R と日本語テキストマイニング

石田 基広 *

2008 年 9 月 14 日

目次

* ishida-m@ias.tokushima-u.ac.jp

1 MeCab と RMeCab

1.1 本章の目的

本章では日本語形態素解析器 MeCab のインストール方法，さらには R から MeCab を操作するためのパッケージである **RMeCab** のインストール方法と利用方法を説明する．

1.2 形態素解析とは

形態素とは言語学で使われる専門用語で，「意味の最小の単位」と説明される．例えば「本を読んだ」という文は，直感的には「本」，「を」，「読んだ」と分割されそうだが，言語学の立場では「本」，「を」，「読む」，「だ」と分割される．「読んだ」は五段活用の動詞である「読む」と過去を表す助動詞の「だ」で構成されていると分解するのである．文を形態素の単位に分割することを形態素解析という．我々がコンピューターで日本語を入力する際に行っている「かな漢字変換」も形態素解析の一つである．形態素解析では，文を形態素の単位に分割すると同時に，各形態素の品詞を特定することまで行われる．

文を形態素に分割することを手作業で行うと膨大な時間と手間がかかる．さらには分析者の主観に左右される場合も多い．人間の言語を，機械の言語 (例えばプログラミング言語) と区別して自然言語と呼ぶことがある．自然言語の本質は曖昧性にあり，形態素解析においても最適な解を見つけるのが困難な場合がある．例えば「にわはにわ」というような文の一部が与えられた場合，これは「庭は庭」とも「二羽は二羽」とも解釈できる．

このような曖昧性があるため，コンピュータに処理させようが，人手に任せようが，いずれにせよ日本語形態素解析の結果は完全なものとみなすことはできない．すると，長文のテキストを対象に形態素解析を行う場合の現実的な対応策として，始めにコンピュータに処理を行わせ，その結果を可能な限り人間が修正することが考えられる．また多くの形態素解析器はユーザーが辞書に項目を追加登録することができるので，辞書機能を充実させることで，後の修正の手間が省けるようにもなる．

本書においては形態素解析器の解析結果をそのまま使ってデータ解析を行っている．独自に辞書を整備することは行っていない．読者は，形態素解析器の結果が決して完全ではないこと念頭におき，自分自身の研究目的で独自にテキスト分析を行う場合には，あらかじめ辞書機能を充実させておくことを検討されたい．

さて，これまでにもたびたび言及してきたが，本書では日本語形態素解析器として工藤拓氏の開発した「和布蕪 (MeCab)」[*1]を利用する．工藤氏は，これまでにも Juman や ChaSen といったフリーの形態素解析器を公開しているが[*2]，総じて MeCab の方が ChaSen より優れた解析結果を出力する．

1.3 形態素解析器 MeCab

MeCab は前節で言及したサイトより，読者の利用する OS にあわせたファイルをダウンロードする．ファイルのダウンロードは多少手間がかかる．Windows 版であれば MeCab のサイトの [ダウンロード] をクリックし，さらに [Binary package for MS-Windows] という項目の下にある [ダウンロード] をクリックする．すると sourceforge.net という別のサイトに画面が切り替わる．このページの下の方に [mecab-win32] というリン

[*1] http://mecab.sourceforge.net/

[*2] http://mecab.sourceforge.net/feature.html

クがあるので，これをクリックする．Mac OS X や Linux であれば [mecab] というリンクをクリックする．ただし Mac OS X や Linux では別に辞書をインストールする必要があるので，[mecab-ipadic] というリンク先のファイルもダウンロードしておく．Windows の場合，辞書を別にインストールする必要はない．

Package	Release	Date	Notes / Monitor	Downloads
mecab	0.97	February 3, 2008		Download
mecab-ipadic	2.7.0-20070801	July 31, 2007		Download
mecab-java	0.97	February 3, 2008		Download
mecab-jumandic	5.1-20070304	March 5, 2007		Download
mecab-perl	0.97	February 3, 2008		Download
mecab-python	0.97	February 3, 2008		Download
mecab-ruby	0.97	February 3, 2008		Download
mecab-win32	0.97	February 3, 2008		Download

図 1–1　MeCab ダウンロードサイト

本書の執筆時点での Windows 版 MeCab のバージョンは 0.97 であり，ファイル名は mecab-0.97.exe であった．適当なフォルダにダウンロードした後ダブルクリックすればインストールが始まる．後はインストール中のメッセージに従って [OK] を押していけばよい．文字コードの設定はデフォルトのまま Shift JIS にしておく．MeCab は標準では C ドライブの Program Files フォルダにインストールされる．

Mac OS X を含めた Unix 系ユーザーであれば，本体のソースファイルと辞書を別々にダウンロードし，それぞれのファイルを解凍後にインストール作業を行う必要がある．以下，簡単に手順をまとめておく．ここではダウンロードしたソースファイルがユーザーの Downloads フォルダに保存されているものとする．また作業はすべて Terminal にコマンドを入力して行う．Mac OS X の場合，あらかじめ付属 DVD から開発環境をインストールしておく必要がある．

```
# 本体のインストール
# ** はファイルのヴァージョン番号に変えること
$ cd Downloads
$ tar zxvf mecab-0.**.tar.gz
$ cd mecab-0.**
$ ./configure --with-charset=utf-8
$ make
$ sudo make install
# 辞書のインストール
```

```
$ tar zxf mecab-ipadic-2.7.0-20070****.tar.gz
$ cd mecab-ipadic-2.7.0-20070****
$ ./configure --with-charset=utf-8
$ make
$ sudo make install
```

Windows 版ではインストール後，デスクトップに白色のアイコンが表示される．これをダブルクリックすると MeCab が起動する．簡単な文章であれば，このウィンドウ内に直接文字入力して [Enter] キーを押せば，形態素解析の結果を返してくれる．ファイルを指定して解析を実行し，その結果を別ファイルに保存したい場合は，コマンドプロンプトを起動し，この中で操作することになる．例えば以下では，C ドライブの work フォルダ内にある test.txt というファイルを解析し，その結果を res.txt という名前のファイルに保存している．

C:¥Program Files¥MeCab¥bin > mecab c:¥work¥test.txt > c:¥work¥res.txt

test.txt の中身が「この近くに郵便局ありますか」であれば，その解析結果が res.txt ファイルに表 1–1 のように記録されている．

この	連体詞,*,*,*,*,*, この, コノ, コノ
近く	名詞, 副詞可能,*,*,*,*, 近く, チカク, チカク
に	助詞, 格助詞, 一般,*,*,*, に, ニ, ニ
郵便	名詞, 一般,*,*,*,*, 郵便, ユウビン, ユービン
局	名詞, 接尾, 一般,*,*,*, 局, キョク, キョク
あり	動詞, 自立,*,*, 五段・ラ行, 連用形, ある, アリ, アリ
ます	助動詞,*,*,*, 特殊・マス, 基本形, ます, マス, マス
か	助詞, 副助詞／並立助詞／終助詞,*,*,*,*, か, カ, カ
EOS	

表 1–1　「この近くに郵便局ありますか」の出力

MeCab は入力テキストを分かち書き (形態素に分割) し，その結果を出力する．各行は左から，表層形と品詞, 品詞細分類 1, 品詞細分類 2, 品詞細分類 3, 活用形, 活用型, 原形, 読み, 発音となっている．最後の行の EOS は文の終端 (end of sentence) を意味する．表層形のことをトークン (token) ，あるいは延べ語数ともいう．一方，各行の右から三つ目に形態素原型が表示されている．これをタイプ (type) ，あるいは異なり語数ともいう．テキストのトークン数とタイプ数は一致しないのが普通である．例えば以下の文章を MeCab で分析してみると表 1–2 の結果がえられる．トークンとタイプを特に区別しない場合はタームといういい方も使われる．

我輩は猫である。名前はまだ無い。

句点を除くと 9 個のトークンに分割されるが，このうち助詞の「は」は二回使われているので，タイプ数は一つ少ない 8 個である．あるいは形態素の種類は八つだといってもよい．

MeCab はトークンごとに行を取って，コンマ区切りで出力してくれるので，CSV 形式のファイルとして表計算ソフトで開くこともできる (**??**ページのファイルへの入出力を参照のこと)．ただし各行の先頭要素は，

我輩	名詞, 一般,*,*,*,*, 我輩, ワガハイ, ワガハイ
は	助詞, 係助詞,*,*,*,*, は, ハ, ワ
猫	名詞, 一般,*,*,*,*, 猫, ネコ, ネコ
で	助動詞,*,*,*, 特殊・ダ, 連用形, だ, デ, デ
ある	助動詞,*,*,*, 五段・ラ行アル, 基本形, ある, アル, アル
。	記号, 句点,*,*,*,*,。,。,。
名前	名詞, 一般,*,*,*,*, 名前, ナマエ, ナマエ
は	助詞, 係助詞,*,*,*,*, は, ハ, ワ
まだ	副詞, 助詞類接続,*,*,*,*, まだ, マダ, マダ
無い	形容詞, 自立,*,*, 形容詞・アウオ段, 基本形, 無い, ナイ, ナイ
。	記号, 句点,*,*,*,*,。,。,。
EOS	

表 1–2 「我輩は猫である。名前はまだ無い」の出力

トークン (表層形) とその品詞情報をコンマではなくスペースで区切っているので注意を要する．

本書では日本語テキストは MeCab に形態素解析をゆだね，その結果を R に取り込んで分析を行う．しかし MeCab と R をそれぞれ別々に操作していたのでは解析効率が悪い．そこで本書では R から MeCab を操作するための独自のパッケージ **RMeCab** を用意した．

1.4 RMeCab について

RMeCab は筆者が独自に開発したパッケージであり，R から日本語の文章やファイルを指定して MeCab に解析させ，その結果を R で標準的なデータ形式に変換して出力させるプログラムである．以下 **RMeCab** の導入方法と基本的な使い方を解説する．

