆さんは図**??**のような行列分解は, 本書でこれまでに覚えがあるのではないでしょうか. これは, 3 章の図 3.32 でみた, 行列分解によるニューラル単語ベクトルの学習

$$\mathbf{X} \simeq \mathbf{W}\mathbf{C}^T \tag{5.90}$$

と, ほとんど同じ形をしていることに注意しましょう. 単語ベクトルの場合, もとの共起行列 $\mathbf{X}$ の要素 $X(w,c)$ は, 単語 w とその周辺に出現した文脈語 c との非負自己相互情報量 (PPMI)

$$X(w,c) = \mathrm{PPMI}(w,c) = \max\left(\log\frac{p(w,c)}{p(w)p(c)}, 0\right)$$

でした (式(3.109)). そこで, 単語–文書行列の場合は単語 w とそれが出現した文書 d の PPMI を用いて,

$$X(w,d) = \mathrm{PPMI}\,(w,d) = \max\left(\log\frac{p(w,d)}{p(w)p(d)}, 0\right) \tag{5.91}$$

を並べた行列 $\mathbf{Y}$ を作り, これを図 5.32 のように

$$\mathbf{X} \simeq \mathbf{W}\mathbf{D} \tag{5.92}$$

と行列分解すれば, 行列 $\mathbf{D}$ の各列 $\vec{d}$ は Word2Vec と数学的に等価な「ニューラル文書ベクトル」に, $\mathbf{W}$ の各行 $\vec{w}$ は「ニューラル単語ベクトル」になるのではないでしょうか. *44

実際に, Mikolov らが Word2Vec の後に提案し, よく使われている **Doc2Vec** [169]は文書ベクトルから文書に含まれる単語を予測するモデルで, 単語ベクトルのスキップグラム (3.5.3 節) と同じ目的関数をしており, 式(5.92)と等価です. しかし, 確率的勾配法による繰り返し計算で近似を行う Doc2Vec に対して,

*44 すなわち, LSI に足りなかったのは適切な単語の重みづけと, それを支える背後の理論だったということです. 144 ページの脚注で述べたように, 実験的には PPMI による単語重みづけが高性能であることは, 深層学習以前に一部ではすでに知られていました.

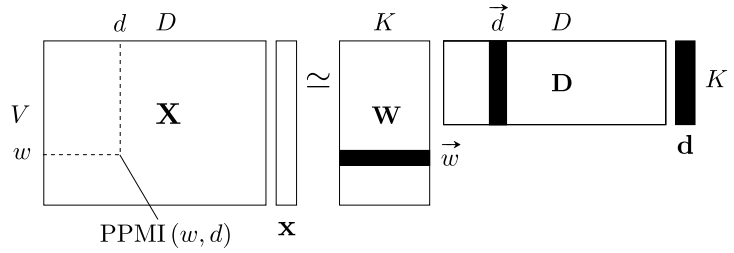


図 5.32: DocVec による文書ベクトルと単語ベクトルの計算. 右辺の $\vec{d}$ と $\vec{w}$ の内積で左辺の PPMI (d, w) を近似し, これは Doc2Vec (Word2Vec) の学習と数学的に等価です.

式(5.92)から得られる文書ベクトル (本書ではこれを **DocVec** とよぶことにします) はこの後で説明するように線形代数で高速に解くことができ, 数学的な最適解が求まるために, 性能も高いことを筆者が確かめています[170].

なお, 式(5.91)はベイズの定理から $p(w|d) = p(w, d)/p(d)$ ですから,

$$\text{PPMI}\,(w, d) = \max\left(\log \frac{p(w, d)}{p(w)p(d)}, 0\right) = \max\left(\log \frac{p(w|d)}{p(w)}, 0\right) \tag{5.93}$$

で求めることができます. すなわち, 式(5.93)の PPMI は, 「文書 d の中で単語 w が通常より何倍多く出現したのか」という値の対数となっています. d の中での w (たとえば "the") の出現確率が平均的な確率とほぼ同じならば, $p(w|d) \simeq p(w)$ ですから, 式(5.93)の比は

$$\log \frac{p(w|d)}{p(w)} \simeq \log 1 = 0$$

となることに注意してください. 機能語はどの文書でも確率がほぼ一定なので, こうすれば「ストップワード」を準備しなくても, 機能語の PPMI はほぼ 0 になることがわかります.

DocVec の学習アルゴリズムを, 図 5.34 に示しました. この計算は, サポートサイトの `docvec.py` で行うことができます [*45]. 図 5.32 からわかるように, このとき **D** の各行として文書ベクトル $\vec{d}$ が, **W** の各行として単語ベクトル $\vec{w}$ が

*45　疎行列の特異値分解を行う SciPy の `svds()` を使う場合, 特異値が大きい順ではなく, 小さい順に返されますので注意してください. 標準では, 次元が重要な順の逆に並ぶことになります.

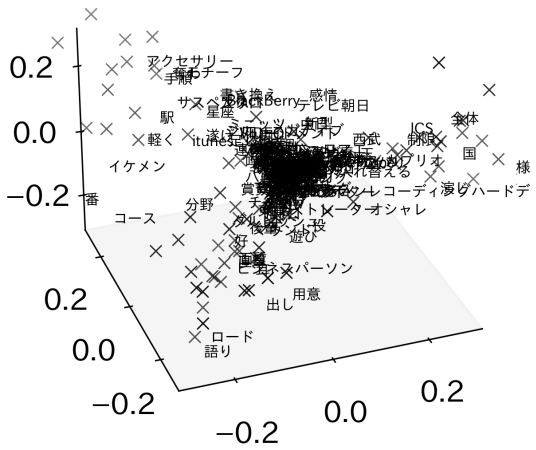


図 5.33: Livedoor コーパスから DocVec で計算した文書ベクトル (×印) と単語ベクトルの一部. 3.5.6 節で議論したように, 最大の特異値に対応する次元は全体のバイアスを表しているため, ここでは特異値の大きい方から 2,3,4 次元目の値でプロットしました. これから, Livedoor コーパスの文書ベクトル・単語ベクトルは大きく「女性軸」(左上), 「電気製品軸」(右), 「一般軸」(左下) に沿って分布していることがわかります. ただし, 座標軸が必ずしもそれらに沿ってとられているわけではありません.

得られることに注意してください. 図 5.33 に示したように, これらのベクトルは同じK次元の潜在空間に存在しており,

$$\mathrm{PPMI}\,(w,d) \simeq \vec{w}\cdot\vec{d} \tag{5.94}$$

となるように文書ベクトル $\vec{d}$, 単語ベクトル $\vec{w}$ が学習されることになります.

文書ベクトルの計算 実際に実験してみましょう. サポートサイトにある `docvec.py` を使うと, Livedoor コーパスのデータ `livedoor.dat` について次のように実行すれば, $K=200$ 次元の文書ベクトルと単語ベクトルが計算できます.

```
% docvec.py -K 200 -d livedoor.lex livedoor.dat model.docvec
⇒ using dictionary: livedoor.lex
  livedoor.dat: K = 200, output = docvec.livedoor.K200
  parsing data..
  creating sparse matrix..
  computing data vectors..
```

```
done.
writing model to model.docvec..
done.
```

この計算は, 執筆時の環境では 14.6 秒で終了しました. これから, たとえば 200 番目の文書に似た文書ベクトルを検索したい場合は, コーパスの実際の中身が書かれているテキスト livedoor.txt (221 ページ) を同時に与えて, 次のように実行します.

```
% docvec-similar.py docvec.model livedoor.txt 200
⇒ loading model from docvec.model.. done.
 1.0000 dokujo-tsushin 30 歳を過ぎた大人の肌へ、世界が認めた美容液「」
 0.6536 dokujo-tsushin 女性も驚愕!?スキンケアしていない人は 70%と男
 0.6530 peachy         あなたの見た目年齢を上げているのは“シミ”だった
 0.6492 peachy         自宅でできるでふっくらお肌を目指そう!毎朝のメイ
 0.6315 peachy         日本初!DHC、“10 倍濃度”の Q10 シリーズを
 0.6148 kaden-channel  ライオン、毛と音波振動でくすみを落とす「プラチア
 0.6104 peachy         男性の 4 割が“すっぴん”にがっかり!最強のすっぴ
 0.5819 peachy         “究極のクリーム”で 10 年後も 20 年後もずっと綺
```

一番上の文書は自分自身 (類似度は $\cos 0 = 1$) ですが, ジャンルを超えて内容的に近い文書が検索できていることがわかります. 実装については, スクリプトの中身を読んでみてください.

キーワードによる検索　それでは, あるキーワード集合に近い文書を探したい場合はどうすればいいでしょうか. 実は, これは自明ではありません.[*46] というのは, いま探したい “映画 東京” のような検索語だけを含む文書は学習データには存在しないからです.

しかし, キーワード集合に対応する仮想的な「文書ベクトル」$\mathbf{d}$ が計算できれば, 学習した文書ベクトルと $\mathbf{d}$ を上記のように比べればよいでしょう. ここで, 図 5.32 では, 左辺の観測データ $\mathbf{Y}$ は常に式(5.91)の PPMI の形で与えられることに注意してください. この値が 0 のとき, 文書内での単語の確率は標準的な確率と等しいか小さいことを意味します. よって, 検索したい語の PPMI をたとえば 1 にすれば, これは仮想的な「文書」の中で確率が $e^1 \simeq 2.72$ 倍高いこと

*46　簡易的には, キーワードに対応する単語ベクトルの平均を計算して文書ベクトルと比較するという方法が考えられますが, モデル上これが最適である保証はありません. 実際, この後で計算して対応する列の和をとる行列 $(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ は, $\mathbf{W}$ 自体とは異なったものです.

を意味します. そこで, キーワード集合に含まれる語について 1, 他は 0 になる V 次元の縦ベクトル $\mathbf{x}$ を用意し, 図 5.32 に表したように

$$\mathbf{x} \simeq \mathbf{W}\mathbf{d} \tag{5.95}$$

が成り立つ文書ベクトル $\mathbf{d}$ を求めればよい, ということになります.

