1　テキストと言語のモデル化

1.1　言語とテキストの特徴

われわれは日々, テキストに囲まれて暮らしています. 図 1.1 のように, 街中にはテキストがあふれていますし, 毎日見る携帯電話の画面から Web ページ, 電子メール, 小説や雑誌に至るまで, われわれがテキストを日々目にしない日はないと言ってよいでしょう.

テキストは言葉を書き起こしたものですが, もちろん, 言葉は書き言葉から生まれたわけではありません. よく知られているように, アイヌ語は文字を持ちませんでしたし, 南米のインカ帝国でも同様でした. しかし, 最初に古代メソポタミアで文字が発明されて言葉が書き表されるようになったことで, 言語は音声によるその場限りのコミュニケーションの媒体から, 空間および時間を超えて伝わる情報を表せるようになり, 抽象化された書き言葉も出現して, 今日みられる高度な文明の礎となりました. 言語にはこのように, テキストに書き表される以前に音声や抑揚, 画像としての文字といった側面もありますが [*1], 本書では記号化されて文字として表されたもの, すなわちテキストを対象とすることにし

図 1.1: われわれは日々, テキストに取り囲まれて暮らしています.
(香港にて, 筆者撮影)

*1　こうしたテキスト以前の音声や文字画像の情報をどうテキストと組み合わせるかは, たいへん興味深い問題です. 実際に深層学習を用いて, フォントから取り出した漢字の部首の画像を文字の情報と組み合わせる研究は現在, さまざまに行われています.

ます.

テキストが, 客観的にみてほかの一般的なデータと異なっている点は何でしょうか. それは,

離散的であること

かつ

1 次元の時系列として表されていること

だと考えられます.

最初に述べたように, テキストはもともと音声を書き起こしたもので, 音声信号は連続的な空気の圧力の違いが時系列で耳に伝わってくるものです. しかし, この連続値の時系列を, われわれは “Good morning” のような離散的なテキストとして認識します. 離散的とは, 情報は文字が単位の場合は “a” … “z”, 単語が単位の場合は “good”, “morning”, “evening”, … のように, 集合のどれかの要素として認識される, ということです. この際, 連続値である音声の周波数や画像の色とは異なり, 綴りが似ているからといって意味が似ているとは限らず, “cat” (猫) と “cut” (切る) はまったく違う言葉になっています. こうした言葉の種類は言語の場合は非常に多く, 文字でも漢字圏なら数千以上, 単語の場合は数万次元を超える超高次元なのが普通です.

また, 画像では色の配置は 2 次元的ですが, テキストはもともと耳で聴く音声を書き起こしたものであるため, 本質的に 1 次元なのが特徴です. よって, ある言葉が次の行にある言葉とたまたま近くにあっても, 両者は原則的に無関係で, これは画像のような空間的なデータとの大きな違いです. 言語は 1 次元ではあるものの, 句構造や埋め込み文, リズムや談話構造といった見えない構造が実は豊富に存在していることも特徴といっていいでしょう.

こうした離散的な時系列であるというテキストの特徴により, 連続値であることを暗に仮定した [*2] 主成分分析, 多変量解析などの古典的な統計学や機械学習の手法は, 本来はそのままでは使うことができません. それではどうすればよいのかについて, 本書で一緒に考えていきましょう.

---

*2 多くの古典的な統計理論は, 観測値が連続値でガウス分布に従うといったことを仮定しています.

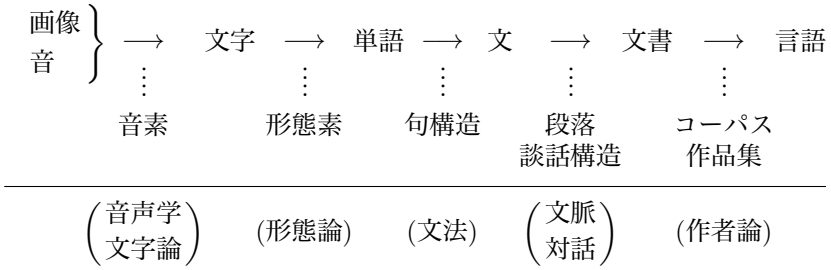

図 1.2: 言語単位の階層構造.

1.2　テキストの階層構造

先ほど, テキストには文字と単語があると書きました. よく考えると, テキストにはもっと多くの構造があることがわかります. この全体像を図 1.2 に示しました.