1.4.1 インストール

RMeCab は R から MeCab を操作するためのパッケージである．従って R と MeCab をあらかじめインストールしておく必要がある．それぞれのインストール方法は**??**ページとこの章の前半部分を参照されたい．まず **RMeCab** を筆者の用意したページ[*1]からダウンロードする．ここには各種 OS 用の **RMeCab** パッケージ本体に加えて，本書に掲載したコードとデータをまとめて圧縮したファイルがある．例えば本書の執筆時点での **RMeCab** パッケージの開発バージョン番号は 0.50 であり，この番号を含んだ RMeCab_0.59.zip, RMeCab_0.59.tgz, RMeCab_0.59.tar.gz の三種類のファイルが公開されている．このうち末尾が .zip で終わるファイルが Windows 用のファイルである．Mac OS X 版であればファイル名の最後は .tgz であり，Unix 系ならば .tar.gz で終わる．なお Windows 版の場合 RMeCabInstall.txt というファイルもあわせてダウンロードしておく．これは Windows 版 R で **RMeCab** を利用する環境を整えるバッチファイルである[*2]．なお，ダウンロードの際，ブラウザによってはファイル名に [1] などの記号を加えることがある．この結果，パッケージをインストールする際にエラーが生じることがある．ファイル名が変更された場合には，インストール作業を実

[*1] http://groups.google.co.jp/group/rmecab/

[*2] このバッチファイルはレジストリやシステム設定を変更するわけではなく，既存のフォルダ間でファイルをコピーするだけである．

行する前に手作業でファイル名をもとに戻しておく．

はじめに Windows 版 R でのインストール方法を説明する[*1]．

- R を起動
- R のコンソール画面で `getwd()` を実行し，出力の一行目が以下であることを確認する．
  ”C:/PROGRA 1/R/R-2* *.*/library” *の部分はバージョン番号に読み替える
- R のメニューから [パッケージ] - [ローカルにある zip ファイルからのパッケージのインストール] を選び，ダウンロードした RMeCab_***.zip ファイルを指定してインストールする (図 1–2)．*** の部分はヴァージョン番号を表す数字である
- RMeCabInstall.txt のファイル名を RMeCabInstall.bat に変更する
- RMeCabInstall.bat をダブルクリックする (図 1–3)．背景色が黒のウィンドウが新たに現れるが，しばらく待つとそこに「1 個のファイルをコピーしました」というメッセージが表示される．適当にキーを叩くとこのウィンドウは閉じる．これでインストールは完了である．

Windows 版 R で `getwd()` の出力の最初の行が ”C:/PROGRA 1/R/R-2* *.*/library” ではない場合，ユーザーがインストールするパッケージは，ユーザー個人のフォルダにインストールされる．その場合は，Windows XP のユーザーであれば，MeCabInstall.bat の代わりに RMeCabInstallXP.bat を，また Vista のユーザーは RMeCabInstallVista.bat を実行して，環境設定を行っていただきたい．

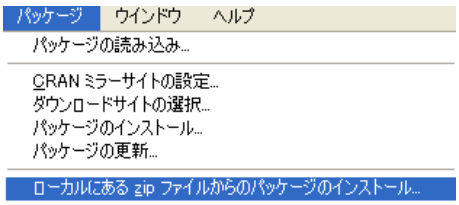

図 1–2　RMeCab のインストール

Mac OS X でのインストール方法を説明する．

- R を起動
- メニューから [パッケージとデータ] - [パッケージインストーラ] を選ぶ．一番上の [CRAN] と表示されているメニューを [このコンピューター上のバイナリパッケージ] に変更する
- 右下の [install] ボタンを押して，ダウンロードした RMeCab_***.tgz を選択する．*** の部分はバージョン番号である．これでインストールは完了である

Linux 版であれば R を起動して，次のコマンドを実行する．この際，ダウンロードしたファイルは R の作業フォルダに保存しておく．作業フォルダは R のコンソール画面で `getwd()` を実行すれば確認できる．以下の ** の部分はバージョン番号である．

[*1] 手順の最後で，サイトからダウロードした RMeCabInstall.txt を実行してインストールを完了させるが，これは R と MeCab を本書の手順に従って，デフォルトのフォルダにインストールした場合にのみ有効である．読者がインストール先を独自に変更した場合，RMeCabInstall.bat を実行する代わりに，次のように処理されたい．まず MeCab のインストール先に bin フォルダがあり，そこに libmecab.dll というファイルがあるので，これをコピーしておく．次に R のインストール先のフォルダを開き，中にある library というフォルダに RMeCab フォルダが作成されているはずなので，これを開く．内部に libs フォルダがあるので，この中に，先の libmecab.dll をコピーする．

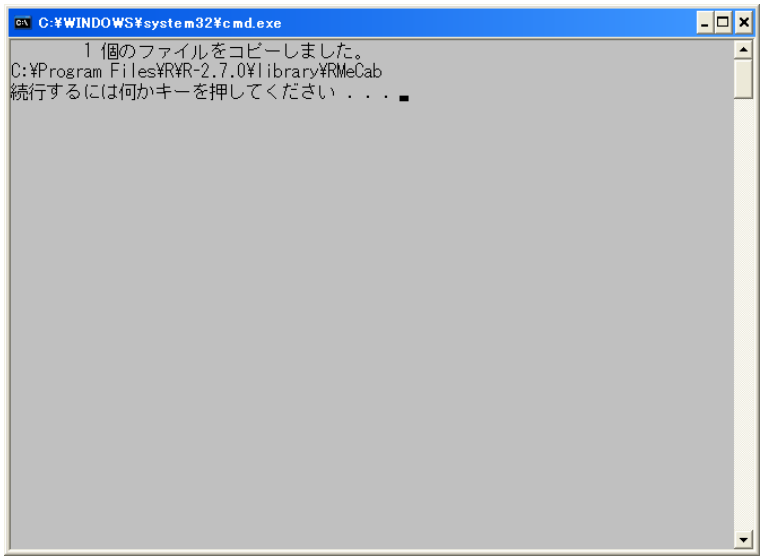

図 1–3　バッチファイル実行後のメッセージ

```
> install.packages("RMeCab_0.**.tar.gz", destdir=".", repos = NULL)
```

2　RMeCab による文字列とテキストの加工

本章では **RMeCab** パッケージを利用して，実際に日本語文字列やテキストの分析を行ってみる．また，テキストマイニングを実行する上で必要となる言語学や自然言語処理について学ぶ．

R に追加したパッケージを利用するには，起動後にパッケージ名を指定して読み込む必要がある．**RMeCab** パッケージを読み込むには，例えば Windows 版であれば R を起動後，メニューの [パッケージ] - [パッケージの読み込み] から **RMeCab** を選択すればよい (図 2–1 を参照)．Mac OS X であれば，メニューの [パッケージ管理] を選び，ダイアログの中の RMeCab にチェックを入れればよい．あるいは R のコンソール画面で `library(RMeCab)` を入力して [Enter] キーを押す．この操作は R を起動するたびに行う．

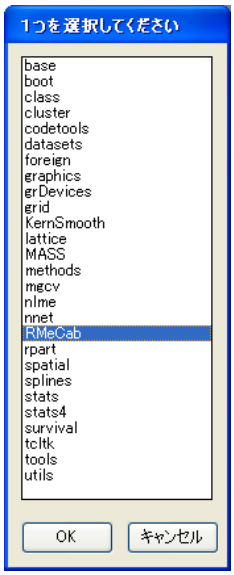

図 2–1　RMeCab の読み込み

関数名	引数	機能
RMeCabC	日本語文字列，タイプ	文字列を形態素解析する
RMeCabText	ファイル	ファイルを形態素解析する
RMeCabDF	データフレーム，列番号，タイプ	データフレームの列を形態素解析
RMeCabFreq	ファイル名	頻度表を作成する
docMatrix	フォルダ，品詞，文書ごとの最小頻度，重み, 記号を含むか	ターム・文章行列を作成
docMatrix2	ファイルあるいはフォルダ名，品詞，文書全体を通じての最小頻度，重み, 記号を含むか	ターム・文章行列を作成
docMatrixDF	データフレームの列，品詞，最小頻度，重み	ターム・文章行列を作成
collocate	ファイル名，キーワード，スパン	共起の頻度表を作成
collScores	共起頻度のオブジェクト，キーワード	共起スコアを計算
Ngram	ファイル名，タイプ，N, 品詞	N-gram を返す
NgramDF	ファイル名，タイプ，N, 品詞	N-gram を列に分解してデータフレームとして返す
NgramDF2	ファイルあるいはフォルダ名，タイプ，N, 品詞, 文書全体を通じての最小頻度，記号を含むか	N-gram を列に分解してデータフレームとして返す
docNgram	フォルダ，タイプ，N	N-gram を文書・ターム行列として返す
docNgram2	ファイルあるいはフォルダ名，タイプ，N, 品詞, 文書全体を通じての最小頻度，記号を含むか	N-gram を文書・ターム行列として返す

表 2–1　**RMeCab** パッケージの主な関数

RMeCab パッケージに実装されている主要な関数を表 2–1 に掲載した[*1]．これ以外にも重み付け関数などが実装されているが，これらは表 2–1 に記載した関数内部から呼び出されることを想定している関数であり，ユーザーが直接操作するものではないので表には含めなかった．

以下，表 2–1 の関数について，その目的と利用方法を解説する．なお本章で利用するファイル，また実行するためのコマンドスクリプトは，筆者が用意した **RMeCab** ダウンロード用のページ[*1] に公開している．Windows 版のユーザーは data2.zip ファイルを，Mac OS X あるいは Unix 系のユーザーは data2.tar.gz をダウンロードして利用してほしい．Windows であればファイルをダブルクリックすれば，新しいウィンドウが出現して，その中に data2 フォルダが確認できるはずである．あるいは zip ファイルを右クリックし，[すべて展開] を選ぶと，[圧縮フォルダの展開ウィザード] ダイアログが表示されるので，[次へ] を選んでいけば，zip ファイルが解凍され，中に data2 というフォルダが含まれたウィンドウが表示される．この data2 フォルダをまるごとハードディスクの適当な場所，例えば C ドライブ直下 (C:) や R の作業フォルダにコピーする．作業フォルダは R のコンソール画面で `getwd()` を実行すれば確認できる．