特異値分解は両辺の二乗誤差を最小化する方法ですので, 最小二乗の意味で式(5.95)が成り立つ $\mathbf{d}$ を求めるには, 単純に最適化を行うこともできますが, 式(5.95)はよく知られた線形回帰モデル (OLS) ですから, $\mathbf{d}$ の最適解は

$$\mathbf{d}^* = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{x} \tag{5.96}$$

で与えられます[6, §1.2]. あらかじめ回帰行列 $\mathbf{R}=(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ を計算しておけば [*47], 式(5.96)は

$$\mathbf{d}^* = \mathbf{R}\mathbf{x} \tag{5.97}$$

と, 一瞬で最適解が求められます.[*48] なお, 新しい文書自体が与えられている場合は, 式(5.93)から PPMI を計算して $\mathbf{x}$ を作れば, 同様に最適な文書ベクトル $\mathbf{d}^*$ を求めることができます.

この方法で Livedoor コーパスの文書について, 意味を考慮したキーワード検索を行った例は次のようになります. ニューラル単語ベクトルを介しているため, 検索語自体が出現していなくても, 意味が似ていれば (=単語ベクトルが近ければ) 文書が検索されることに注意してください.

```
% docvec.py -K 200 -R -d livedoor.lex livedoor.dat model.docvec-R
# -R をつけて実行し，回帰行列を事前に計算しておく
% docvec-search.py model.docvec-R livedoor.txt 映画 東京
⇒ loading model from model.docvec-R.. done.
keyword: 映画 東京
 0.0158 movie-enter     この夏、東京スカイツリーが映画に 2008 年 7 月の
 0.0152 movie-enter     小栗旬は“使えない若者”、映画『キツツキと雨』の
 0.0152 movie-enter     スパイダーマンが地上 75 メートルの通天閣を『アメ
```

*47 $\mathbf{R}\mathbf{W}=(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{W}=\mathbf{I}$ ですから, この $\mathbf{R}$ は $\mathbf{W}^+$ とも書かれる Moore-Penrose の擬似逆行列です.

*48 $\mathbf{y}$ の要素はほとんどが 0 ですから, 式(5.97)の計算は実際には, $\mathbf{R}$ のうち $\mathbf{x}$ の要素が 1 の列を足すだけの操作になり, `mmap()` などで $\mathbf{R}$ をディスクに保存すれば, 空間計算量も最小限に抑えることができます.

アルゴリズム 7: 文書ベクトル (DocVec) の計算アルゴリズム.

1: 式(5.91)に従って文書–単語行列 $\mathbf{X}$ を作成する. (疎行列フォーマットで)
2: $\mathbf{U}, \mathbf{S}, \mathbf{V} = \mathrm{svds}(\mathbf{X}, K)$ と K 次元に特異値分解する.
3: 文書ベクトル行列 $\mathbf{D} = \mathbf{U}\mathbf{S}^{1/2}$, 単語ベクトル行列 $\mathbf{W} = \mathbf{V}^T\mathbf{S}^{1/2}$ を計算する.

図 5.34: 特異値分解による文書ベクトル (DocVec) の計算アルゴリズム.

```
0.0149 peachy          【スナップレポート】東京ガールズコレクション 2011
0.0149 movie-enter     食べて、で、映画を観れる『東京ごはん映画祭』が今
0.0149 peachy          【スナップレポート】東京ガールズコレクション 2011
0.0148 movie-enter     基礎から勉強しよう!初心者でもわかる「東京国際映
0.0144 movie-enter     三池監督が映画『一命』について「満島ひかりがいろ
0.0141 movie-enter     東京国際映画祭、『最強のふたり』がグランプリを受
```

ノート：tf.idf と単語の重みづけ

自然言語処理の多くの場面で, “日” “方法” のように意味の弱い語には低い重みを, “ペプチド” “厳密” のような意味の強い語には高い重みを与えたい場合があります. このための古典的な方法として, **tf.idf** というヒューリスティックが知られています.

前者のような言葉の特徴は, 「どんな文書にもまんべんなく現れる」ということでしょう. 対して, 後者のような言葉は特定のトピックと結びついており, 全体のごく一部の文書にしか現れないのが特徴です. よって, N 個の文書の中で, ある単語 w が 1 回以上出現した文書の数を $\mathrm{df}(w)$ とおくと (これを**文書頻度** (document frequency) といいます)[*49], N に対する割合 (文書確率)

$$p = \frac{\mathrm{df}(w)}{N}$$

が 1 に近いほど w の意味は弱く, 0 に近いほど意味が強いと考えられます. $\log 1 = 0$ ですから, この情報量 (式(2.63)) をとった

単語 w	$\mathrm{df}(w)/N$	$\mathrm{idf}(w)$			
日	0.7003	0.3563	ソニー	0.0300	3.5075
人	0.5640	0.5726	知識	0.0272	3.6028
年	0.5014	0.6904	芸能	0.0245	3.7081
的	0.4632	0.7696	夏休み	0.0191	3.9595
日本	0.3651	1.0075	手作り	0.0163	4.1136
時間	0.2371	1.4395	画質	0.0136	4.2959
使用	0.1253	2.0767	新曲	0.0109	4.5191
姿	0.0899	2.4089	真っ白	0.0082	4.8067
NTT	0.0790	2.5381	危機	0.0082	4.8067
説明	0.0654	2.7273	豪雨	0.0054	5.2122
Web	0.0327	3.4205	愛着	0.0054	5.2122

表 5.11: Livedoor コーパスから計算した, 単語 w の文書確率 $\mathrm{df}(w)/N$ と $\mathrm{idf}(w) = \log N/\mathrm{df}(w)$ (抜粋). “日” のような語は全体の 7 割の文書に出現しているため idf は小さく, 一方で “豪雨” のような語は全体の 1%未満の文書にしか出現していないため, idf が大きいことがわかります.

*49 $n(d,w)$ を使うと, 数学的には $\mathrm{df}(w) = \sum_{n=1}^{N} \mathbb{I}(n(d,w) > 0)$ と表すことができます.

$$\text{idf}\,(w) = -\log p = \log \frac{N}{\text{df}\,(w)} \tag{5.98}$$

を単語 w の意味的な重みと考えることができます. df が分母に現れることから, これを**逆文書頻度** (inverse document frequency, idf) といいます. 表 5.11 に, Livedoor コーパスから計算した単語の p と idf の例を示しました.

一方で, 特定の文書 d の中では, 単語 w の出現回数 $n(d,w)$ が大きいほど意味は強いでしょう. ただし, その影響は線形ではなく, 実質的な強さはその対数くらいだと考えられます (ウェーバー=フェヒナーの法則). つまり, 「ソニー」が 50 回出現したからといってその情報量が 50 倍あるわけではなく, 効果は $\log 50 \simeq 3.91$ 倍に比例する程度だということです. このままだと頻度 1 のとき $\log 1 = 0$ になってしまいますので, しばしば

$$\text{tf}\,(n) = 1 + \log n \tag{5.99}$$

のように定義されます. これを**用語頻度** (term frequency) とよびます. tf と idf の二つを組み合わせると, 頻度 $n(d,w)$ を

$$\text{tf}\,(n(d,w)) \cdot \text{idf}\,(w) = (1 + \log n(d,w)) \cdot \log \frac{N}{\text{df}\,(w)} \tag{5.100}$$

と変換すれば, 適切な重みづけになると考えられます. この単語重みづけを **tf.idf** といいます.

実際には対数の底に任意性があり, tf や idf の定義も上に示したものに限らず, いくつかのバリエーションがあります[35, §15.2.2]. tf.idf は有効に働くヒューリスティックですが, 自然言語処理のタスク全体としてこの重みが最適であるという保証はなく, 現代的には 5.2.1 節で説明した PMI や, 4.2.2 節で説明した SIF による重みづけの方が数学的な最適性があり, より効果的です.

5.5.2 単語ベクトル/文書ベクトルの解釈

こうして得られた文書ベクトルは高速に計算でき, 数学的に Word2Vec(Doc2Vec) と同じニューラル文書ベクトルとなっているため, 高い性能を持っています. 唯一の欠点は, K 個の次元が LDA のようにトピックとして解釈ができないということでしょう. たとえば, 上の実験で得られた単語ベクトルを並べた行列 $\mathbf{W}$ について, その 1 次元目, 2 次元目, …の値が大きい単語を求めると表 5.12 のようになり, ここには強い規則性は見出せそうにありません.

考えてみるとこれは当然で, 式(5.92)による行列分解は式(5.94)のように内積だけを問題にしているため, 空間全体を任意に回転しても, 図 5.35 のように 2 つのベクトルの間の内積は同じになるからです. 数学的には, ある直交行列 $\mathbf{R}$ をとって $\widetilde{\mathbf{D}} = \mathbf{DR}$, $\widetilde{\mathbf{W}} = \mathbf{WR}$ とベクトル全体を回転したとき, 式(5.92)の右辺は

$$\widetilde{\mathbf{D}}\widetilde{\mathbf{W}}^T = \mathbf{DR}(\mathbf{WR})^T = \mathbf{D}\underbrace{\mathbf{RR}^T}_{=\mathbf{I}}\mathbf{W}^T = \mathbf{DW}^T \tag{5.101}$$

となり, もとと等しくなります.

すなわち, 解釈のためには図 5.35 に示したように, 単語ベクトルや文書ベクトルがちょうど軸の近くに配置されるような回転 $\mathbf{R}$ を見つける必要があります. こうした方法として, 心理学ではバリマックス回転のような方法が知られ

次元 1		次元 2		次元 3		次元 4	
級	0.6090	自身	0.6138	Watch	0.7628	再生	0.7540
節電	0.5832	イベント	0.5650	Sports	0.6780	ホラー	0.5644
全	0.5279	クリスマス	0.5647	婚	0.5346	恐怖	0.5515
電力	0.5185	http	0.5261	活	0.5325	音楽	0.5425
シリーズ	0.4905	作っ	0.5027	答え	0.5069	デビュー	0.4848
AKB	0.4835	城	0.4955	音楽	0.5028	PC	0.4742
母親	0.4638	シーズン	0.4949	佐	0.4505	現象	0.4396
通話	0.4564	テレビ	0.4719	曲	0.4487	娘	0.4229
スマ	0.4509	婦	0.4476	学校	0.4450	ロンドン	0.4222
出展	0.4390	超	0.4430	枝	0.4409	YouTube	0.4201
家族	0.4374	家政	0.4391	調査	0.4366	必ず	0.4186
額	0.4268	www	0.4380	選手	0.4173	韓国	0.4184

表 5.12: Livedoor コーパスから DocVec で計算した単語ベクトルの各次元の値が大きい単語 (一部). もとのままでは, 各次元に明確な意味は見出せそうにありません.