(1) 第 1 のレベルは, 文字です. これは知覚に際しては, 画素 (ピクセル) の集合とみることができます.

(2) 第 2 のレベルは, 単語です. これはテキストでは文字の集合で, 直接知覚する場合には, 音声から得られる音素の集合です. *3

(3) 第 3 のレベルは, 文です. これは単語の集合で, 単語が連なって一つの文を作ります.

(4) 第 4 のレベルは, 文書です. これは文の集合で, 文が連続して, まとまった意味を持つ一つの文書となります. *4

(5) 第 5 のレベルは, (狭い意味での) 言語です. 多くの文書が日本語や英語など特定の言語で書かれていることを考えると, それらの文書の集合が, 日本語や英語のテキスト全体になります.

もちろん, これらの中間のレベルもあり, たとえば接頭辞や接尾辞, 語幹といった形態素を扱う形態論 (morphology) は文字と単語のレベルの中間にあり, 名詞

*3　フランス人の子供が “beaucoup” のような難しい綴りを後から覚えるように, 文字を介さずに単語を認識することも原理的には可能です.

*4　この「文書」のことをテキストとよぶことも多いのですが, 本書では書かれた言葉全体をテキストと呼ぶことにします.

句や動詞句, 係り受けといった構造は単語と文の中間に, 段落や談話構造は文と文書の中間に存在するでしょう. ただし多くの場合, 単語, 文, 文書の構造はほぼ [*5] 自明に与えられているため, 上記の分類を採用しています.

重要なことは, これらは**階層構造**をなしているということです. すなわち, 文字の集合が単語に, 単語の集合が文に, 文の集合が文書に, 文書の集合が言語になっています [*6]. このうち, どのレベルに注目するかは興味と目的によるでしょう. 形態論に興味のある場合は文字がベースに, 同意や補足といった談話構造に興味がある場合は文がベースになると考えられます. [*7] エンジニアの方が文書を扱う場合も, 一般的には単語または文をベースにして文書を考える必要があるでしょう.

そこで本書では, 文字→単語→文→文書の順にその統計モデルを見ていくことにします. 必要になる統計的な概念についても, 基本的なレベルから少しずつ学んでいくことにしましょう.

1.3 教師あり学習と教師なし学習

(執筆中)

1.4 統計的な方法とアドホックな方法

なおここで, 本書がどうして「統計的」なテキストのモデルを考えるかについてふれておくことにします. たとえば, 本書の 5 章では文書のモデル化を考えますが, ある文書が 20 個の単語からなり, 含まれる単語の頻度がそれぞれ

“チューリップ” が 4 回, “栽培” が 3 回, “の” が 8 回, “こと” が 5 回

---

*5 正確に言えば, 「単語」や「文」とは何かというのは自明ではない問題です. 日本語では空白で区切られた「単語」は存在しませんし, 「。」で文の終わりが示されるものの, 必ず「。」で終わっているとは限らず, 口語の場合はどこまでが文かも曖昧なところがあります. この問題については, 4.1 節を参照してください.

*6 なお言語学では, 文は形態素の連続に, さらに語は音素の連続に分かれるという構造を**二重分節**と呼んでいます[2].

*7 もちろん, 形態論は前後の単語の影響を受けますし, 談話構造は文に含まれる単語を考慮する必要があるでしょう. このように, 隣接するレベルの情報は必要になりますが, 隣接しないレベルは多くの場合必要ないでしょう. すなわち, 形態論には文書の構造は関係せず, 談話構造には文字のレベルはほとんど影響しないと考えられます.

だったとしましょう [*8]. この文書を本書のように確率的なモデルを考えず, 単に頻度を並べたベクトル

$$\mathbf{x} = (0, \cdots, 0, \underset{\text{チューリップ}}{4}, 0, \underset{\text{栽培}}{3}, 0, \cdots, 0, \underset{\text{の}}{8}, 0, \cdots, 0, \underset{\text{こと}}{5}, 0, \cdots, 0)$$

で表せばよい, という人がいるかもしれません. [*9]

この場合, 上は語彙数 (たとえば 10000) の次元をもつ, 超高次元のベクトルで, そのほとんどは 0 です. あれ, これでは文書の長さで表現が変わってしまうので, ベクトルの長さを 1 にする [*10] ために $\sqrt{4^2+3^2+8^2+5^2} = \sqrt{114}$ で割って,

$$\mathbf{x} = \left(0, \cdots, 0, \frac{4}{\sqrt{114}}, 0, \frac{3}{\sqrt{114}}, 0, \cdots, 0, \frac{8}{\sqrt{114}}, 0, \cdots, 0, \frac{5}{\sqrt{114}}, 0, \cdots, 0\right)$$