2.1 RMeCab による形態素解析

始めに，日本語の文章を引数に形態素解析を行ってみる．**RMeCab** パッケージでは，日本語文章を形態素に分解する際，動詞などの活用形については，文に出現した語形，すなわち表層形として出力するか，あるいは原形に変換するかをオプション指定することができる．

また解析対象となる日本語文章がテキストに収録されている場合は `RMeCabText()` 関数を利用する．この場合も活用形については表層形と原形のいずれを出力させるか選ぶことができる．

日本語テキストを形態素解析した場合，各形態素の頻度を表にまとめることがある．これを頻度表と呼ぶ．頻度表からは，そのテキストに特に多く出現する，いい方を変えると，テキストに特徴的なタームを調べることができる．頻度表の作成はテキスト分析におけるもっとも基本的な作業である．そこで `RMeCabFreq()` 関数を利用した頻度表の作成について述べる．

さらに，この節では形態素解析の精度をあげるために欠かせない MeCab 辞書の整備方法について簡単に説明する．

2.1.1 RMeCabC() 関数

`RMeCabC()` 関数では，引数として日本語文字列を与えると，これを MeCab で解析した結果を返す．日本語文字列は引用符で囲むことを忘れないようにする．なお，スクリプトファイルの方に R のコードを記述して実行する場合，Windows であれば，そのコードと同じ行にカーソルを合わせた状態で [Ctrl] キーを押しながら [r] キーを押せば，コードが自動的にコンソール画面にコピーされて実行される．ただし，日本語文字列を含むコードを実行する場合には，コード全体を範囲指定して実行する必要があるので注意されたい．

```
> res <- RMeCabC("すももももももものうち")
> res
[[1]]
    名詞
```

[*1] 読者から要望があれば，関数の追加を検討する．

[*1] http://groups.google.co.jp/group/rmecab

```
"すもも"

[[2]]
助詞
"も"

[[3]]
  名詞
 "もも"

[[4]]
助詞
"も"

# ...  中略
> res[[1]]  # リストの各要素，つまり各形態素にアクセスする
     名詞
"すもも"
> unlist(res)
   名詞       助詞    名詞       助詞   ...
 "すもも"    "も"   "もも"    "も" ...
> x <- "すももももももものうち"  # オブジェクトに代入してもよい
> res <- RMeCabC(x)
> unlist(res)
   名詞       助詞    名詞       助詞   ...
 "すもも"    "も"   "もも"    "も" ...
```

結果は R のリスト形式で返され，表示は数行に渡る．入力文は分かち書きされ，分割された形態素の一つ一つがベクトル形式でまとめられている．リストオブジェクトにアクセスするには `[[]]` と数値を組み合わせて，例えば `res[[1]]` と実行する必要がある．R の関数 `unlist()` を併用すれば，解析結果はリストからベクトルに変換される．このベクトルは分割された形態素が要素であり，要素ごとに品詞情報が名称として登録されている．上記の出力では「すもも」という要素には「名詞」という名前が付いている．「も」という要素には「助詞」という名前が付いている．

なお文字列はあらかじめオブジェクト (上記のコード中では `x`) に代入しておいて，これを `RMeCabC()` 関数の引数として実行しても構わない．

`RMeCabC()` 関数に第 2 引数として数字の 1 を指定すると，入力文字列を分かち書きした上で，動詞や形容詞などの活用形があれば，その原形 (終始形) を返す．第 2 引数はデフォルトでは 0 に設定されている．第 2 引数に 1 が指定されない限り，形態素の表層形，つまり文を分割した結果がそのまま返される．以下のコードの実行結果をそれぞれを比較されたい．

```
> res <- RMeCabC("ご飯を食べた", 1)
> unlist(res)       # 形態素原形を出力
    名詞   助詞       動詞     助動詞
  "ご飯"    "を"   "食べる"       "た"
> res <- RMeCabC("ご飯を食べた", 0)
> unlist(res)       # 形態素表層形を出力
    名詞     助詞      動詞       助動詞
 "ご飯"    "を"    "食べ"         "た"
```

第 2 引数として 1 を加えた場合,「食べた」は動詞原形の「食べる」と助動詞「た」に分割されているが，第 2 引数に 0 を指定した場合 (あるいは第 2 引数を指定しなかった場合) は動詞活用形 (表層形) 「食べ」と助動詞「た」と切り分けた結果がそのまま返されている.

形態素解析の結果には品詞情報が名称として加えてあるので，特定の品詞だけを取り出すことも可能である.

```
> res <- RMeCabC("すももももももものうち")
> res2 <- unlist(res)
> res2
    名詞      助詞      名詞      助詞      名詞      助詞      名詞
"すもも"      "も"    "もも"      "も"    "もも"      "の"    "うち"
> res2[names(res2) == "名詞"]
    名詞      名詞      名詞      名詞
"すもも"    "もも"    "もも"    "うち"
> names(res2) == "名詞"        # 比較演算子の働きを確認
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
```

このコードを解説すると，最初に RMeCabC() 関数の出力をリストからベクトルに変換して，新たに res2 オブジェクトに代入している． res2 はベクトルであるから，各要素には添字を使ってアクセスすることができる．ここでは添字として要素番号ではなく，条件を指定している．また条件は，要素の名称が「名詞」であることとしたい．そこで，まず各要素の名称をチェックする．そのために names() 関数を利用する．この関数を使うと，引数で指定されたオブジェクトの要素の名称をすべて抽出することができる．この中から「名詞」と一致するものを探し出すわけだが，こうした目的のため，プログラミング言語には比較演算子が用意されている．ここでは左辺と右辺の比較を行う == 演算子を使っている．比較演算子は左のオブジェクトのすべてについて，その名前が ”名詞” かどうかをチェックし，一致していれば「真 (TRUE)」と判断し，一致しなければ「偽 (FALSE)」と判断する．かぎ括弧 [] による添字指定の括弧内には，要素番号だけでなく，真や偽を入れることもでき,「真」に対応する要素だけを抽出し,「偽」である要素は抽出しないことを意味するようになる．上記では，ベクトルの要素の名称が ”名詞” と一致する場合は TRUE となり，一致しない場合は FALSE となる．この真偽の判定結果全体を res2 オブジェクトの添字とすることで,「名詞」だけを抽出できるのである.

なお比較演算子の結果について，TRUE と判定された要素番号を確認したければ which() 関数を使うことができる．あるいは TRUE と判定された要素があるかどうかは any() 関数で確かめることができる．

```
> res3 <- names(res2) == "名詞"
> res3
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> which(res3)
 [1] 1 3 5 7
> any(res3)
 [1] TRUE
```

which() 関数の結果は TRUE と判定された要素の位置番号を表し，any() 関数の出力 TRUE は，比較演算子の結果に少なくとも一つは TRUE が含まれていることを示している．

2.1.2 RMeCabText() 関数

RMeCabText() 関数は第 1 引数で指定されたファイルを解析し，その結果をリスト形式でそのまま返す関数である．リストの各要素はベクトルであり，それぞれが 10 個の要素からなる．これは MeCab の出力をそのまま要素にしたもので，最初の要素が形態素の表層形であり，続いて品詞情報，カタカナ書きなどが続く．

以下の実行例では作業フォルダが，**RMeCab** のページからダウンロードした data2 ファルダに設定されているものとする．data2 フォルダ内のファイル yukiguni.txt は川端康成『雪国』の冒頭部分である．

```
> res <- RMeCabText("yukiguni.txt")
> res
 [[1]]
  [1] "国境"        "名詞"        "一般"        "*"           "*"
  [6] "*"           "*"           "国境"        "コッキョウ" "コッキョー"
 [[2]]
  [1] "の"      "助詞"    "格助詞" "一般"    "*"       "*"       "*"       "の"
  [9] "ノ"      "ノ"
 [[3]]
  [1] "長い"              "形容詞"              "自立"              "*"
  [5] "*"                 "形容詞・アウオ段" "基本形"            "長い"
 # ... 以下略
```

2.1.3 RMeCabFreq() 関数

RMeCabFreq() 関数は指定されたファイルを形態素解析し，活用形は原形に変換した上で，その頻度を数え，結果をデータフレームとして返す．なお出力には句読点などの記号とその頻度も含まれている．以下に示すのは Windows 上で実行した結果である．Linux や Mac OS X では，文字コードの関係で，出力が Windows とは逆順に表示される．

```
> res <- RMeCabFreq("yukiguni.txt")
 length = 13
> res
      Term  Info1    Info2 Freq
 1       。   記号     句点    3
 2    長い 形容詞     自立    1
 3    白い 形容詞     自立    1
 # ... 以下略
```

res オブジェクトの名前を入力すると，解析結果がすべて表示される．出力の一行目は列名であり，Term が形態素，Info1 が品詞，Info2 は品詞細分類であり，最後の Freq が頻度である．この例では「長い」という形容詞がテキスト中に 1 回だけ出現していることが分かる．また句点は 3 回出現している．

対象となるファイルが長いテキストである場合は，解析して結果をえるまでには時間がかかる．例えば，芥川龍之介の『蜘蛛の糸』を解析したところ，筆者の環境では速くて 1 秒強，R の利用状況によっては数秒を要した．data2 フォルダ内の kumo.txt が『蜘蛛の糸』の全文テキストであるので，以下のように実行されたい．

```
> pt1 <- proc.time()    # 起動後の経過を取得
> res <- RMeCabFreq("kumo.txt")
 length = 447
> pt2 <- proc.time()
> #  実行時間を見る
> pt2 - pt1             # 単純な引き算
   ユーザ    システム        経過
 0.008       0.008       1.703
```