ています[171]. しかし, これは言語のように数百次元以上にもなる高次元の場合にはあまりうまくいかず[172], 最近, 京都大学の下平らにより, **ICA** (独立成分分析) を適用することで解釈が容易な軸が, しかも言語横断的に見つかることが示されました[173]. ICA はデータを線形変換して, 各次元が可能な限り統計的に独立となるような座標軸を発見する方法です [*50]. たとえば図 5.36 では, PCA(主成分分析) ではデータの分散を最大にするように真ん中のような座標軸がとられてしまいますが, ICA ではデータの分布を独立な軸の積で説明する右のような座標軸を計算することができます.

「統計的に独立」とは, 2 章の式(2.21) でみたように, 確率変数 x と y の同時確率が $p(x,y)=p(x)p(y)$ のように周辺確率の積に分解できることでした. われわれの場合, たとえば式(5.92)で得られた, 単語ベクトルを並べた行列 $\mathbf{W}$ を行列 $\mathbf{A}$ を使って $\mathbf{S}=\mathbf{WA}$ と線形変換したとき, $\mathbf{S}$ の各行ベクトル $\mathbf{s}=(s_1,s_2,\cdots,s_K)$ について, 分解

$$p(s_1,s_2,\cdots,s_K)=p(s_1)p(s_2)\cdots p(s_K) \tag{5.102}$$

が可能な限り成り立つような行列 $\mathbf{A}$ を求めることになります [*51].

こうした $\mathbf{A}$ は, 機械学習の分野で ICA を定式化したフィンランドの Hyvärinen

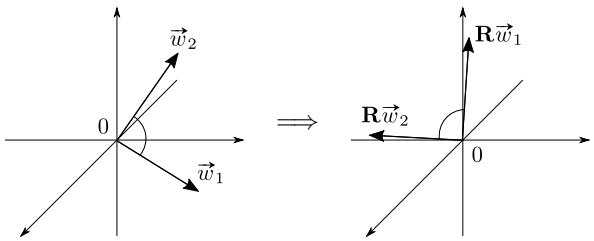


図 5.35: 単語ベクトルの回転. 単語ベクトルを直交行列 $\mathbf{R}$ で回転しても, 二つの単語ベクトルのなす角度は不変です. 適切な回転 $\mathbf{R}$ を求めることで, 単語ベクトルを軸に沿った形で, よりわかりやすく解釈することができます.

---

*50　157 ページで説明した白色化はデータを無相関にする方法ですが, 独立とは無相関を含んでおり, それより強い条件になります.

*51　ICA は実際には, 157 ページで説明したデータの白色化を行ってから $\mathbf{R}$ による回転を行うことと等価で, 白色化行列を $\mathbf{B}$ とおくと $\mathbf{A}=\mathbf{BR}$ になります.

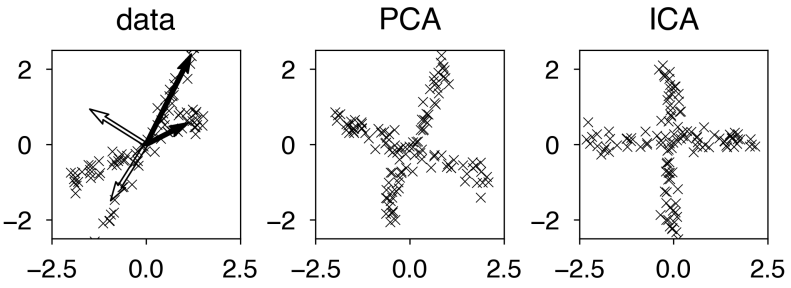


図 5.36: ICA の概要図. PCA(主成分分析) では白矢印のように, データ (×印) の分散を全体的に説明する軸がとられてしまいますが, ICA(独立成分分析) では黒矢印のように, データの分布を独立な軸の積として説明する軸が求められているのがわかります.

によるパッケージ `FastICA`[*52] で計算することができます. Python では,

```
from sklearn.decomposition import FastICA # ICA の計算
from scipy.stats import skew              # 歪度の計算
import numpy as np
def ica (X):
    X = X - np.mean (X, axis=0)
    analyzer = FastICA (whiten="arbitrary-variance")
    S = analyzer.fit_transform (X)
    A = analyzer.components_
    # sort by skewness
    N,D = S.shape
    skews = np.abs (skew (S, axis=0)) # 0=Gaussian
    index = map (lambda x: x[1], sorted (zip(skews, np.arange(D)),
                 key=lambda x: x[0], reverse=True)))
    return S[:,list(index)], A
```

で $\mathbf{S}$ および $\mathbf{A}$ を求めることができます. 多くの信号が混ざると, 中心極限定理によって分布はガウス分布に近づくことから, 逆に ICA で分解した軸では, 各次元の周辺分布は非ガウス的になります. その度合いは, 期待値 μ, 標準偏差 σ をもつ確率変数 X の**歪度**(わいど)

$$\delta = \mathbb{E}[(X-\mu)^3]/\sigma^3 \tag{5.103}$$

で表すことができます[174]. ガウス分布では δ は 0 で, $\pm$ になるほど左右の裾

*52 `http://research.ics.aalto.fi/ica/fastica/`

が広くなることが知られています. ICA は PCA と異なり, 独立性の高い軸から求まるとは限りませんので, 上のコードでは δ の絶対値を用いて, 非ガウス性の高い順に次元を並び換えています. Livedoor コーパスから計算した単語ベクトルについて, ICA で変換した各次元の値の大きい単語を表 5.13 に示しました. ほとんどトピックモデルのような, 解釈性の高い意味的な軸が求められていることがわかります.

ただしトピックモデルと異なり, これはベクトル空間の「軸」ですので, 負の方向も存在します [*53]. 表 5.14 に, 表 5.13 のいくつかの軸について値が負の単語を示しました. 単なる文書–単語の共起行列から得られたにもかかわらず, iPhone ⟷ Android, ビジネス ⟷ 悩み相談のような興味深い対立軸が教師なしで得られていることがわかります.

5.6　確率的潜在意味スケーリング (PLSS)

前節の ICA の結果から, 文書ベクトルおよび単語ベクトルの存在する埋め込み空間には, トピックモデルのトピックに対応するようなさまざまな「軸」が存在することがわかりました. 実際に Word2Vec (これは前節で主成分分析で求めたものと, 3.5.4 節で説明したように数学的には同じものでした) の空間には, ICA で見つかるものに限らず多くの意味的な軸が存在することがわかっています[175].

特に重要なのは, 表 5.14 でみたように多くの場合, この軸は正の方向と負の方向をもつ対立軸となっているということです [*54]. すると, 表 5.13 と表 5.14 で軸と向きが近い, または逆の単語を計算したように, 図 5.37 のようにこの軸上に文書ベクトルを射影し, テキストの性質をある軸に沿って 1 次元の ± の実数値として表すことができるでしょう. これを心理学の用語では, テキストの**尺度化** (scaling) といいます.

---

*53　空間全体をある軸に関して折り返してもベクトル間の関係は変わりませんので, 符号が正負どちらになるかに大きな意味はありません.

*53　`https://news.livedoor.com/article/detail/6029862/`

*54　ただし, Transformer で得られる埋め込み空間は注意機構のために複雑になり, こうした線形性があまり成り立たなくなっています.

次元 3		次元 5		次元 6		次元 10	
iPhone	0.0224	apps	0.0853	ビジネス	0.0180	等	0.0272
クリック	0.0185	google	0.0851	経営	0.0174	由里子	0.0269
ブック	0.0163	要件	0.0848	成功	0.0172	吉	0.0268
電子	0.0160	store	0.0847	キャリア	0.0164	愛	0.0265
術	0.0160	play	0.0842	オフィスエム	0.0155	サイト	0.0260
アップ	0.0152	ANDROID	0.0821	笑	0.0147	幸せ	0.0246
サイト	0.0150	details	0.0820	話し	0.0144	果たし	0.0243
携帯	0.0142	Play	0.0779	管理	0.0139	公式	0.0242
既報	0.0135	Google	0.0600	スキル	0.0132	務め	0.0234
背面	0.0134	Hisumi	0.0565	戦略	0.0129	篇	0.0233
ライフ	0.0123	以上	0.0421	力	0.0129	リアル	0.0224
iPad	0.0123	Android	0.0420	重要	0.0124	女優	0.0223

次元 15		次元 19		次元 28		次元 39	
高画質	0.0122	サッカー	0.2559	ロードショー	0.0428	スタイル	0.0218
デジタル	0.0105	代表	0.2401	全国	0.0425	オシャレ	0.0197
実現	0.0095	戦	0.2193	女優	0.0237	着	0.0171
操作	0.0093	試合	0.2175	決意	0.0206	ファッション	0.0170
ソニー	0.0084	杯	0.1897	土	0.0200	楽しむ	0.0166
ズーム	0.0082	W	0.1591	実力	0.0199	シンプル	0.0164
進化	0.0081	チーム	0.1532	最強	0.0198	ダイエット	0.0153
保存	0.0080	ゴール	0.1410	黄金	0.0194	味わい	0.0153
シーン	0.0080	日本	0.1280	絆	0.0193	体重	0.0151
新	0.0079	予選	0.1277	祭	0.0187	食事	0.0151
軽量	0.0079	選手	0.1261	ベルセルク	0.0186	誰	0.0147
レス	0.0079	リーグ	0.1239	TOHO	0.0184	運動	0.0145

表 5.13: ICA で変換した単語ベクトルの各次元の値が大きい単語 (抜粋). 次元は歪度の絶対値の大きい順に並んでいます. 表 5.12 と比べて, ほとんどトピックモデルのような, 意味的な軸が学習されていることがわかります. 次元 10 の意味については, 表 5.14 を参照してください.