とした方がいいですね. あるいは, “の” や “こと” のような一般的な単語の重みを下げるために, 各単語の頻度を tf.idf (5 章) で置き換えた方がいいでしょうか. こうしてベクトルが得られたら, あとは K 平均法でクラスタリングしたり, 主成分分析にかけて次元圧縮し, 「主成分」を抽出すれば, 一丁あがりです. …

こうした素朴な方法は, 特に文系や初心者を対象とした「テキストマイニング」の本でよくみられますが, これでは充分な分析とはいえないでしょう.

充分でない理由は色々ありますが, まず第一に, 上のような処理をすることに直感以外の理由がなく, 処理に任意性がありすぎる, ということが挙げられます. 上でベクトルを正規化するのに, 和を 1 にする方法と長さを 1 にする方法がありましたが, どちらを使えばよいのでしょうか. また tf.idf には, tf 部分を対数にするのか (するなら, 対数の底を何にするのか), tf に 1 を足すのか, といった多くのバリエーションがあります. そもそも tf.idf が最適だという保証もありませんし, これらの選択によって, 分析の結果はまったく違ってきてしまいます. 一方, 統計モデルを使えば, **数学的に何を最適化するのか**, という理由が明確に

∗8　これでは日本語になりませんが, 説明のために簡単にしています.

∗9　実際にこれは「ベクトル空間モデル」とよばれ, 初期の自然言語処理や情報検索で使われていました [3].

∗10　頻度の総和で割って正規化することもできますが, そうすると値は, その単語の文書内での確率と同じです. それならば, なぜ最初から確率的に考えないのでしょうか?

表 1.1: 各テキスト A,B,C,D に現れた単語とその頻度の例.

テキスト	A	B	C	D
単語 1	0	6	20	0
単語 2	2	8	30	8
単語 3	0	1	12	1
:	:	:	:	:
単語 N	3	0	5	2
長さ	50	200	1000	500

なります. その際, 本書の 5 章でみるように数学的に自然な形で “の” のような一般的な語の重みを自動的に下げることができ, ベクトルを正規化する必要もありません. 実際に自然言語処理の歴史は, 最初は発見的に導入された手法が, 数学的でより性能の優れた手法に置き換えられることの連続でした.

第二に, よく行われる主成分分析やベクトルのクラスタリングは, 本来連続的なデータを対象にしており, 頻度のような離散的なデータに直接適用してはいけない, という点が挙げられます. 上の文書ベクトルは超高次元 (たとえば 10000 次元) な上, 値は必ず正で, そのほとんどは 0 になっている, という制約があります. 通常の主成分分析は値が負の場合もある連続値で, 誤差が平均 0 のガウス分布に従うことを仮定しており [4], こうした状況には当てはまりません. 単語の頻度も充分に高ければ, 近似的に連続値とみなすことができますが, これは頻度の高い, ごく一部の単語についてしか成り立ちません. 言語では数回といった低頻度でも意味があることが多く, たとえば, ある人物が「開腹」という言葉を数回使っただけで, われわれはこの人は医師あるいは医療系の方だという大きな情報を得ることができます. しかし, こうした低頻度の言葉を, 通常の多変量解析で適切に扱うことはできません.

たとえば, アンケートの自由回答やある小説家の作品群といった複数のテキストを比較するために, 表 1.1 のように単語の頻度を数えたとしましょう. この表は, 単語 1 はテキスト A には 0 回, B には 6 回, C には 20 回, …出現したことを表しています [*11]. このとき, テキストを比較してクラスタ分析を行うため

*11 これは架空例ではなく, 仏教学の分野で石井公成氏が開発した NGSM [5]は, 仏典の漢字 n グラムについてこうした表を作成して分析を行うものです. また, 政治学方法論の分野でテキスト分析のために広く使われている R のパッケージ `quanteda` では, こうした表を簡単に作ることができ, 基本的なデータ構造になっています.

表 1.2: 表 1.1 を, 各テキスト内で単語が出現する**確率**に直したもの.