なお MeCab 本体に搭載されている辞書のバージョンが異なるため，Windows と Mac OS X や Linux では異なった結果がえられることがある．例えば，上記の実行例では，RMeCabFreq() 関数の実行直後に length = 447 と表示されているが，これはテキスト内の形態素のタイプ数 (異なり語数) が (句読点などの記号を含め) 447 個であったことを意味する．ところが Linux 及び Mac OS X で実行すると，形態素のタイプ数は 446 個となる．Linux や Mac OS X で MeCab を導入する場合は，本体とは別に辞書をそれぞれ最新のソースからインストールする．これらの OS の MeCab では「何とも云えない」という日本語文字列を次のように解析する．

```
何とも 副詞,一般,*,*,*,*,何とも,ナントモ,ナントモ
云え 動詞,自立,*,*,五段・ワ行促音便,命令ｅ,云う,イエ,イエ
ない 形容詞,自立,*,*,形容詞・アウオ段,基本形,ない,ナイ,ナイ
```

ところが辞書ごとインストールする Windows 版では次のように解析している．

```
何とも 副詞,一般,*,*,*,*,何とも,ナントモ,ナントモ
云 名詞,一般,*,*,,*,*,云,ウン,ウン
```

え フィラー,*,*,,*,*,*,エ,エ
ない 形容詞,自立,*,*,形容詞・アウオ段,基本形,ない,ナイ,ナイ

このように形態素解析の結果が Windows と Mac OS X や Linux では異なることがある．これは OS の違いではなく，それぞれの OS 用の MeCab に備わっている辞書の整備状況が異なっていることが原因である．大量のデータ解析においては，この違いは小さな誤差で済まされる場合もある．しかし，例えば特定の単語の分布に注目しているのであれば，出力結果を精査し，関心のある単語の解析が正しく行われているかを必ず確認すべきである．MeCab の解析結果が分析者の研究意図にそぐわない場合は，独自の辞書定義を与えればよい．MeCab の辞書はユーザー側で更新することが可能なのである．詳細は開発者の工藤氏のサイト[*1]で確認することができる．ただ，工藤氏のサイトの説明は Unix 系 OS での作業を想定している．そこで，以下に Windows 版 MeCab で辞書を拡張する方法を説明する．

2.2 MeCab の辞書整備

Windows 版 MeCab の辞書にユーザ定義のタームを追加するには以下の手順を行う．Mac OS X や Linux であっても，ほぼ同じ手順でタームの追加を行うことができる．

まず始めに辞書定義ファイルを用意する．例えば「基広」という個人名を新たに辞書として定義してみる．辞書を整備しない段階で Mecab を実行すると以下のような結果になる．

C:¥Program Files¥MeCab¥bin > mecab

石田基広です
石田　　名詞,固有名詞,人名,姓,*,*,石田,イシダ,イシダ
基　　名詞,固有名詞,人名,名,*,*,基,ハジメ,ハジメ
広　　形容詞,自立,*,*,形容詞・アウオ段,ガル接続,広い,ヒロ,ヒロ
です　　助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
EOS

そこで次のような定義ファイルを作成し，これを CSV 形式のファイルとして保存する．

基広,-1,-1,1000, 名詞, 固有名詞, 人名, 名,*,*, 基広, モトヒロ, モトヒロ

これは MeCab の標準的な出力とほぼ一致する辞書定義となっている．左から表層形，左文脈 ID，右文脈 ID，コスト，品詞，品詞細分類 1，品詞細分類 2，品詞細分類 3，活用形，活用型，原形，読み，発音となっている．こうした記述内容を，例えば motohiro.csv として C ドライブの data フォルダ内 (以下 "C:\data" と表記する) に保存したとする．左文脈 ID と右文脈 ID は -1 を指定しておけば，MeCab が自動的に適当な数値に変換してくれる．コストの方は，よく使うタームであれば比較的小さな数値にしておく．実行してみて解析がうまくいかない場合は，この値を小さくしていけばよい．

次に Windows 付属のコマンドプロンプトから辞書の追加作業を行う．まず [スタート]-[プログラム]-[ア

[*1] http://mecab.sourceforge.net/dic.html

クセサリ]-[コマンドプロンプト] としてコマンドプロンプトを起動する．フォルダを移動するコマンド cd を使って，MeCab のインストール先の bin フォルダに移動する．MeCab のデフォルトのインストール先は C:¥Program Files¥MeCab である．移動したら mecab-dict-index.exe を実行するが，この際，MeCab 辞書の位置するフォルダ名，生成したい独自辞書の名前，入出力ファイルの文字コード，そして先に作成した motohiro.csv ファイルを指定する．以下の実行例では逆スラッシュ記号 (\) を使って途中改行しているが，実際には一行で入力して構わない (その場合は逆スラッシュ記号は不要である).

```
C:¥data > cd “C:¥Program Files¥MeCab¥bin”
C:¥Program Files¥MeCab¥bin¥> mecab-dict-index.exe \
        -d ”c:¥Program Files¥MeCab¥dic¥ipadic” \
        -u ishida.dic -f shift-jis -t shift-jis \
        c:¥data¥motohiro.csv
        reading c:¥data¥mecabDic.csv ... 1

emitting double-array: 100% |###########################################|

done!
```

done と表示されれば，辞書の生成は成功である．辞書は mecab-dict-index.exe を実行したフォルダに作成される．この場合は ishida.dic というファイルであり，これを C:¥data にコピーしておく．次に作成した辞書の場所を MeCab に指示する．C:¥Program Files¥MeCab¥dict フォルダ内に dicrc というファイルがある．このファイルを Windows 付属のメモ帳 ([スタート]-[プログラム]-[アクセサリ]-[メモ帳]) などで開いて，末尾に次のように書き足す.

```
userdic = C:¥data¥ishida.dic
```

ここまでの処理を終えて，改めて MeCab を実行すると，次の結果がえられる.

```
C:¥Program Files¥MeCab¥bin > mecab
石田基広です
石田    名詞,固有名詞,人名,姓,*,*,石田,イシダ,イシダ
基広    名詞,固有名詞,人名,名,*,*,基広,モトヒロ,モトヒロ
です    助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
EOS
```

追加したいタームが活用するような場合は，その活用形をすべてユーザー定義しなければならない．詳細は工藤氏のサイト[*1]で確認されたい.

[*1] http://mecab.sourceforge.net/dic.html

2.3 データファイルの解析

自由記述形式のアンケートでは，被験者の回答として，性別や年齢などの情報とともに，日本語の自由記述文が記録されている．R ではデータファイルはデータフレームとして読み込まれる．

本節では，データフレームの日本語自由記述文が記録されている列を指定して，その列だけを対象に形態素解析を行う方法を説明する．

2.3.1 RMeCabDF() 関数

RMeCabDF() 関数は，例えば表 2–2 のようなファイルを対象として解析を行うものである．このデータには自由記述形式の設問への回答が含まれている．アンケートの集計結果は，多くの場合，このような形式のファイルにまとめられるであろう．

この例では 1 列目が被験者番号，2 列目が性別，そして 3 列目に日本語自由記述文が記録されている．この 3 列目を形態素解析にかける場合，次のように RMeCabDF() 関数の第 1 引数にオブジェクトを，第 2 引数に列番号，あるいは列名を指定する．なお第 3 引数として 1 を加えれば形態素原形が返される．ここでは data フォルダ内にある photo.csv を実際に解析してみる．

ID,	Sex,	Reply
1,	F,	写真とってくれよ
2,	M,	写真とってください
3,	F,	写真とってね
4,	F,	写真とってください
5,	M,	写真とってっす

表 2–2　アンケートの集計結果を CSV 形式のファイルにした例

```
> # まずファイルからデータを読み込み
> dat <- read.csv("photo.csv")
> res <- RMeCabDF(dat, 3)          # 形態素活用形（表層形）を返す
> res <- RMeCabDF(dat, 3, 1)       # 形態素原形を返す
> res <- RMeCabDF(dat, "Reply",1) # 列名で指定する場合引用符で囲む
```

RMeCabDF() 関数の解析結果を代入したオブジェクト res はリスト形式になっており，この例では五つのベクトルからなる．リストの長さは length(res) を実行すれば確認できる．リストの個々の要素を確認するには，かぎ括弧の [[]] を使った添字指定を行う．例えば，res オブジェクトの最初の要素であれば res[[1]] として表示することができる．res[[1]] は 5 個の形態素からなるベクトルが要素となっている．ベクトルの各要素には品詞情報が名称として登録されている．なお以下の出力例では形態素が原形で返されていることに注意されたい．

```
> res[[1]]
    名詞      動詞    助詞      動詞    助詞
```

"写真"　"とる"　"て" "くれる"　"よ"

2.4 ターム・文書行列

テキストマイニングを実行する場合，対象とするテキストの集合から，ターム・文書行列 (term-document matrix) を作成することがある．ターム・文書行列とは次のような行列である．

Term	*doc*1	*doc*2	*doc*3
は	1	1	1
学生	1	1	0
僕	1	0	0
彼女	0	1	1
で	0	0	1
を	0	0	1
数学	0	0	1

ここで列名の doc1, doc2, doc3 はそれぞれファイル名であり，次の内容のテキストファイルである．

doc1: 僕は学生です。
doc2: 彼女は学生です。
doc3: 彼女は数学を学んでいます。

各ファイルの中身を RMeCabText() 関数で解析すると，doc1 は「僕」,「は」,「学生」,「です」,「。」の五つの形態素に分割でき，doc2 は「彼女」,「は」,「学生」,「です」,「。」，doc3 は「彼女」,「は」,「数学」,「を」,「学ぶ」,「で」,「いる」,「ます」,「。」の九つに分割される．句点を除けば，トークンとしては合計 16 個の形態素があるが，タイプとして数えれば七つあることになる．そこで後者のタイプを行にとり，また列に文書名を並べ，各タイプが各文書で出現した頻度を成分として埋めたものが先のターム・文書行列である．すなわちタームとは形態素のことであり，多くの場合，形態素原形（タイプ，あるいは異なり語）である．

テキストマイニングでは，文書間の内容的な近さを分析しようとすることがしばしばある．その際，一部のテキストの集合では高い頻度で使われているタイプが，別のテキストの集合ではほとんど出現しないことがある．このようなタイプは，テキストをテーマごとに分類する手がかりになることが多い．そこで，あるタイプがある文書で出現している頻度をまとめた表を作成すると便利である．そのような表をターム・文書行列と呼び，テキスト解析の出発点となる．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞"))
 file = doc/doc1.txt
 file = doc/doc2.txt
 file = doc/doc3.txt
 Term Document Matrix includes 2 information rows!
 whose names are [[LESS-THAN-1]] and [[TOTAL-TOKENS]]
 if you remove these rows, run
 result[ row.names(result) !=  "[[LESS-THAN-1]]" , ]
```