テキストを分類するのではなく, 連続値で測る尺度化は実際のさまざまな場面において有効です. たとえば, 法案を保守 (右翼) か革新 (左翼) のどちらかに分類するのは簡単でも, 実際の法案は日本ではほとんど保守 (自民党) の側から提出されており, 問題はむしろ, その法案がどれくらい保守的なのか, ということでしょう. また, 住民へのアンケートやホテルの評価に書かれたテキストを肯定・否定に分類するのは簡単ですが, 重要なのはどの意見が強い賛成・反対であり, どの意見が軽い文句なのかを見極めることでしょう. 小説家のテキストからその分裂症気質を測りたいといった場合でも, 分裂症かどうかという二値分類

次元 3		次元 6		次元 10		次元 19	
apps	−0.2364	お答え	−0.2046	妖	−0.2595	フィギュア	−0.0405
store	−0.2363	辛口	−0.2041	ヶ	−0.2562	スケート	−0.0403
details	−0.2362	悩め	−0.2026	巻	−0.2316	選手権	−0.0364
play	−0.2300	説教	−0.2018	劇場	−0.1994	演技	−0.0312
google	−0.2299	尽き	−0.1938	勅使河原	−0.1992	克也	−0.0268
要件	−0.2225	早	−0.1856	妖怪	−0.1967	浅田	−0.0265
ANDROID	−0.2187	面白く	−0.1820	栄華	−0.1961	真央	−0.0262
id	−0.2119	姉妹	−0.1689	仙人	−0.1900	メダリスト	−0.0262
Play	−0.2006	悩み	−0.1680	お嬢様	−0.1695	野村	−0.0255
com	−0.1913	type	−0.1663	ネコ	−0.1589	ノム	−0.0246
Store	−0.1798	若手	−0.1650	話	−0.1497	バンクーバー	−0.0231
カテゴリ	−0.1682	瞬時	−0.1596	次	−0.1430	野球	−0.0225

表 5.14: ICA で変換した単語ベクトルの各次元の値が小さい単語 (抜粋). 表 5.13 と見比べると, 次元 3 では iPhone の「反対の概念」として Android が, 次元 6 ではビジネスについてお悩み相談が, 次元 19 ではサッカーについてフィギュアスケートが得られていることがわかります. 次元 10 は元データの Livedoor ブログでの “いちおう妖ヶ劇場” という連載記事 *54 に関連している軸で, 負の方にだけ意味を持っています.

にはあまり意味がなく (小説家の多くは分裂症気質のため), 分裂症気質の強さやその変化が主な興味の対象になると考えられます. *55

こうした尺度化を行うもっとも簡単な方法は, 単語ベクトルや文書ベクトルの与えられた軸への近さを, 前節のように cos 距離を計算して求めることです. しかし, 3 章の図 3.37 で示したように, 埋め込みベクトルの間の cos 距離は 0 を中心には分布せず, 値も $[-1, 1]$ の中の一部しかとりません. 別の言い方をすれば, cos 距離はあくまで後付けの値であり, ある軸に関係しないベクトルの値が 0 となる尺度としては設計されていない, ということです.

5.6.1 Wordfish

特に, 社会科学においてテキストの尺度化は重要な問題であるため, 政治学方法論 (Political methodology) の分野では, Slapin らが 2008 年に政党などの連続した極性の時間変化を求める Wordfish という方法を提案しました[177].

*55 図 5.5 でもみたように, 分類モデルは尤度を上げるために極性を ±1 のどちらかに寄せる傾向があり, SVM のようなベクトル空間の分類器でも, 分離平面からの距離がクラスに対応する尺度とは限りません[176].

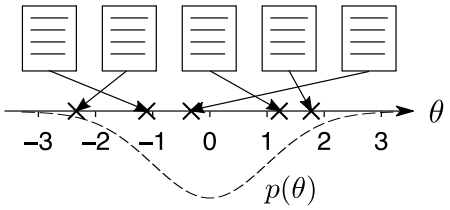


図 5.37: テキストの尺度化と潜在スケール (尺度) θ. 各文書に対して, 測りたい軸に沿った実数値 $\theta \in \mathbb{R}$ を推定します. 項目反応理論に従い, θ は標準正規分布 $\theta \sim \mathcal{N}(0,1)$ に従う潜在変数だと考えます.

y_{itv} を政党 i が時刻 t のテキスト (たとえば選挙時の公約) で単語 v を使った回数とすると, Wordfish はデータ $Y = \{y_{itv}\}$ $(i = 1, \cdots, I, t = 1, \cdots, T, v = 1, \cdots, V)$ の確率を, 次のようにポアソン分布 $\mathrm{Po}(y|\lambda)$ でモデル化します.

$$\left\{ \begin{array}{l} p(Y) = \displaystyle\prod_{i=1}^{I} \prod_{t=1}^{T} \prod_{v=1}^{V} \mathrm{Po}(y_{itv}|\lambda_{itv}) \\ \lambda_{itv} \ = \exp\big(\alpha_{it} + \beta_v + \phi_v \cdot \theta_{it}\big) \end{array} \right. \tag{5.104}$$

ここで α_{it} はテキスト it での固定効果 (ベースライン), β_v は単語 v の固定効果で, 興味があるのは単語 v の極性軸上での位置 $\phi_v \in \mathbb{R}$ と, 政党 i の時刻 t での潜在位置 [*56]$\theta_{it} \in \mathbb{R}$ です.

式(5.104)は, 頻度 y_{itv} はポアソン分布 $\mathrm{Po}(\lambda)$ に従い, その期待値はテキスト it と単語 v で決まるベースライン $\alpha_{it} + \beta_v$ を, 政党 i の時刻 t での極性 θ_{it} と単語の持つ極性 ϕ_v で上下して決まる, ということを意味しています. $\theta > 0$ が右翼, $\theta < 0$ が左翼を表すとしたとき, 政党の位置と単語の符号が一致する, すなわち $\theta_{it} > 0$ かつ $\phi_v > 0$ (たとえば $v =$ “軍備”), または $\theta_{it} < 0$ かつ $\phi_v < 0$ (たとえば $v =$ “社会保障”) のとき y_{itv} の期待値 λ_{itv} は大きくなり, 符号が逆ならば小さくなります. $\{\alpha, \beta, \theta, \phi\}$ はすべて未知のため, 推定には 241 ページで学習した EM アルゴリズムを用い, 適切な初期値から始めて, (α, θ) の推定と (β, ϕ) の推定を反復します. こうして求めたドイツの各政党の θ_{it} の時間変化を, 図 5.38 に示しました.

∗56 これを政治学の分野では, 理想点 (ideal point) といいます.

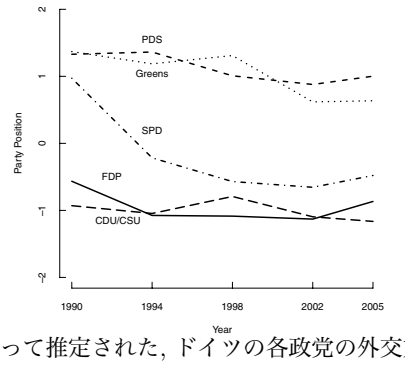


図 5.38: Wordfish によって推定された, ドイツの各政党の外交方針 θ_{it} の時間変化の例 (1990–2005) ([177]より引用). 縦軸の θ は平均が 0 の潜在変数になっています.

5.6.2　確率的潜在意味スケーリング (PLSS)

Wordfish は与えられたテキストに対する系統的な統計モデルですが, 完全な教師なし学習のため, これによって得られる θ の極性が分析の目的と一致している, という保証はありません. 適切な結果を得るためには, 入力のテキストを注意深く選ぶ必要がありますが, そのための客観的な方法は示されておらず, また学習データの単語頻度に対するポアソン分布はテキストの長さに暗黙に依存するため, 新しいテキストでの θ を計算できないという問題もあります.

そこで, 心理統計学における**項目反応理論** (Item Response Theory, IRT) を参考に, 潜在的な極性 $\theta \in \mathbb{R}$ をもつテキストで単語 $v \in \{1, \cdots, V\}$ が出現する確率を, 次のように多項分布でモデル化します.

$$
\begin{aligned}
p(v \,|\, \theta, \phi) &\propto p(v) \exp\left(\theta \cdot \phi_v\right) \\
&= \frac{\exp\left(\log p(v) + \theta \cdot \phi_v\right)}{\sum_{v=1}^{V} \exp\left(\log p(v) + \theta \cdot \phi_v\right)}
\end{aligned}
\tag{5.105}
$$

ここで $\phi_v \in \mathbb{R}$ は式(5.104)の Wordfish の場合と同様に, 単語 v の「極性」を表すパラメータで, θ は 0 を中心とした標準正規分布

$$
\theta \sim \mathcal{N}(0, 1) \tag{5.106}
$$

に従うとします.

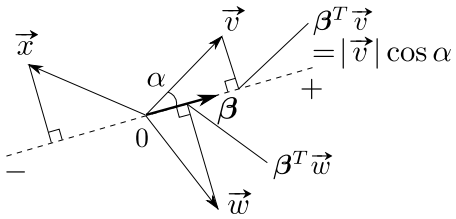


図 5.39: 方向ベクトル $\boldsymbol{\beta}$ による単語の極性の計算. $\boldsymbol{\beta}$ との内積, すなわち $\boldsymbol{\beta}$ 上への正射影の長さで単語の極性 $\phi_v = \boldsymbol{\beta}^T\vec{v}$ を計算します. ここで $|\boldsymbol{\beta}|=1$ で, $\vec{v}, \vec{w}, \vec{x}$ は単語ベクトルを表します. 単語ベクトルが $\boldsymbol{\beta}$ と逆方向の場合は, 極性はマイナスになります.

式(5.105)より, このモデルでは Wordfish と同様に単語 v の確率は ϕ_v と θ の正負と大きさが一致すれば事前確率 $p(v)$ より高くなり, 逆になれば低くなる, というモデルになっています. $p(v)$ はコーパスから容易に計算できるため, このモデルは IRT の多項分布化, あるいは式(5.105)で表される多値ロジスティック回帰において, 説明変数 θ も回帰係数 ϕ も未知の場合の教師なし学習とみなすことができます.