テキスト	A	B	C	D
単語 1	0	0.03	0.02	0
単語 2	0.04	0.04	0.03	0.016
単語 3	0	0.005	0.012	0.002
:	:	:	:	:
単語 N	0.06	0	0.005	0.002

表 1.3: 確率の負の対数をとって, **情報量**に変換したもの.

テキスト	A	B	C	D
単語 1	∞	3.51	3.91	∞
単語 2	3.22	3.22	3.51	4.14
単語 3	∞	5.30	4.42	6.21
:	:	:	:	:
単語 N	2.81	∞	5.30	6.21

に, テキスト A の特徴ベクトルを, 表 1.1 を縦に読んで $(0, 2, 0, \cdots, 3)$, C の特徴ベクトルを $(20, 30, 12, \cdots, 0)$ などとするのは, 明らかに適切ではありません. なぜなら, そもそも表 1.1 の背後にある各テキストの長さが大きく違っているからです. テキスト C は長いので頻度は自然と大きくなり, このままではテキスト C の影響が過大に見積もられてしまいます [*12]. また, テキスト D は表 1.1 で見えている範囲の出現頻度はテキスト A と似ていますが, もとになるテキストはずっと長いため, 頻度の意味はかなり違っているはずです.

明らかに, この場合は頻度そのものではなく, 各テキストの中でそれぞれの単語が出現する**確率**を考えるべきでしょう. 表 1.1 の頻度を各テキストの長さで割って求めた確率は, 表 1.2 のようになります. これで, テキストの長さにかかわらず, 単語の出やすさを平等に比較することができました.

ただし, これで終わりではありません. 単語 1 がテキスト B と C で出現する確率はそれぞれ 0.03 と 0.02 で, 0.01 の差があります. 1.5 倍の差ですね. 一方で単語 3 が C と D で出現する確率は 0.012 と 0.002 で, これも 0.01 の差ですが, 実は 6 倍の差があるわけです. これを同じ違いとするのは, 不合理ではないでしょうか [*13]. こうした場合は表 1.3 のように, 確率の負の対数をとって本書の 2 章で説明する**情報量**に直せば, 違いを適切に表すことができます. 表 1.3 のように, このとき前者の差は $-\log 0.03 - (-\log 0.02) = 3.51 - 3.91 = -0.41$, 後者の差は $-\log 0.012 - (-\log 0.002) = 4.42 - 6.21 = -1.79$ となり, 後者の方が同じ確率 0.01 の差でも, 大きな意味があることを表現することができました. 実際

*12　一般にアンケートの自由回答などでは, テキストの長さは大きく異なっているのが普通です.

*13　頻度は確率に比例していますので, 頻度を直接使うもとの方法も, 同じ問題を持っていることがわかります.

にはさらに, ∞ を避けたり, 単語 2 のようにすべてのテキストで高確率で出現する単語 (「の」など) の重みを下げるために, 3 章で説明するように確率の平滑化を行ったり, 非負の自己相互情報量 (PMI) を計算して特徴量とするのがよいでしょう. いずれにしても, こうした考察には, 本書で説明するような**確率・統計的な知識**が不可欠になります.

第三に, 最初に示したようなアドホックな方法は理論的な裏付けがないため, 方法を拡張することができないという問題もあります. たとえば, 単語の頻度に外れ値がある場合や, 欠損値がある場合はどうすればいいのでしょうか. 文書をベクトル化したとして, その時間的な変化を見たい場合も, 理論的な裏付けがなければ, 「適当にプロットする」程度のことしかできません. これには無数の任意性がある上に, たとえば観測のなかった時期があると, 一気に使えなくなってしまいます. こうした例は, 統計モデルを導入すれば, すべて自然な形で解くことができます [*14].

1.5 本書の自然言語処理のギャラリー

(執筆中)

1.6 本書の構成と読み方

ここまでに述べたように, 本書では以下, 文字の統計モデル (2 章), 単語の統計モデル (3 章), 文の統計モデル (4 章), 文書の統計モデル (5 章) に分けてテキストの統計モデルについて説明します. 必要になる確率論や情報理論の基礎, EM アルゴリズムや MCMC 法などの推定法についても, 従来の教科書のように無味乾燥な「予備知識」の章を立てるのではなく, 必要に応じて本文の中で, 言語の例を使いながら説明することにしました. 特定の概念 (たとえばエントロピー) が何だったかな, と復習したい方は, 巻末の索引を利用してください.