```
 result[ row.names(result) !=  "[[TOTAL-TOKENS]]" , ]
> res
                        docs
 terms               doc1.txt doc2.txt doc3.txt
   [[LESS-THAN-1]]          0        0        0
   [[TOTAL-TOKENS]]         4        6        8
   学生                     1        1        0
   私                       1        0        0
   数学                     0        1        1
   彼女                     0        1        1
```

出力から，助詞の「は」,「で」,「を」が消えているが，[[TOTAL-TOKENS]] の数は変わらない．

行列の情報を表す二つの行を削除するには次のようにする．

```
> res <- res[ row.names(res) !=  "[[LESS-THAN-1]]" , ]
> res <- res[ row.names(res) !=  "[[TOTAL-TOKENS]]" , ]
> res
            docs
 terms  doc1.txt doc2.txt doc3.txt
   は          1        1        1
   学生        1        1        0
   私          1        0        0
   の          0        1        0
   数学        0        1        1
   彼女        0        1        1
   で          0        0        1
   を          0        0        1
```

また，docMatrix() 関数の出力から，たとえば，全文書を通じて規定の頻度に達したタームだけからなる行列を作成したい場合は次のようにする．

```
> res <- res[rowSums(res) >= 2,]  # 全文書を通しての総頻度が 2 以上のターム
            docs
 terms  doc1.txt doc2.txt doc3.txt
   は          1        1        1
   学生        1        1        0
   数学        0        1        1
   彼女        0        1        1
```

上のコードでは，全文書を通じて総頻度が 2 以上となるタームだけを抽出している．rowSums() は行列の行ごとに合計を求める関数である．この関数の出力を比較演算子の>=2 を使って判定することで，頻度合計が 2 以上となるタームだけを残すことができる．

この他に，docMatrix() 関数では引数 minFreq に数値を指定することで，各文書から抽出するタームの最低頻度を条件付けることができる．デフォルトではこの引数は 1 に設定されており，出現したすべてのタイプを行列にふくむことを意味する．引数を 2 以上に指定すると，文書中に少なくとも 2 回出現したタームだけが抽出される．ただし，ある文書では 1 回しか出現していないタームでも，別の文書で 2 回以上出現していることがある．このような場合は，前者の文書内での頻度は 0 と調整される．たとえば，minFreq 引数に 3 を指定する．いま，文書 A では 3 回以上出現したタームが，他のすべての文書でそれぞれ 1 回か 2 回しか出現していないとしよう．すると，最終的に作成されるターム・文書行列で，このタームの頻度は文書 A では 3 となるが，他のすべての文書では 0 と設定される．また，出力の文書・ターム行列には，[[LESS-THAN-3]] という行が追加され，そこに各文書ごとに頻度が指定の値（いまの場合は 3）未満であったタームの出現回数を合計したトークン数が記録される．[[TOTAL-TOKENS]] の方は pos 引数で指定しなかった品詞をふくめた全トークン数である．ただしデフォルトでは記号はカウントされない．

規定頻度に達していないタームの頻度を 0 に設定してしまうことは，一部の文書に特徴的な語彙を強調することになる．なお，後述する dcoMatrix2() 関数では，引数 minFreq は文書全体を通じての規定頻度を意味する．すなわち上で rowSums(res) 関数を使って行列を操作した場合と同じ出力がえられる．

minFreq 引数に 2 を指定した結果を以下に示す．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞"), minFreq = 2)
#... 中略
> res
                      docs
 terms              doc1.txt doc2.txt doc3.txt
  [[LESS-THAN-2]]          2        3        2
  [[TOTAL-TOKENS]]         4        6        8
```

このテキスト集合の場合，三つのどの文書にも頻度が 2 以上となるタイプはないので，個別のタームに関する行は出力されず，作成される行列は情報に関わる 2 行だけである．[[LESS-THAN-2]] は，指定された品詞で頻度が 2 未満，つまりは頻度が 1 であったタイプそれぞれの頻度の合計を表している．例えば doc1.txt であれば，この文書には記号を除いて四つのトークン（「僕」，「は」，「学生」，「です」）が使われているが，頻度が 2 未満の名詞ないし形容詞のタイプ（「僕」，「学生」）の総頻度が 2 であることがわかる．

別のテキスト集合を例にとって実行してみよう．以下では morikita ファルダ内の三つのテキストをまとめて解析している．

```
> res <- docMatrix("morikita", pos = c("名詞","形容詞"))
 file = morikita/morikita1.txt
 file = morikita/morikita2.txt
 file = morikita/morikita3.txt
 Term Document Matrix includes 2 information rows!
 whose names are [[LESS-THAN-1]] and [[TOTAL-TOKENS]]
```

```
 if you remove these rows, run
 result[ row.names(result) !=  "[[LESS-THAN-1]]" , ]
 result[ row.names(result) !=  "[[TOTAL-TOKENS]]" , ]
> res
                          docs
 terms              morikita1.txt morikita2.txt morikita3.txt
  [[LESS-THAN-1]]               0             0             0
  [[TOTAL-TOKENS]]             42            61            77
  こと                          1             0             0
  化                            1             0             0
  家                            1             1             0
  学                            1             0             2
  系                            1             0             1
  研究                          1             1             1
  #  ... 以下略
```

ここで行列の情報を含む行を削除して，全文書を通じての合計頻度が 2 以上となるタームのみを抽出してみよう．

```
> res <- res[ row.names(res) !=  "[[LESS-THAN-1]]" , ]
> res <- res[ row.names(res) !=  "[[TOTAL-TOKENS]]" , ]
> res <- res[rowSums(res) >= 2,]  # 全文書を通して 2 回以上出現したターム
> res
                  docs
 terms    morikita1.txt morikita2.txt morikita3.txt
  家                  1             1             0
  学                  1             0             2
  系                  1             0             1
  研究                1             1             1
  者                  1             5             2
  #  ... 以下略
```

これに対して，minFreq 引数に 2 を指定して，各テキストから頻度が 2 以上となるタームのみを抽出すると次の結果がえられる．上の実行結果とは異なることに注意して欲しい．

```
> res <- docMatrix("morikita", pos = c("名詞","形容詞"), minFreq = 2)
 file = morikita/morikita1.txt
 file = morikita/morikita2.txt
 file = morikita/morikita3.txt
 Term Document Matrix includes 2 information rows!
```

```
 whose names are [[LESS-THAN-2]] and [[TOTAL-TOKENS]]
 if you remove these rows, run
 result[ row.names(result) !=  "[[LESS-THAN-2]]" , ]
 result[ row.names(result) !=  "[[TOTAL-TOKENS]]" , ]
> res
                          docs
 terms              morikita1.txt morikita2.txt morikita3.txt
  [[LESS-THAN-2]]             18            19            21
  [[TOTAL-TOKENS]]            42            61            77
  出版                         2             0             0
  専門                         2             0             0
  者                           0             5             2
  著者                         0             2             0
  編集                         0             2             0
  皆さん                       0             0             2
  学                           0             0             2
  書籍                         0             0             2
  理工                         0             0             2
```

このテキスト集合の場合，各テキストの記号以外の総トークン数が，それぞれ 42，61，77 であることがわかる．そして頻度数が 2 未満のトークンが，それぞれ 18，19，21 個あることになる．また,「出版」というタームは morikita1.txt で頻度が 2 となっており，他の二つ文書では 0 と表記されている．実際には「出版」は morikita3.txt にも 1 回だけ出現しているが，指定された頻度の 2 には達していないので，他の頻度 1 のタームと同様に [[LESS-THAN-2]] にふくまれてカウントされている．

なお，デフォルトではテキスト内の句読点などの記号はすべて削除されており，頻度数にもカウントされていない．sym 引数に 1 以上の数値を指定するか，あるいは pos 引数のベクトルに” 記号” を含めると，句読点などの記号をふくめた総語数が計算される．句読点などの記号を総頻度（[[TOTAL-TOKENS]]）にふくめる場合は以下のように実行する．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞"), sym = 1)
# ... 中略
> res
                      docs
 terms              doc1.txt doc2.txt doc3.txt
   [[LESS-THAN-1]]         0        0        0
   [[TOTAL-TOKENS]]        5        7        9
   学生                    1        1        0
   私                      1        0        0
   数学                    0        1        1
   彼女                    0        1        1
```

下の出力と比較し，[[TOTAL-TOKENS]] 行の頻度が異なっていることに注意して欲しい．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞"))
# ... 中略
> res
                      docs
 terms              doc1.txt doc2.txt doc3.txt
   [[LESS-THAN-1]]         0        0        0
   [[TOTAL-TOKENS]]        4        6        8 # この行の頻度数が違う
   学生                    1        1        0
   私                      1        0        0
   数学                    0        1        1
   彼女                    0        1        1
```

なお，pos 引数に記号を含めると，自動的に記号の頻度をふくめた総頻度が出力される．

```
> res <- docMatrix(targetDir, pos = c("名詞","形容詞","記号"))
# ... 中略
> res
                    docs
 terms              doc1.txt doc2.txt doc3.txt
   [[LESS-THAN-1]]         0        0        0
   [[TOTAL-TOKENS]]        5        7        9 # sym=1 を設定した場合に同じ
   .                       1        1        1
   学生                    1        1        0
   私                      1        0        0
   数学                    0        1        1
   彼女                    0        1        1
```