このとき, テキスト d の確率は単語 v のテキスト内での頻度を n_{dv} とおくと

$$p(d|\theta,\phi) = \prod_{v=1}^{V} p(v|\theta,\phi)^{n_{dv}} \tag{5.107}$$

と書けますから, D 個のテキストからなるコーパス $\mathcal{D}$ 全体の確率は

$$p(\mathcal{D}|\Theta,\phi) = \prod_{d=1}^{D}\prod_{v=1}^{V} p(v|\theta_d,\phi)^{n_{dv}} \tag{5.108}$$

と表されます. Wordfish と同様に, 事前確率 (5.106) の下で式(5.108)を最大化するパラメータ $\Theta=\{\theta_1,\cdots,\theta_D\}$ および $\phi_1,\cdots,\phi_V$ を MCMC 法や EM アルゴリズムなどによって計算することができます.[*57]

ただし, こうするとたとえ単語 v と w が意味的に関係が深くても, ϕ_v と ϕ_w は別のパラメータとして推定しなければならないという問題があります. たとえば $\phi_{\text{good}} > 0$ と推定できても, これは $\phi_{\text{excellent}}$ や ϕ_{well} とは無関係で, excellent

*57 これは多項分布によるモデルのため, ポアソン分布による Wordfish と異なり, パラメータが学習テキストの長さに依存せず, 新しいテキストについても適用することができるという特徴があります.

や well がコーパスに現れなければ, まったく学習することができません.

そこで, $\phi_1, \cdots, \phi_V$ を独立に学習する代わりに, 与えられたコーパスあるいは一般的なコーパスから事前に Word2Vec や GloVe などで学習された K 次元のニューラル単語ベクトル $\vec{v}$ を用いて, ϕ_v を

$$\phi_v = \boldsymbol{\beta}^T \vec{v} \qquad (\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_K)^T) \tag{5.109}$$

とモデル化します. この後で説明するように $\boldsymbol{\beta}$ の長さは 1 に正規化しますから, これは図 5.39 のように, 各単語ベクトル $\vec{v}$ と $\boldsymbol{\beta}$ のなす角を α としたとき, 単語の極性を $\vec{v}$ の $\boldsymbol{\beta}$ 上への正射影の長さ $\boldsymbol{\beta}^T \vec{v} = |\boldsymbol{\beta}||\vec{v}| \cos\alpha = |\vec{v}| \cos\alpha$ で「測って」いることに相当します. これにより, V 個の独立な $\phi_1, \cdots, \phi_V$ を求めるかわりに, K 次元の方向ベクトル $\boldsymbol{\beta}$ を一つだけ推定すればよいことになります. このとき, 式(5.105)は $\ell_v = \log p(v)$ とおけば,

$$p(v|\theta, \boldsymbol{\beta}) = \frac{\exp(\ell_v + \theta \cdot \boldsymbol{\beta}^T \vec{v})}{\sum_{v=1}^{V} \exp(\ell_v + \theta \cdot \boldsymbol{\beta}^T \vec{v})} \tag{5.110}$$

と表すことができます. 筆者が開発したこの方法は, 政治学分野で提案された, 281 ページの LSI をベースにしたベクトル空間におけるアルゴリズムである LSS (Latent Semantic Scaling) [178]の確率化ともみなすことができるため, 確率的 LSS, **PLSS** (Probabilistic Latent Semantic Scaling) [179]とよぶことにします.

この $\boldsymbol{\beta}$ は, 単語埋め込みベクトルの空間において “良い–悪い”, “右翼–左翼” といった θ の極性を与える「極性軸」, あるいは「意味方向」を表しています. $\boldsymbol{\beta}$ は完全に教師なしで学習することもできますが, 5.6.1 節で述べたように, そうして得られた $\boldsymbol{\beta}$ が分析の目的と一致しているとは限りません. そこで, PLSS では表 5.15 のように, 正例および負例として与える少数の極性語辞書から, 単語ベクトルを用いて $\boldsymbol{\beta}$ を次のように計算します. なお, $\boldsymbol{\beta}$ のノルムは 1 に正規

表 5.15: 表 5.4 の WRIME コーパスの極性語を参考に作成した, 標準的な極性辞書. + が正例を, – が負例を表します.

+	最高 嬉しい 楽しい 楽しみ 可愛い きれい しあわせ 好き かわいい
–	悪い 悲しい 寂しい 嫌い 怒り ない つらい 無理 しんどい めんどう

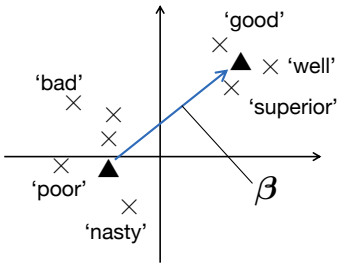


図 5.40: 極性辞書と単語ベクトルによる $\boldsymbol{\beta}$ の計算.

化します.

$$\boldsymbol{\beta} \propto \left(\frac{1}{|S_+|} \sum_{v \in S_+} \vec{v} - \frac{1}{|S_-|} \sum_{w \in S_-} \vec{w} \right) \tag{5.111}$$

ここで S_+ は極性辞書のうち正の語の集合, S_- は負の語の集合です. 式(5.111)は単純に, $\boldsymbol{\beta}$ として負の単語ベクトルの平均から, 正の単語ベクトルの平均へ向かう方向を取ることを表しています. この様子を図 5.40 に示しました. 極性辞書としては, 表 5.15 に示した標準的な極性辞書のように, ごく少数の単語のリストがあれば動きます. なお, 正例・負例の単語が「最も極端な語」である必要はなく, 意味方向のみが合っていればよいことに注意してください.

ここで「正例」「負例」とは必ずしも感情的な正負と関係していなくてもよく, 「右翼–左翼」「都会–田舎」「東洋–西洋」のように, 任意の意味的な軸を扱うことができます. ドイツの Schütze らは, 単語埋め込みの空間に実際にこうした, 与えられたタスクに関係する低次元の部分空間が存在することを発見し, これを超密埋め込み (Ultradense embedding) と呼んでいます[175]. 超密埋め込みは, 最も単純には, この場合のように 1 次元の部分空間となります. ただし, 予備実験でこの超密埋め込みを行列の固有ベクトルとして求める DensRay [180] を使用したところ, 式(5.111)と比べて明らかにノイズの多い方向となったため, PLSS では単純な式(5.111)を採用することとしました.[*58]

---

*58 これは, DenseRay の目的関数が, 極性辞書で単語 v の属する極性を $s(v)$ としたとき, 行列 $\mathbf{A} = \frac{1}{|S_+|} \sum_{\substack{(v,w):\\ s(v)=s(w)}} (\vec{v}-\vec{w})(\vec{v}-\vec{w})^T - \frac{1}{|S_-|} \sum_{\substack{(v,w):\\ s(v)\neq s(w)}} (\vec{v}-\vec{w})(\vec{v}-\vec{w})^T$ の固有ベクトルの計

表 5.16 に, 表 5.15 の標準的な極性辞書と, 後で説明する日本語 Wikipedia 記事から Word2Vec の方法で学習した $K=100$ 次元の単語ベクトルを用いて計算した単語の極性 $\phi_v = \boldsymbol{\beta}^T \vec{v}$ を示しました. 非常に少ない教師データにもかかわらず, 肯定的な単語および否定的な単語が, その強さとともに連続的に取り出せている様子がわかります.

極性辞書を用いる場合は, PLSS は単語ベクトルの計算以外にパラメータの学習を必要としません. 式(5.107)から, テキスト d と極性 θ の同時確率は

$$p(d,\theta) = \prod_{v=1}^{V} p(v\,|\,\theta,\boldsymbol{\beta})^{n_{dv}} \cdot p(\theta) \tag{5.112}$$

$$= \prod_{v=1}^{V} \left(\frac{\exp\left(\ell_v + \theta \cdot \boldsymbol{\beta}^T \vec{v}\right)}{\sum_{v=1}^{V} \exp\left(\ell_v + \theta \cdot \boldsymbol{\beta}^T \vec{v}\right)} \right)^{n_{dv}} \cdot \mathcal{N}(\theta \tag{5.113}$$

$$|0,1)$$

となり, これを最大にするテキストの潜在的な極性 θ の MAP 解は, 1 次元の最適化で容易に計算することができます.

PLSS の実験　227 ページで使った WRIME コーパスのツイートの感情極性を, PLSS で分析してみましょう. 本書では基本的に Python を使用していますが, 政治学方法論分野では R が普及しているため, 本節では R で実装を行っています. 以下で使用しているスクリプトは, すべてサポートサイトからダウンロードすることができます.

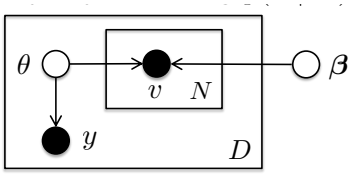


図 5.41: $\boldsymbol{\beta}$ を推定するための教師データのグラフィカルモデル. $\boldsymbol{\beta}$ だけでなく θ も未知のため, θ については周辺化を行って積分消去することで $\boldsymbol{\beta}$ を推定します.

算に帰着され, 正例と負例の内部およびその間のペアを結ぶ個別のベクトルの和になっていることが原因だと考えられます. 式(5.111)と異なり, この定式化では S_+, S_- の中で和をとることでノイズを消す作用が働かず, v, w の選択に起因するノイズが $\mathbf{A}$ に残ってしまうからです.

表 5.16: Wikipedia のコーパスから事前学習した単語ベクトルと, 表 5.15 の極性辞書を用いて計算した WRIME コーパスでの単語の極性 $\phi_v = \boldsymbol{\beta}^T \vec{v}$.