また, 本書はわれわれの周りに自然にみられるテキストをどうモデル化するかを考えていますので, 文書分類や通常の形態素解析のように, テキストにその所属するクラスや単語境界といった「正解データ」が人手で付与されている場

*14 この場合, 統計モデルを使えば, 5 章で説明するような別の数学的なベクトル化を行った上で, カルマンフィルタやガウス過程[6]で時間発展を追いかけることが可能になります.

合の分類問題, すなわち機械学習の言葉でいう「教師あり学習」は基本的に対象としていません.[*15] これらは多くの場合, SVM や CRF といった汎用の機械学習手法を適用すれば充分だからです. テキストに対する教師あり学習については, 巻末の参考書ガイドを参照してください.

なお, 正解データが付与されている場合であっても, 本書でみるような統計モデル化が可能, あるいは不可欠な場合もあります. 特に, 正解データが一部しか付与されていない場合 (半教師あり学習) や, 正解データを予測することが目的ではなく, それを利用して別の構造を学習したい場合には, 統計モデルはほぼ不可欠といえます. これらをどう行うか, 少しずつ学んでいきましょう.

1.7　本書の例と実装について

本書で無印のタイプライター体の例は, サポートサイトの対応する章の Jupyter Notebook にあるように, Python の Jupyter Notebook に順番に入力して実行できるようになっています. ⇒ は出力です [*16].

```
    print（"これは例です．"）
⇒ これは例です．
```

ただし, 複雑な例ではノートブック上での入力や実行には限界がありますので, %で始まる行はスクリプトやファイルのある各章のフォルダでコマンドラインを実行することを表しています. % から後が, 実際に実行するコマンドです.

```
    % date
⇒ 2020 年 11 月 22 日 日曜日 22 時 35 分 56 秒 JST
```

コマンドラインは, MacOS や Linux では「ターミナル」アプリを起動すればそのまま使うことができます. Windows では, WSL (Windows Subsystem for Linux) を導入するといいでしょう. WSL の導入方法およびコマンドラインの初歩については, 付録 A を参照してください.

∗15　確率の言葉でいえば, 通常の教師あり機械学習はテキスト $\mathbf{x}$ に付与されたラベル $\mathbf{y}$ をいかに予測するか, すなわちラベルの確率 $p(\mathbf{y}|\mathbf{x})$ だけをモデル化しており, 本書のようにテキスト自体の構造やそのラベルとの関係, すなわち $p(\mathbf{x})$ や $p(\mathbf{x}, \mathbf{y})$ はモデル化していない, ということです.

∗16　Jupyter Notebook の `Out[]:` にあたります.

テキストを扱う場合は, Jupyter Notebook や RStudio のような環境の中ではできることが限られるため, プログラム開発の面からも, ぜひスクリプトを書いてコマンドラインで実行できるようになっていただきたいと考えています. スクリプトを書くことで複雑なプログラムとそのデバッグが可能になるほか, 汎用性のあるモジュールを自分で書いて `import` することで, プログラムの再利用が可能になり, より見通しのよい開発を行うことができます. また, テキストの行数を数えるといった簡単な場合も, Python の中では

```
n = 0
with open('brown.txt','r') as f:
    for line in f:
        n += 1
print(n)
```

のようにプログラムを書かなければなりませんが [*17], コマンドを使えば

```
% wc -l brown.txt
⇒ 55039 brown.txt
```

と, 一瞬で答えを得ることができます. 他にもテキスト分析のためには豊富なコマンドがあり, 本書でも様々なスクリプトを書いて使っていますので, ノートブックや統合環境の箱庭の中だけで分析を行うのではなく, ぜひコマンドラインを使いこなせるようになってください.

サポートサイトには本書で用いた Python スクリプトがありますが, これが「正解」ということはなく, 様々な実装がありえます. 理解のために, ぜひ中身を読んで自分で試してみてください. その際, 各章の最後にある「実習」で興味を持ったものを行ってみるとよいでしょう.

このほか, テキストを処理できる主な言語に R や Julia があります. 特に Julia は高速なことで知られており, 本書の後半に現れるような複雑な確率モデルを計算する際には有効です. また, テキスト処理専用の言語として Unix の初期から存在する `awk` や `sed` は, 簡単なテキスト処理なら 1 行で書くこともでき, たいへん便利です. こうしたテキスト処理のための言語とその特徴については, 67 ページのコラムを参照してください.

*17 もっと短く書けるというエキスパートの方や, `system()` を使う方もいるかもしれませんが, そうした方はすでにコマンドラインも知っていることでしょう.