2.5 docMatrix2() 関数

docMatrix2() 関数は第 1 引数で指定されたファイル (フォルダが指定された場合は，その中の全ファイル) を読み込んで，ターム・文書行列を作成する．指定可能な引数は directory，pos，minFreq，sym，weight である．directory 引数はファイルないしフォルダを指定する (どちらが指定されたかは自動的に判定される)．pos 引数は抽出する品詞を指定する．minFreq 引数には頻度の最低値を指定するが，docMatrix() 関数の場合とは異なり，全テキストを通じての総頻度を判定対象とする．たとえば minFreq = 2 と指定した場合，すべての文書を通じての合計頻度が 2 以上のタームが，出力のターム・文書行列にふくまれる．これに対して

docMatrix() 関数では，個別の文書ごとの最低頻度の指定であった．sym 引数は，抽出タームに句読点などの記号を含めるかを指定する．デフォルトでは sym = 0 とセットされており，記号はカウントされない．sym = 1 とすると，記号を含めてカウントした結果が出力される．pos 引数に記号がふくまれる場合は自動的に sym = 1 とセットされる．なお，docMatrix() 関数に含まれていた [[LESS-THAN-1]] と [[TOTAL-TOKENS]] の二つの情報行は，docMatrix2() 関数では出力されない．

```
> res <- docMatrix2("doc")# doc フォルダを指定
 to open doc
 f_count=3
 doc2.txt
 doc3.txt
 doc1.txt
 to close dir
 file_name =  doc/doc2.txt opened
 file_name =  doc/doc3.txt opened
 file_name =  doc/doc1.txt opened
 number of extracted terms = 4
 to make matrix now
> res
     doc1.txt doc2.txt doc3.txt
 学生        1        1        0
 私          1        0        0
 数学        0        1        1
 彼女        0        1        1
> # 記号を含める場合は pos 引数を指定する
> res <- docMatrix2("doc", pos = c("名詞","形容詞","記号") )
# ... 中略 ...
> res
      doc2.txt doc3.txt doc1.txt
 .           1        1        1 ## 記号が含まれた
 学生        1        1        0
 私          1        0        0
 数学        0        1        1
 彼女        0        1        1
```

単独のファイルを指定した場合は RMeCabDF() 関数とほぼ同じ結果がえられる．ただし，品詞情報ではなく，タームの語形でカウントするので，語形が同じだが品詞が異なるようなタームがある場合は，RMeCabDF() 関数の出力とは同じにならない．

```
>  # 単独のファイルで最低頻度を 5 と指定した例
```

```
> res <- docMatrix2("kumo.txt", minFreq = 5)
 file_name =  kumo.txt opened
 number of extracted terms = 21
 to make matrix now
 > res
        texts
ない       12
の         18
よう       13
... 中略 ...
蜘蛛       14
血の池      7
釈迦        7
針          5
陀多       17
```

2.6 docMatrixDF() 関数

docMatriDF() 関数は，テキストファイルではなく，データフレームの指定行からターム・文書行列を作成する．指定可能な引数は docMatrix2() 関数と同じである．たとえば，photo.csv ファイルを読み込み，その Reply 列の日本語テキストを解析するには次のように実行する．

```
> dat <- read.csv("photo.csv", head = T)
> res <- docMatrixDF(dat[,"Reply"])
> res
          OBS.1 OBS.2 OBS.3 OBS.4 OBS.5
くださる      0     1     0     1     0
くれる        1     0     0     0     0
とる          1     1     1     1     1
写真          1     1     1     1     1
```

列名は OBS. と行番号の組み合わせになる．無回答 (NA や空白行) の場合は，すべてのタームについて 0 がセットされる．

2.7 行列の重み付け

前節で説明したターム・文書行列では，その成分はあるタームがあるテキストに現れる頻度がそのまま利用されていた．しかしながら，各テキストの長さ (分量) にはばらつきがあるのが普通である．ところが，テキストが長くなると，単語が繰り返し現れる可能性も大きくなる．しかし，例えば 1 万語からなるテキストに 3 回

だけしか出現しなかったタームが，別の 100 語のみからなるテキストに同じく 3 回現れた場合，それぞれのテキストにおけるそのタームの重要度は異なるだろう．

また，コンピューター関係の文書を分析すると，各文書には「コンピュータ」というタームが頻出する．この場合，「コンピュータ」の頻度を比較することにはあまり意味が無い．ところが，対象とするテキスト集合の一部には「メール」や「ブラウザ」というタームが多く出現しているとする．一方，残りのテキストにはこれらのタームはまれで，逆に「ハードディスク」や「CPU」,「メモリ」といったタームが頻出しているとする．この場合，前者はインターネット関連の文書集合であり，後者はハードウェア関係の文書であると分類することが可能になる．

従って，テキストをタームの頻度をもとに分析する場合，タームの頻度をそのまま利用するのではなく，そのタームがテキスト内に占める重要度を考慮すべきである．これはテキストマイニングにおいては重みとして実現することができる．重みとは，ある文書におけるタームの相対的重要度を示す概念である．

重みには「局所的重み (local weight)」と「大域的重み (global weight)」，そして「正規化 (normalization)」の 3 種がある．局所的重みは索引語頻度 TF (term frequency) などとも呼ばれ，ある文書に多く現れる語ほど大きな重みを与えられる．単純にはその語の出現頻度が使われる．一方，大域的重みは文書集合全体を考慮して各語に重み付けを行うもので，文書頻度逆数 IDF (inverse document frequency) と呼ばれる指標が使われる場合が多い．これは，すべての文書に出現しているタームよりは，一部の文書にのみ出現するタームに大きな重みを与える方法である．最後の正規化は，長い文書ほど含まれる語が多くなるため，重みも大きくなってしまうが，こうした文書の長さによる影響を調整することである．ターム・文書行列の重みは，これら三つの重みを組み合わせて作成される．重みについて詳細は**?**や**?**を参照されたい．

2.7.1 ターム・文章行列への重み付け

`docMatrix()` 関数と `docMatrix2()` 関数，`docMatrixDF()` 関数は，抽出された各形態素の頻度に「重み」を付けることができる．この関数は，局所的重みとして `tf` (索引語頻度), `tf2` (対数化索引語頻度: logarithimic TF)，`tf3` (2 進重み: binary weight) を，また大域的重みとして `idf` (文書頻度の逆数)，`idf2` (大域的 IDF), `idf3` (確率的 IDF)，`idf4` (エントロピー) を，さらに正規化については `norm` (コサイン正規化) を実装している．これらの重み付けを利用する場合は `weight` 引数に，それぞれの重みを `*` でつなげた文字列を指定する．

例えば局所的重みに `tf` を，さらに大域的重みに `idf` を使う場合は次のように実行する．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞","助詞"),
                   weight = "tf*idf")
> res
             docs
 terms  doc1.txt doc2.txt doc3.txt
  は    1.000000 1.000000 1.000000
  学生  1.584963 1.584963 0.000000
  私    2.584963 0.000000 0.000000
  の    0.000000 2.584963 0.000000
  数学  0.000000 1.584963 1.584963
  彼女  0.000000 1.584963 1.584963
  で    0.000000 0.000000 2.584963
```

```
を    0.000000 0.000000 2.584963
```

この場合，重みは次のように計算される．doc1.txt の場合，「は」，「学生」，「私」の頻度がそれぞれ 1 であった．これが局所的頻度 tf となる．

次に idf では少数の文書に現れる単語ほど高い重みが与えられる．実際には次の計算式が用いられる．

$$idf = log\frac{N}{n_i} + 1$$

ここで N は文書の数，n_i はターム w_i を含む文書の数である．なお対数の底は 2 である．「は」の場合，三つのどの文書にも現れるので idf は $log2(3/3) + 1$ となり，結果は 1 である．「学生」の場合は三つの文書のうち二つに現れるので $log2(3/2) + 1)$ で 1.584963 となる．「私」は一つの文書にしか現れないので $log2(3/1) + 1)$ で 2.584963 となり，重みも大きくなる．この数値を，最初の tf に乗じたものが，ターム・文書行列全体としての重みとなる．

さらに正規化を行う場合は weight 引数に *norm を加える．

```
> res <- docMatrix("doc", pos = c("名詞","形容詞","助詞"),
                   weight = "tf*idf*norm")
> res
             docs
 terms    doc1.txt  doc2.txt  doc3.txt
   は    0.3132022 0.2563399 0.2271069
   学生 0.4964137 0.4062891 0.0000000
   私    0.8096159 0.0000000 0.0000000
   の    0.0000000 0.6626290 0.0000000
   数学 0.0000000 0.4062891 0.3599560
   彼女 0.0000000 0.4062891 0.3599560
   で    0.0000000 0.0000000 0.5870629
   を    0.0000000 0.0000000 0.5870629
```

正規化は各文書ベクトルの長さが 1 になるように調整することである．文書ベクトルとは，各タームの出現頻度 (あるいはそれに重みを付けた数値) を要素とするベクトルである．上の例ではターム数が八つあるので，各文書は 8 次元のベクトルということになる．docMatrix() 関数では正規化の方法としてコサイン正規化を用いている．コサイン正規化とは，各文書のベクトルのノルムを計算し，その文書の各タームの頻度をノルムで割ったものである．ノルムとはベクトルの大きさを表す数値で，次の式で計算される．

$$\sqrt{\sum(tf * idf)^2}$$

例えば，doc1.txt であれば，$\sqrt{1^2 + 1.584963^2 + 2.584963^2} = 3.192827$ であり，この値で先ほどの tf*idf を割れば，上の実行結果に示した数値がえられる．各文書の列の数値を自乗して足せば，それぞれ 1 となる．これが正規化の意味である．

2.8 N-gram

N-gram とは文字あるいは形態素，または品詞が N 個つながった組み合わせのことである．例えば「国境の長いトンネル」という文字列を，文字を単位として区切ることにし，さらに N を 2 に取れば，表 2–3 の組み合わせになる．すると，最初に [国 - 境] の二文字のペア，さらに右に移動すると [境 - の] の二文字のペアが認められ，以下同様にして，最後に [ネ - ル] の組み合わがある．なお N が 2 の場合を bi-gram (バイグラム) という．

国　境
境　の
の　長
長　い
い　ト
ト　ン
ン　ネ
ネ　ル

表 2–3　文字によるバイグラム

形態素単位で分割し，N を 2 に取るならば，表 2–4 の組み合わせとなる．

国境　　の
の　　長い
長い　トンネル

表 2–4　形態素によるバイグラム

品詞情報で bi-gram を作れば，「国境」は名詞，「の」は助詞，「長い」は形容詞，そして「トンネル」は名詞と判断されて，bi-gram は表 2–5 の組み合わせとなる．

名詞　　助詞
助詞　　形容詞
形容詞　　名詞

表 2–5　品詞情報によるバイグラム

2.8.1 Ngram() 関数

`Ngram()` 関数は，文字あるいは形態素，または品詞が N 個つながった組み合わせの総数を求める．川端康成の『雪国』を例に実際に bi-gram を作成してみよう．なお N-gram で抽出される組み合わせの数は一般に非常に大きいものとなる．従って `Ngram()` 関数を呼び出す際には，解析結果はいったんオブジェクトに代入

し，表示する前にオブジェクトのサイズをあらかじめチェックすることをお勧めする．さもなければ R のコンソール画面に延々とデータの表示が続くことであろう．

始めに文字を単位とした bi-gram を示す．