(a) 極性 $\phi_v > 0$ の上位語

ミッフィー	1.5639	星座	1.1436
チェキ	1.4430	スク	1.1274
キラキラ	1.3147	メゾン	1.1142
♪	1.2942	プレゼント	1.0828
かわいい	1.2609	リゾート	1.0742
アメニティ	1.2367	展	1.0505
クロワッサン	1.2283	土産	1.0492
たのしい	1.2268	ミント	1.0474
ミルキー	1.1964	マリオ	1.0255
しあわせ	1.1900	満喫	1.0134
ニベア	1.1663	美容	1.0097
水着	1.1637	hello	1.0028
応募	1.1579	ほっと	0.9945
アニメイト	1.1544	香菜	0.9894
♡	1.1466	コス	0.9867

(b) 極性 $\phi_v < 0$ の下位語

苦しん	−1.8143	許せ	−1.4058
耐え	−1.7748	察し	−1.3846
生じ	−1.7656	しまっ	−1.3837
治まっ	−1.7211	デメリット	−1.3749
予断	−1.6880	しまう	−1.3586
断ち切ら	−1.6177	起き	−1.3475
原因	−1.6022	断ち切っ	−1.3388
ざる	−1.5514	後悔	−1.3380
容赦	−1.5478	きれ	−1.3336
断ち切れ	−1.5090	症状	−1.3309
吐き	−1.4649	不安	−1.3211
ひどく	−1.4551	許さ	−1.3163
訴え	−1.4377	気付か	−1.3146
隙	−1.4274	村八分	−1.3140
困憊	−1.4211	雑言	−1.3083

情報が少ないと精密な分析ができないため, ここでは `wrime.test` の中で 20 単語以上のツイートを対象にすることにして, 次のようにしてテキストだけを抽出します. 結果は, 302 ツイートになりました. [*59]

```
% awk 'NF>20' wrime.test | cut -f 2 > wrime.txt
% head wrime.txt
先輩 の 息子 が 、 まじ 一瞬 だ けど つかまら ず 立つ ..
わたし の 伝え 方 が おかしい の か ? なぜ 、 全て や..
「 落ちる 」 場面 や 、 走っ て も 走っ て も 前 に 進..
Shift + win キー + S キー で 範囲 指定 の 画面 キャプ..
```

`wrime.txt` はこのように, 1 行に 1 文書 (1 ツイート) の単語が並んだ, 単なるテキストファイルです. PLSS は正解ラベルを必要としませんが, 検証のために取り出しておきましょう.

```
% awk 'NF>20' wrime.test | cut -f 1 > wrime.label
```

*59 `cut` は, タブで区切られたテキストの指定された番号のフィールドを表示する, Unix の標準コマンドです. `awk 'NF>20'` とは, 空白で区切られたフィールド数が 20 より大きいとき標準アクション (その行を表示) を行う, という awk (67 ページ) のコマンドです.

単語ベクトルの種類は任意ですが[*60], ここでは Wikipedia の記事から作成された単語ベクトルの公開データ Wikipedia2Vec [181] [*61] から, 100 次元の日本語単語ベクトルを使うことにします. サイトから “Japanese 100d(txt)” のベクトルを入手し, bzip2 で解凍します.

```
% bzip2 -dc jawiki_20180420_100d.txt.bz2 > jawiki.vec
```

分析に使う極性辞書は, 表 5.15 のものを用います. これは +,− の順に “ラベル<TAB>単語..” というフォーマットで, サポートサイトに posneg.ja という名前で置いてあります.

```
% cat posneg.ja
positive        最高 嬉しい 楽しい 楽しみ 可愛い きれ..
negative        悪い 悲しい 寂しい 嫌い 怒り ない つら..
```

この上で, 次のように plss を実行してツイートの極性を計算します. 使い方は plss の中を読むか, このスクリプトを単独で実行してみてください. 以下では R でテキストを扱うパッケージ quanteda[*62] および関連ライブラリがインストールされていることを前提にしていますので, エラーが出た場合は install.packages でインストールしておいてください.

```
% plss wrime.txt posneg.ja jawiki.vec output
⇒ preparing word vectors..
  total 2762 words selected.
  running PLSS..
  loading wordvectors from /tmp/wordvec-47752.vec.. done.
  preparing data.. done.
  documents = 302, vocabulary = 2762
  computing theta..
  computing 302/302.. done.
  theta written to output.theta.
  phi   written to output.phi.
```

保存された output.theta が各ツイートの極性 $\theta \sim \mathcal{N}(0,1)$, output.phi は表 5.16 に示した, 内部で計算した単語の極性 ϕ_v です. (推定に用いなかった) 正

*60 対象となるテキスト自体から, 3 章で紹介した方法で単語ベクトルを学習することも原理的には可能ですが, これには通常, かなり大量のテキストを必要とします.

*61 https://wikipedia2vec.github.io/wikipedia2vec/

*62 http://quanteda.io/ ; 執筆時は quanteda 4.0 の上で実装しています.

解のラベルおよびツイート本文とともに表示してみましょう. 次のスクリプト theta.sh を実行すると, ランダムな 20 ツイートを極性 θ の値でソートして表示します.

```
    % theta.sh output.theta wrime.label wrime.txt
⇒ positive   1.8362  あー (∗´∇`) キセキたちが私を萌殺そうとす..
  positive   0.9879  ぐっどこんでぃしょん。心も頭もクリア。秋分..
  positive   0.9121  やっぱ神席だった。キラー T 細胞さんってマチ..
  positive   0.7518  3 週間ぶりに卓球してきました♪高校生の時の..
  positive   0.7212  ソフトバンク、楽天、元 zozo の前澤さんなど..
  negative   0.6797  あっ、ぶんぐ博の荷物忘れた…。今日、先生に..
  positive   0.6390  高級牛乳買ってから寝る前に牛乳を 1 杯飲むよ..
  positive   0.5202  今日発売のじょーちゃん載ってる雑誌さすがに..
  positive   0.3314  同席のご夫婦さんに JAW さんファン歴何年ですか..
  positive   0.0837  早速遊びすぎて電池無くなったよ!家の中って被..
  positive  -0.0546  日々幸せだなぁって思ってるんだけど、今日ほん..
  negative  -0.0575  あ、明日大阪に行ったら接触確認アプリの働きが..
  negative  -0.2291  ほわ○ぷワナビとか借○玉ワナビ、大抵頭 (特に)..
  positive  -0.4468  うおー肩と腰が痛いわーごみ出しに行かなきゃー..
  negative  -0.6453  とある友人の言動が気になって仕方がない時があ..
  negative  -0.6546  気にかけてくれる素晴らしい上司に恵まれて、な..
  negative  -0.6990  どうしよう。来年行く旅行のことについて考えよ..
  negative  -0.7387  うちの母親でもここまでやったことはないかなあ..
  negative  -1.0035  体は疲労困憊なのに妙な緊張で眠れぬ今日もぐっ..
  negative  -1.1121  4 時過ぎまで眠れず昨日打ち合わせた仕事モヤ..
```

このように, PLSS ではごく少数の極性語辞書だけで, テキストをその軸に沿って極性の強さに応じた連続値で並べることができ, 陽性–陰性の場合は人手で付与した極性とも, ほぼ一致していることがわかります.

極性辞書は陽性–陰性だけでなく, 任意の軸で作ることができます. 図 5.42 では, 右翼–左翼の政治的立場を表す (a) の極性語辞書を用いて, 5.1 節の Livedoor コーパスの “topic-news” カテゴリーの記事を解析した様子を (b) に示しました. 記事にはもちろん, 右翼–左翼の軸に関するラベルはありませんが, 確かにこの極性軸に沿って文書を並べることができることがわかります.

5.6.3* PLSS の半教師あり学習

上では θ の極性を表す基準として LSS と同様に少量の極性辞書を用いましたが, こうした辞書が分析対象について自明に作成できるとは限りません. 例え

positive	保守 国家 伝統 経済 秩序 軍事 成長 資本
negative	平等 多様性 環境 平和 福祉 権利 労働

(a) 右翼–左翼の政治的立場を表す極性語辞書.

1.8022	「富士山をしろ」日本の軍事誌の内容に中国ネットユーザー..
1.6412	熊本の“政治家らしからぬ”経歴が話題 25 日に行われた熊..
1.4820	2011 年の日本の地震図に「すぎる」2011 年 1 月 1 日 00:00..
1.3375	富士山近郊でする地震に不安の声相次ぐ 1 月 29 日に発生し..
1.2284	中華圏で大ブレイクした“日本作り”19 日に「日経 NET」に..
1.1621	好きな女子アナランキングにネット騒然 9 日、STYLE が行っ..
1.1586	北朝鮮が´´ ミサイル´´ 発射も 1 分で落下 13 日、TBS を含..
1.1239	世界で有名な日本人ベスト 3 に意外な人物が続出 16 日放送..
:	:
-0.9924	猫ひろし、日本国籍再取得にも「当たり前だ」の声法律相談..
-1.0196	死刑廃止論者がネット掲示板でに 29 日、ダイヤモンドオン..
-1.0341	学校を休んでディズニーランドへは、ありかしかり、費、読..
-1.0834	生活保護受給をめぐり物議を醸す河本親子に法的措置の可能..
-1.1430	見知らぬ団体が勝手にハロプロの YouTube 公式チャンネルを..
-1.2003	東京都がまたマンガ・アニメ規制の動きか東京都が男女平等..
-1.2050	河本の姉が片山さつき氏に「後で謝ることになる」ネットで..
-1.4021	生活保護のイメージ悪化に抗議も、反論の声多数 30 日、生活..

(b) 解析した θ の上位および下位 8 個の記事.

図 5.42: PLSS による Livedoor ニュースコーパスの解析. “topic-news” のラベルをもつ 770 個の記事の内容を, (a) の極性語辞書を用いて分析した結果を (b) に示しました. 右寄りな記事にはより高い θ が, 左寄りな記事にはより低い θ が推定されています.

ば, 欧州において移民労働者についての賛成派と反対派を特徴づけるキーワードが, 分析前から明らかとは限らないからです.

しかし, そうした場合でも典型的な「正例」のテキストと「負例」のテキストは示せる場合が多いと考えられます. 直感的には, それぞれのテキストに共通して現れる単語 (単語ベクトル) から, 間接的に単語の極性が導かれるはずです. コーパスのうち, こうした極性が既知のテキストの集合を X_ℓ, それらへの 1/0 のラベルを Y_ℓ とすると, 各テキストとそのラベル $(y, d) \in (Y_\ell, X_\ell)$ について,

$$p(y, d, \theta, \boldsymbol{\beta}) = p(y|\theta) \prod_{v=1}^{V} p(v|\theta, \boldsymbol{\beta})^{n_{dv}} \cdot p(\theta)p(\boldsymbol{\beta}) \tag{5.114}$$

を最大化する $\boldsymbol{\beta}$ を求めることを考えましょう. ここで第 1 項はロジスティック

回帰

$$p(y=1|\theta) = \sigma(\theta) = \frac{1}{1+e^{-\theta}} \tag{5.115}$$

です. このグラフィカルモデルを図 5.41 に示しました. θ が大きい, あるいは小さい方が第 1 項の教師データに対する識別モデルの尤度が高くなりますが, 逆に第 2 項の単語の生成確率が下がる可能性があるため, 両者のトレードオフで θ が決まり, それによって式(5.112)から $\boldsymbol{\beta}$ が決まることになります.

ここで問題なのは, 回帰パラメータ $\boldsymbol{\beta}$ だけでなく, テキストの潜在的な極性 θ も未知なことです. θ および $\boldsymbol{\beta}$ を同時に最適化することも可能ですが, 心理統計学においてこうした同時推定は一致性を持たないことが知られています[182]. また, 二値ラベルの y という弱い教師情報からは θ は一意には決まらず, θ を学習時に点推定することは教師データへの過学習をもたらす可能性があります.

適応的ガウス–エルミート求積による解法 そこで, θ を推定する代わりにモデルから積分消去し,

$$\begin{aligned} p(y,d,\boldsymbol{\beta}) &= \int_{-\infty}^{\infty} p(y,d,\theta,\boldsymbol{\beta})d\theta \\ &= \int_{-\infty}^{\infty} p(y|\theta) \prod_{v=1}^{V} p(v|\theta,\boldsymbol{\beta})^{n_{dv}}\, p(\theta)d\theta \cdot p(\boldsymbol{\beta}) \end{aligned} \tag{5.116}$$

を $\boldsymbol{\beta}$ について最適化することを考えます.[*63] $p(\theta)$ は標準正規分布 $\mathcal{N}(0,1)$ ですから, 式(5.116) の形のガウス分布に関する積分はガウス–エルミート求積 [*64] と呼ばれる方法で, きわめて正確に数値的に求めることができます.

ガウス–エルミート求積では, 関数空間での直交多項式を用いて, 関数 $f(x)$ の e^{-x^2} に関する積分を次の形で高精度に近似します.

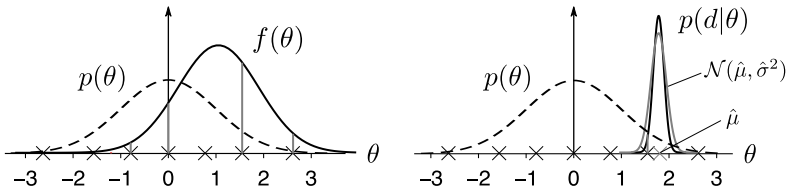
$$\int_{-\infty}^{\infty} f(x)e^{-x^2}dx \simeq \sum_{i=1}^{H} w_i f(x_i) \tag{5.117}$$

ここで $\mathbf{x} = (x_1, \cdots, x_H)$ は分点 (abscissa) と呼ばれる座標, $\mathbf{w} = (w_1, \cdots, w_H)$ は

---

*63 Hamiltonian MCMC 法[183]を用いた $\boldsymbol{\beta}$ のベイズ推定も検討しましたが, MAP 推定を用いる方が安定した結果となりました.

*64 求積 (quadrature) とは, 定積分の値を数値的に求めることです.



(a) 一般の関数 $f(\theta)$ の場合　(b) テキストの尤度 $p(d|\theta)$ の場合

図 5.43: 適応的なガウス–エルミート求積. 一般の関数 $f(\theta)$ の期待値 $\int f(\theta)p(\theta)d\theta$ は, ×印で示した分点 x_i での $f(\theta)$ と重み w_i から高精度に計算することができますが, テキストの場合は尤度 $p(d|\theta)$ が一部の θ に集中するため, 変数変換を行って MAP 解 $\widehat{\mu}$ の周辺で適応的なガウス–エルミート求積を行い, 期待値 $\int p(d|\theta)p(\theta)d\theta$ を計算します.

対応する重みで, $H=9$ のとき (本書の例では $H=20$ としました) は

$$\begin{aligned}\mathbf{x} &= (-3.191,\ -2.267,\ -1.469,\ -0.724,\ 0,\ 0.724,\ 1.469,\ 2.267,\ 3.191),\\ \mathbf{w} &= (0.00004,\ 0.005,\ 0.088,\ 0.433,\ 0.720,\ 0.433,\ 0.088,\ 0.005,\ 0.00004)\end{aligned}$$

です. これらの値は, 標準的なガウス–エルミート求積のパッケージで計算することができます.[*65] 式(5.117)は e^{-x^2} についての積分なので, $e^{-x^2}=e^{-\theta^2/2}$ すなわち $\theta=\sqrt{2}x$ とおけば, 変数変換により

(5.118)
$$\int_{-\infty}^{\infty} f(\theta)\mathcal{N}(\theta|0,1)d\theta = \int_{-\infty}^{\infty} f(\theta)\frac{1}{\sqrt{2\pi}}e^{-\frac{\theta^2}{2}}d\theta \simeq \frac{1}{\sqrt{\pi}}\sum_{i=1}^{H} w_i f(\sqrt{2}x_i)$$

と計算することができます.

ただし, 式(5.118)による積分は, そのままでは非常に効率が悪くなります. 一般にテキストは多くの単語を含むため, θ の事後分布はある値 $\widehat{\theta}$ の近くに集中しており, 式(5.118)ではほとんどの分点での尤度が 0 になってしまうからです. そこで, [184]の方法を用いて, 積分を $\widehat{\theta}$ の周りで実行することにします. θ の事後分布を近似する平均 $\mu=\widehat{\theta}$ と分散 σ^2 は二分探索と二階差分により容易に求めることができますので, まず, $\phi=\mu+\sigma\theta$ と変数変換すると, 簡単な計算により

∗65　Python では `numpy.polynomial.hermite.hermgauss` で, R では `gaussquad` パッケージの `hermite.h.quadrature.rules` で計算できます.

$$\int_{-\infty}^{\infty} f(\theta)\mathcal{N}(\theta\,|\,\mu,\sigma^2)d\theta \simeq \frac{1}{\sqrt{\pi}}\sum_{i=1}^{H} w_i f(\mu+\sqrt{2}\sigma x_i) \tag{5.119}$$

であることがわかります. このとき, 求める積分を次のように変形します.

$$I = \int_{-\infty}^{\infty} f(\theta)\mathcal{N}(\theta\,|\,0,1)d\theta = \int_{-\infty}^{\infty} \underbrace{f(\theta)\,\frac{\mathcal{N}(\theta\,|\,0,1)}{\mathcal{N}(\theta\,|\,\mu,\sigma^2)}}_{h(\theta)}\mathcal{N}(\theta\,|\,\mu,\sigma^2)d\theta \tag{5.120}$$

式(5.120)の最初の 2 項を $h(\theta)$ とおけば, 式(5.119)より

$$I \simeq \frac{1}{\sqrt{\pi}}\sum_{i=1}^{H} w_i h(\mu+\sqrt{2}\sigma x_i) \tag{5.121}$$

と, $\mathcal{N}(\theta\,|\,\mu,\sigma^2)$ についての適応的な積分で置き換えて求めることができます.

われわれの場合, 求めたい積分は式(5.116)でしたから, 対数で計算するために

$$\ell(\theta) = \log p(y\,|\,\theta) + \sum_{v=1}^{V} n_{dv}\log p(v\,|\,\theta,\boldsymbol{\beta}) \tag{5.122}$$

と定義すれば, $h(\theta)$ は

(5.123)

$$h(\theta) = e^{\ell(\theta)}\frac{\mathcal{N}(\theta\,|\,0,1)}{\mathcal{N}(\theta\,|\,\mu,\sigma^2)} = \exp(\ell(\theta)+\log\mathcal{N}(\theta\,|\,0,1)-\log\mathcal{N}(\theta\,|\,\mu,\sigma^2))$$

となります. 見やすくするために $y_i = \mu+\sqrt{2}\sigma x_i$ とおけば,

$$\begin{aligned} p(y,d,\boldsymbol{\beta}) &= \int_{-\infty}^{\infty} h(\theta)\mathcal{N}(\theta\,|\,\mu,\sigma^2)d\theta\cdot p(\boldsymbol{\beta}) \\ &\simeq \frac{\sigma}{\sqrt{\pi}}\sum_{i=1}^{H}\exp\left[\log w_i+\ell(y_i)+\frac{1}{2}\left(\frac{1}{\sigma^2}(y_i-\mu)^2-y_i^2\right)\right] \\ &\quad\cdot p(\boldsymbol{\beta}) \end{aligned} \tag{5.124}$$

となり, ℓ_i の中に $\boldsymbol{\beta}$ が含まれるこの式の対数を $\boldsymbol{\beta}$ について偏微分し, L-BFGS 法で最適化することで $\boldsymbol{\beta}$ を計算します. 式(5.124)の対数の偏微分は, 計算すると文書 d の長さを L として,

$$\frac{\partial}{\partial \boldsymbol{\beta}} \log p(y, d, \boldsymbol{\beta}) = L\theta \left[\frac{1}{L} \sum_{v \in d} \vec{v} - \sum_{v=1}^{V} p(v \mid \theta, \boldsymbol{\beta}) \vec{v} \right] \tag{5.125}$$

という直感的にも妥当な結果となります.

PLSS の実験 (続き)　それでは, 実際のテキストを分析してみましょう. ここでは, 国会の議事録を使ってみることにします. 国会会議録検索システム [*66] から, 2023 年度の通常国会での衆議院文部科学委員会のテキストをダウンロードします. サポートサイトにあるスクリプトを用いて次のように実行すると, これは 16 回の開催で, 2,210 個の発言が `livedoor.txt` と同じ「発言者<TAB>発言内容..」の形式で得られます.

```
% diet-split.py *.txt | diet-format.py > all.txt
```

このうち, 次のように実行して大臣および委員長の司会・答弁および 50 文字未満の短い発言を除外すると, 984 個の単語分割された発言とその発言者が得られました. [*67]

```
% diet-sieve.awk all.txt | cut -f 2 | mecab -O wakati \
  -b 65536 > education.txt
% diet-sieve.awk all.txt | cut -f 1 > education.label
```

発言の多い上位順に宮本岳志委員 (共産党), 柚木道義委員 (立憲民主党), 西岡秀子委員 (国民民主党) のうち, 違いの自明でない後者二名の発言から次のようにしてランダムに 20 個の発言を抽出し, これを正例–負例とみて違いの軸を抽出してみることにしましょう. なお, この二者はいずれも野党です.

```
% article-id.py education.label 柚木 20 > yunoki.id
% article-id.py education.label 西岡 20 > nishioka.id
% cat yunoki.id
4,13,149,156,160,162,165,167,234,240,243,250,251,310,311,..
```

この上で, 次のようにして PLSS の半教師あり学習を実行します. 内部で `plss-semi.R` を呼んでいますので, 先にこのスクリプトを単体で実行して, 必要なライブラリをインストールしておいてください.