```
> res <- Ngram("yukiguni.txt")
 file = yukiguni.txt Ngram = 2
 length = 38
> nrow(res)
 [1] 38
> res    # 結果を確認する
      Ngram Freq
 1  [。-信]    1
 2  [。-夜]    1
 3  [あ-っ]    1
 4  [い-ト]    1
 5  [が-止]    1
 6  [が-白]    1
   # ... 中略
 34 [汽-車]    1
 35 [白-く]    1
 36 [車-が]    1
 37 [長-い]    1
 38 [雪-国]    1
```

次に形態素原型（終止形）を単位とした bi-gram を示す．

```
> res <- Ngram("yukiguni.txt", type = 1, N = 2)
 file = yukiguni.txt Ngram = 2
 length = 25
> nrow(res)
 [1] 25
> res
              Ngram Freq
 1        [。-信号]    1
 2          [。-夜]    1
 3        [ある-た]    1
 4      [が-止まる]    1
 5        [が-白い]    1
  # .. 中略
 20     [抜ける-と]    1
```

```
 21     [止まる-た]    1
 22       [汽車-が]    1
 23     [白い-なる]    1
 24 [長い-トンネル]    1
 25       [雪国-だ]    1
```

続いて品詞情報を単位とした bi-gram と tri-gram を示す．tri-gram とは N を 3 とした 3-gram のことである．

```
> # bi-gram の場合
> res <- Ngram("yukiguni.txt", type = 2, N = 2)
 file = yukiguni.txt Ngram = 2
 length = 13
> nrow(res)
 [1] 13
> res
              Ngram Freq
 1  [助動詞-助動詞]    2
 2    [助動詞-記号]    3
 3      [助詞-動詞]    2
 4      [助詞-名詞]    3
 5    [助詞-形容詞]    2
 6    [動詞-助動詞]    2
 7      [動詞-助詞]    1
 8    [名詞-助動詞]    1
 9      [名詞-助詞]    6
 10     [名詞-名詞]    1
 11   [形容詞-動詞]    1
 12   [形容詞-名詞]    1
 13     [記号-名詞]    2
>
> # tri-bram の場合
> res <- Ngram("yukiguni.txt", type = 2, N = 3)
 file = yukiguni.txt Ngram = 3
 length = 20
> nrow(res)
 [1] 20
> res
                      Ngram Freq
 1  [助動詞-助動詞-助動詞]    1
 2    [助動詞-助動詞-記号]    1
```

```
3       [助動詞-記号-名詞]     2
4       [助詞-動詞-助動詞]     1
5         [助詞-動詞-助詞]     1
 # ... 中略
16        [名詞-名詞-助詞]     1
17   [形容詞-動詞-助動詞]      1
18      [形容詞-名詞-助詞]     1
19        [記号-名詞-助詞]     1
20        [記号-名詞-名詞]     1
```

また Ngram() 関数で type に 1，すなわち形態素を指定した場合は，品詞を指定して N-gram を抽出することができる．

```
> res <- Ngram("yukiguni.txt", type = 1, N = 2, pos = "名詞")
 file = yukiguni.txt Ngram = 2
 length = 7
> res
              Ngram Freq
1 [トンネル-雪国]     1
2       [信号-所]     1
3 [国境-トンネル]     1
4         [夜-底]     1
5       [底-信号]     1
6       [所-汽車]     1
7       [雪国-夜]     1
```

解析対象のテキストは「国境の長いトンネルを抜けると雪国であった。夜の底が白くなった。信号所に汽車が止まった。」である．pos = "名詞" を指定すると，助詞の「の」や形容詞の「長い」などは省いて N-gram を抽出する．すなわち，テキストは「国境 トンネル 雪国 夜 底 信号 所 汽車」とみなされて，N-gram とその頻度が抽出されるのである．

なお，N-gram は選択によっては組み合わせの数が非常に多くなる．Ngram() 関数では，組み合わせ数が 4 万を越えた場合，頻度が 1 のカテゴリについては，すべてを表示せず，その総数だけを表示するようになっている．この制限は後で説明する docNgram2() 関数 には設けられていない．

2.8.2 NgramDF() 関数

NgramDF() 関数は，機能また引数の指定については Ngram() 関数と同じであるが，N-gram を構成する各要素ごとに列を区切ったデータフレームとして出力する．以下の例を参照されたい．

```
> kekkaDF <- NgramDF("yukiguni.txt", type = 1, N = 2,
                     pos = "名詞")
```

```
 file = yukiguni.txt Ngram = 2
> kekkaDF
     Ngram1   Ngram2 Freq
1 トンネル      雪国    1
2      信号        所    1
3      国境  トンネル    1
4        夜        底    1
5        底      信号    1
6        所      汽車    1
7      雪国        夜    1
```

一行目の出力は「トンネル」,「雪国」という bi-gram の頻度 (Freq) が 1 であることを意味する．Ngram() 関数では前節の出力にあるように，[トンネル- 雪国] のように，かぎ括弧とハイフンを使って 1 列に出力される．なお，N-gram は選択によっては組み合わせの数が非常に多くなる．NgramDF() 関数では，組み合わせ数が 4 万を越えた場合，頻度が 1 のカテゴリについては，すべてを表示せず，その総数だけを表示するようになっている．この制限は後で説明する NgramDF2() 関数 には設けられてない．

2.8.3 NgramDF2() 関数

NgramDF2() 関数は，機能については NgramDF() 関数と同じであるが，引数指定が少し異なる．指定可能な引数は directory，type，pos，minFreq，N，sym である．第一引数はファイルないしフォルダであり (どちらが指定されたかは自動判定される)，type は文字を単位とするか (type=0)，形態素を単位とするか (type=1)，あるいは品詞を単位とするか (type=2) を指定する．デフォルトは形態素である．pos 引数は pos = c(''名詞'', ''形容詞'') のように指定する．minFreq 引数には頻度の閾値を指定する．例えば minFreq=2 と指定した場合，全文書を通じて 2 回以上出現したタイプが，出力のターム・文書行列に含まれる．N 引数で求める N-gram を指定する．上限は設けていないが，極端に大きな数値を指定すると，R の処理能力を超えてしまう可能性があるので注意されたい．sym 引数は，抽出タームに句読点なので記号を含めるかを指定する．type 引数に 1 (形態を指定した場合) は，記号を抽出するかどうかを指定できる．デフォルトでは sym = 0 とセットされており，記号はカウントされないが，sym = 1 とすると，記号を含めてカウントした結果が出力される．pos 引数に記号が含まれた場合は自動的に sym = 1 とセットされる．

```
> # 解析対象は個別ファイルでも，フォルダ全体でも良い
> res <- NgramDF2("yukiguni.txt", type = 1, N = 2,  pos = "名詞")
file_name =  yukiguni.txt opened
number of extracted terms = 7
> res
    Ngram1   Ngram2 yukiguni.txt
1 トンネル      雪国            1
2      信号        所            1
3      国境  トンネル            1
4        夜        底            1
```

```
5        底      信号            1
6        所      汽車            1
7     雪国        夜             1
#
# NgramDF2() 関数では記号を含めるかどうかを指定できる
> res <- NgramDF2("yukiguni.txt", type = 1, N = 2,  pos = c("名詞","記号"))
file_name =  yukiguni.txt opened
number of extracted terms = 10
> res
     Ngram1   Ngram2 yukiguni.txt
1         。      信号            1
2         。        夜            1
3  トンネル      雪国            1
4       信号        所            1
5       国境 トンネル            1
6         夜        底            1
7         底        。            1
8         所      汽車            1
9       汽車        。            1
10      雪国        。            1

> targetDir <- "doc"

> res <- NgramDF2(targetDir)# フォルダを指定してもよい
> res  # デフォルトは type = 0, N = 2
#    Ngram1 Ngram2 doc1.txt doc2.txt doc3.txt
# 1       い      ま        0        0        1
# 2       す      .         1        1        1
# 3       で      い        0        0        1
# 4       で      す        1        1        0
# 5       の      学        0        1        0
# ...