---

*66　`https://kokkai.ndl.go.jp/`

*67　この場合のように 1 行が長いと MeCab の解析バッファが足りなくなるため, `-b 65536` のようにしてバッファを標準の 4KB から 64KB に増やしています.

```
% plss-semi education.txt yunoki.id nishioka.id jawiki.vec output
preparing word vectors..
total 8031 words selected.
running PLSS-semi..
loading wordvectors from /tmp/wordvec-38450.vec.. done.
preparing data of language ‘‘ja’’.. done.
optimizing beta from examples..
iter    1 value 15.745812
iter    2 value 14.242224
iter    3 value 13.447749
:
iter   29 value 10.210794
final  value 10.210794
converged
number of docs = 984
word dimension = 100
norm of beta   = 2.25
computing theta..
computing 984/984.. done.
theta written to output.theta.
phi   written to output.phi.
```

output.theta および output.phi が推定した各発言の極性 θ と, 計算した $\boldsymbol{\beta}$ に沿った単語の極性 $\phi_v = \boldsymbol{\beta}^T \overrightarrow{v}$ です. 次のように実行すると, θ および発言者, 発言内容を並べて表示し, θ でソートすることができます.

```
% paste output.theta education.label education.txt \
  | sed 's/ //g' | sort -nr
⇒  1.400558 柚木委員 簗副大臣、お考えについては答えてください。L..
   1.369929 柚木委員 そんな、文部科学副大臣として、今おっしゃって..
   1.292844 柚木委員 ということは、永岡文部科学大臣としては、LG..
   :
  -1.292683 西岡委員 大変様々な要件が課されておりまして、また今後..
  -1.305099 西岡委員 これまでも議論になっておりますけれども、やは..
  -1.333402 西岡委員 関連いたしまして、先ほども申し上げましたよう..
```

この全体からランダムに抽出した 20 発言とその極性を図 5.44(a) に示しました. 現場感覚を大事にするアクティブな柚木議員に対し, 文教族の参議院議員を父親に持つ西岡議員は, より制度的なアプローチをとっていることが読み取れます. この違いは, ϕ_v にも見てとることができます. 図 5.44(b) では確かに, 柚

θ	発言者	発言内容
0.9843	柚木委員	まとめて答えていただいたので、最後、ちょっとだけ時間ができたので。..
0.7876	柚木委員	ちょっと警察庁、この後、答弁いただきます。　伊佐副大臣、最後の質問..
0.7387	梅谷委員	そうですね。　じゃ、この組織的あっせん事件の中心的人物が今大学で役..
0.7260	柚木委員	これは驚きの答弁ですよ、大臣。　私、一応、ＬＧＢＴは種の保存に背く..
0.6559	宮本委員	学校給食法第十一条は障害にならない、当たり前です。　しかも、自民党..
0.3464	吉川委員	指針ですけれども、先ほど触れたとおり、強制力を伴わない、望ましいだ..
0.3308	柚木委員	もうこの項目は最後にしたいと思いますが、だとすると、まさに十三ペー..
0.2456	柚木委員	非常に、今後の方向性をある意味中間報告的に今整理して御答弁いただい..
0.1865	梅谷委員	検討中という話ですので、これ以上は申し上げませんが。　ただ、私がも..
−0.0026	菊田委員	大切なことは、やはり、現場の教職員の先生方がすごいプレッシャーとか..
−0.0574	宮本委員	幼児教育無償化したからやれという話じゃなくて、されなくてももちろん..
−0.1607	宮本委員	おっしゃるとおりで、経緯があって、だから多様になっているということ..
−0.2457	吉川委員	これから検討し、またパブコメ等ということですけれども、私は、審議会..
−0.2807	牧委員	確かにおっしゃるように、ガバナンスは大切なんですけれども、逆に、萎..
−0.3307	宮本委員	当然ですね。　さらに、私に対応してくださった金沢大学の職員のお一人
−0.4056	鰐淵委員	ありがとうございます。　今御紹介もありましたけれども、当初は評議員..
−0.4151	早坂委員	先ほど、私も私学出身で、両親が共働きで、どうにか学校を出していただ..
−0.5283	宮本委員	教育機会確保法附則の三には、「この法律の施行後三年以内にこの法律の..
−0.6099	西岡委員	基準、要件については、これから法案が成立した後、審議会で議論をする..
−1.0058	西岡委員	やはり、支出の妥当性、透明性は大変重要だと思いますので、しっかりそ..

(a) 各発言に推定された θ (一部)

勝目	3.0700	簗	2.3292	ワーキング..	−2.6780	学寮	−2.0325
辺野古	3.0331	浮島	2.3156	産学	−2.3894	防災	−2.0261
ワールドカ..	2.6581	恥ずかしい	2.2712	気象	−2.2831	周期	−2.0104
利府	2.6491	飲酒	2.2636	か年	−2.2318	高度	−2.0074
有権者	2.5233	わいせつ	2.2398	凝らし	−2.2294	豊橋技術科学..	−1.9977
中体連	2.4912	英弘	2.2019	耐震	−2.2254	科目	−1.9972
左折	2.4738	下線	2.1374	災	−2.2171	カリキュラム	−1.9865
衆議院	2.4640	起点	2.1102	整い	−2.1866	不断	−1.9793
双葉	2.4262	知事	2.0699	屋根	−2.1721	地震	−1.9738
乗車	2.3836	野球	2.0688	くらし	−2.1421	築い	−1.9514
$\phi_v > 0$ (柚木議員側)				$\phi_v < 0$ (西岡議員側)			

(b) 推定された単語の極性

図 5.44: PLSS の半教師あり学習による極性の推定. 極性辞書を使わなくても, 与えた文書集合のうち, 少数の正例と負例の文書の番号を与えれば, 極性を表す軸 $\boldsymbol{\beta}$ をそこから学習することができます.

木議員 (+ 側) では「左折」「乗車」「野球」といった単語の極性値が高く, 西岡議員 (− 側) では「産学」「耐震」「防災」といった単語の極性値が高くなっており, 同じ野党でありながらも, 興味の違いや政治姿勢を反映していることがわかります. また他の議員の立場も同じ軸の上で解釈することができ, 全体的な構造が一見不明な文部科学委員会での議論に, 一定の計量的な視点を与えることができました.

5 章の演習問題

(1) ナイーブベイズ法において, 式(??)の平滑化パラメータ α_w の値を変えると, 性能にどのような影響があるでしょうか. これを 0 にすると, どんな問題が生じる可能性があるでしょうか.

(2) ナイーブベイズ法で学習したモデルについて, 各単語のラベル事後確率 $p(y|w)$ および対数オッズ比を計算してみましょう. また, 単語の出現確率と対数オッズ比はどのような関係にあるか, プロットしてみましょう.

(3) Livedoor コーパスの各カテゴリでの単語分布 $p(w|y)$ から, NPMI を使って, 各カテゴリの特徴語を計算してみましょう. 上位語, 下位語はどのようになっているでしょうか. また, UM で教師なしで得られたカテゴリの特徴語とは, どのような違いがあるでしょうか.

(4) ナイーブベイズ法で, 単語のカテゴリ所属確率 $p(y|w)$ が特定のカテゴリについて高い語には, どんなものがあるでしょうか. $p(y|w)$ がどのくらい特定の y に集中しているかは, 確率分布のエントロピー (??節) で測ることができます. $p(y|w)$ のエントロピーの大小で単語をソートすると, どんな結果になるでしょうか.

(5) ユニグラム混合モデルについて, 学習したモデルから各単語 w のトピック事後確率 $p(z|w)$ を計算してみましょう. 意外なトピックに事後確率が高い単語はあるでしょうか.

(6) 任意の単語 w_1, w_2, w_3 を 3 つ選んで, 語彙がこれらの単語だけからなるとしたとき, 適当なコーパスの各文書について最尤推定で計算した確率分布 $\mathbf{p}=(p_1, p_2, p_3)$ を, 図 5.14 のように単体上にプロットしてみましょう. 意味的に相関のある単語のペアを選んだとき, プロットにはどんな傾向が現れるでしょうか.

(7) DM で学習されたディリクレ分布のハイパーパラメータ $\boldsymbol{\alpha}_k$ から, 期待値を求めることで, 各トピックがどんな意味を持っているかを調べてみましょう. UM と何か違いはあるでしょうか.

(8) 〜で行ったように, 適当な小説などのテキストに対して, ある単語が出現

してから, 次に出現するまでの時間 (=何単語後か) をプロットしてみましょう. 地震のようなイベントがランダムな時間に起きる様子を記述するポアソン過程の言葉では, これは前方再起時間 (forward recurrence time) とよばれています. 単語によって, どんな特徴があるでしょうか. この分布をモデル化するには, どうするとよいでしょうか.

(9) 文書による単語の長さの分布の違い.

(10) 文書の特徴を文字あるいは文字の種類で考えてみる.

(11) Livedoor コーパスについて, LDA の θ と教師ラベルとの対応. ラベルごとに, どんなトピックが多いのかをプロットしてみましょう. 教師ラベルとしては, どんな話題が足りなかったといえるでしょうか.

(12) ICA の各軸の小さい方を調べてみる. (iPhone¡-¿Android, 国内映画¡-¿海外映画等)

(13) 3 章で前後の共起窓から求めた単語ベクトルについても, 同様に ICA で変換した独立成分軸を求めてみましょう. 本章で文書内全部を共起の対象とした単語ベクトルと, 何か違いがあるでしょうか. また, Word2Vec, GloVe, SVD による単語ベクトルの計算法で何か差がみられるでしょうか.

(14) 同じコーパスでトピックモデルと文書ベクトルを学習したとき, 学習された表 5.9 のようなトピックと, 表 5.13 のような ICA の独立成分軸はどのように異なるでしょうか. どのトピックと軸が似ており, どれが似ていないかを定量化するには, どうすればいいでしょうか?

(15) PLSS で, positive–negative 以外の辞書を使ってテキストを分析してみる.