> res <- NgramDF2(targetDir, type = 1, pos = c("名詞","形容詞") )
> res
#   Ngram1 Ngram2 doc1.txt doc2.txt doc3.txt
# 1     私    学生         1        0        0
# 2   数学    学生         0        1        0
# 3   彼女    数学         0        1        1
```

```
> res <- NgramDF2(targetDir, type = 1, pos = c("名詞","形容詞","記号") )
> res  # 記号を含める

#   Ngram1 Ngram2 doc1.txt doc2.txt doc3.txt
# 1   学生      .         1        1        0
# 2     私    学生         1        0        0
# 3   数学      .         0        0        1
# 4   数学    学生         0        1        0
# 5   彼女    数学         0        1        1

> res <- NgramDF2(targetDir, type = 2)
> res  # 品詞
#   Ngram1 Ngram2 doc1.txt doc2.txt doc3.txt
# 1   助詞    動詞         0        0        1
# 2   助詞    名詞         1        1        1
# 3 助動詞    記号         1        1        1
# 4   動詞    助詞         0        0        1
# 5   動詞 助動詞         0        0        1
# 6   名詞    助詞         1        1        1
# 7   名詞 助動詞         1        1        0

> res <- NgramDF2(targetDir, type = 2, minFreq = 2)
> res  # 品詞で全文書を通じての頻度が 2 以上

#   Ngram1 Ngram2 doc1.txt doc2.txt doc3.txt
# 1   助詞    動詞         0        0        1
# 2   助詞    名詞         1        1        1
# 3 助動詞    記号         1        1        1
# 4   名詞    助詞         1        1        1
# 5   名詞 助動詞         1        1        0

##
```

2.8.4 docNgram() 関数

docNgram() 関数は，前節の Ngram() 関数を拡張したもので，第 1 引数にファイルではなくフォルダを指定し，そのフォルダ内のすべてのファイルを解析対象としてターム・文書行列を作成する．なお，この関数で

も引数 type と N を Ngram() 関数の場合と同じように指定することができる．以下では data フォルダ内にある doc フォルダに含まれた全ファイルを対象に解析する．なおデフォルトではタイプは形態素，また Ngram は 2 に設定されている．

```
> res <- docNgram("doc")
file = doc/doc1.txt Ngram = 2
length = 1
file = doc/doc2.txt Ngram = 2
length = 2
file = doc/doc3.txt Ngram = 2
length = 1
> res
             Text
Ngram         doc1.txt doc2.txt doc3.txt
  [私-学生]          1        0        0
  [数学-学生]        0        1        0
  [彼女-数学]        0        1        1
```

なお??ページでで取り上げるが，Ngram() 関数や docNgram() 関数の出力から，特定の N-gram を取り出すには %in% 演算子を利用する．詳細はお??ページを参照されたい．なお，N-gram は選択によっては組み合わせの数が非常に多くなる．docNgram() 関数では，組み合わせ数が 4 万を越えた場合，頻度が 1 のカテゴリについては，すべてを表示せず，その総数だけを表示するようになっている．この制限は後で説明する docNgram2() 関数 にはない．

2.8.5 docNgram2() 関数

docNgram2() 関数は，文字あるいは形態素，または品詞が N 個つながった組み合わせの総数を求める．基本的には Ngram() 関数と同じであるが，指定可能な引数は directory，type，pos，minFreq，N，sym である．第一引数はファイルないしフォルダであり (どちらが指定されたかは自動判定される)，type は文字を単位とするか (type=0)，形態素を単位とするか (type=1)，あるいは品詞を単位とするか (type=2) を指定する．デフォルトは形態素である．pos 引数は pos = c(''名詞'', ''形容詞'') のように指定する．minFreq 引数には頻度の閾値を指定する．例えば minFreq=2 と指定した場合，すべての文書を通じての総頻度が 2 以上のタイプが，出力のターム・文書行列に含まれる．N 引数で求める N-gram を指定する．上限は設けていないが，極端に大きな数値を指定すると，R の処理能力を超えてしまう可能性があるので注意されたい．sym 引数は，抽出タームに句読点なので記号を含めるかを指定する．type 引数に 1 (形態を指定した場合) は，記号を抽出するかどうかを指定できる．デフォルトでは sym = 0 とセットされており，記号はカウントされないが，sym = 1 とすると，記号を含めてカウントした結果が出力される．pos 引数に記号が含まれた場合は自動的に sym = 1 とセットされる．

```
> res <- docNgram2(targetDir, pos = c("名詞","形容詞") )
> res # デフォルトは文字の 2-gram
```

```
#         doc1.txt doc2.txt doc3.txt
# [い-ま]        0        0        1
# [す-. ]        1        1        1
# [で-い]        0        0        1
# [で-す]        1        1        0
## ...

> res <- docNgram2(targetDir, type = 1, pos = c("名詞","形容詞") )
> res # 形態素、記号を含まず

#             doc1.txt doc2.txt doc3.txt
# [私-学生]          1        0        0
# [数学-学生]        0        1        0
# [彼女-数学]        0        1        1


> res <- docNgram2(targetDir, type = 1, pos = c("名詞","形容詞","記号") )
> res # 形態素、記号を含む

#             doc1.txt doc2.txt doc3.txt
# [学生-. ]          1        1        0
# [私-学生]          1        0        0
# [数学-. ]          0        0        1
# [数学-学生]        0        1        0


> res <- docNgram2(targetDir, type = 2)
> res

#               doc1.txt doc2.txt doc3.txt
# [助詞-動詞]          0        0        1
# [助詞-名詞]          1        1        1
# [助動詞-記号]        1        1        1
# [動詞-助詞]          0        0        1
# [動詞-助動詞]        0        0        1
# [名詞-助詞]          1        1        1
# [名詞-助動詞]        1        1        0

res <- docNgram2(targetDir, type = 2, N = 5)
```

```
res

#                                doc1.txt doc2.txt doc3.txt
# [助詞-動詞-助詞-動詞-助動詞]        0        0        1
# [助詞-名詞-助詞-動詞-助詞]          0        0        1
# [助詞-名詞-助詞-名詞-助動詞]        0        1        0
# [動詞-助詞-動詞-助動詞-記号]        0        0        1
# [名詞-助詞-動詞-助詞-動詞]          0        0        1
# [名詞-助詞-名詞-助詞-動詞]          0        0        1
# [名詞-助詞-名詞-助詞-名詞]          0        1        0
# [名詞-助詞-名詞-助動詞-記号]        1        1        0

res <- docNgram2(targetDir, type = 2, minFreq =2, N = 5) # 文書全体での総頻度を指定
res
#                               doc1.txt doc2.txt doc3.txt
# [名詞-助詞-名詞-助動詞-記号]        1        1        0
```

2.9 語の共起関係

言語学で共起 (collocation) とは，ある語が別の特定の語と隣接して現れる (共起する) ことをいう．例えば『蜘蛛の糸』から「御釈迦様は極楽の蓮池のふちを」という一節を例に取り，キーワードあるいは中心語 (node) を「極楽」と考えてみよう．この語の前の三語だけを取り出すと「(御) 釈迦」,「様」,「は」であり，一方，中心語の後には「の」,「蓮池」,「の」が続く．これらの語が「極楽」の共起語である．中心語を真ん中に置いて前後の語数を指定した範囲をウィンドウという．

仮にすべてのタームがテキスト中にランダムに散っているとすれば，あるタームがテキストのある箇所に出現する場合，そのタームを中心としたウィンドウ内部に，別の特定のタームが決まって出現する可能性は高くないであろう．そこで中心語のウィンドウ内に出現する，あるいは「共起する」タームが，テキスト全体での出現数と比較して，そのウィンドウ内に「有意に」多く現れているのであれば，二つのタームの間には強い関係があると判断できる．

ただし，実際には単語は国語の文法や書き手の意図などの強い制約に影響されているため，テキスト中にランダムに散らばるとは考えられない．しかし共起頻度は，語彙研究の分野では，単語間の関連性を知る目安として広く使われている．

2.9.1 collocate() 関数

RMeCab パッケージに実装されている collocate() 関数は，第 1 引数にファイル名を，引数 node に中心語 (キーワード)，引数 span に前後の語数を指定する．引数 span はデフォルトでは 3 に設定されている．

```
> res <- collocate("kumo.txt", node = "極楽", span = 3)
```

```
> nrow(res)
 [1] 33
> res[25:33,]
            Term  Span Total
 25          極楽    10    10
 26            様     2     7
 27          蓮池     4     4
 28          蜘蛛     2    14
 29          行く     1     4
 30          釈迦     2     7
 31            間     1     3
 32 [[MORPHEMS]]   31   413
 33   [[TOKENS]]   70  1808
```

出力されるデータフレームにはキーワード自身も含まれているので注意されたい．Span 列は指定されたウィンドウでの出現頻度，Total はテキスト全体での出現頻度である．またデータフレーム末尾の 2 行で [[MORPHEMS]] は形態素の種類の数を，[[TOKENS]] は総語数を意味する．なお，どちらにも中心語自身の頻度が含まれている．この例では Span 内の総語数は 70 語であるが，その内訳は中心語が全体として 10 回出現し，その前後 3 語に出現したトークン数が 60 語である．テキスト全体の総トークン数は 1808 語で，総タイプ数は 413 語である．ただし頻度には句読点などの記号も含まれている．

collocate() 関数によって中心語と共起する語の頻度が算出されれば，次に共起語の頻度が有意に大きいかを調べることができる．ここで有意とは，二つのタームが特定の範囲内に共起した回数が，偶然では考えられないほど大きいことを意味する．有意かどうかを統計的に判定する基準として，コーパス言語学では T や MI といった指標が利用されている．

T は，統計解析では平均値の差の検定を行う場合などに使われる指標である．従って母集団，すなわち単語の分布が正規分布していることを前提とする．先にも記述したが，単語は文法や書き手の意図に制約されているので，正規分布を仮定した指標は，言語データを分析する手段としては適切ではないとする考え方もある．しかし，コーパス言語学ではタームの共起関係の有無を調べる指標として広く使われている．T 値を計算する式は以下である (**?**, p.97).

(実測値 - 期待値) ÷ 実測値の平方根

ここで分母の「実測値の平方根」は，中心語と共起語に関する標準偏差の近似値を表す．詳細は**?**を参照されたい．一方，期待値は次のように計算する．共起語が「蓮池」の場合を例に取ると，「極楽」と共起した回数の 4 回が実測値である．次にテキスト全体のトークン数が 1808 で，「蓮池」の頻度は 4 である (つまり 4 回すべて「極楽」と共起してる)．従って，その出現割合は 4/1808 である．一方，いま中心語の頻度が 10 で，それぞれ前後 3 スパンの単語をチェックしている．つまり $3 * 2 * 10$ が指定したウィンドウ内の総語数である．この語数に先の「蓮池」の出現確率を乗じれば，指定ウィンドウ内での期待値が求まる $(4/1808 * 3 * 2 * 10)$．この期待値を実測値から引き，実測値の平方根で割ったのが T 値である．

T 値の評価であるが，統計学では，T の絶対値が 2 が越えるかどうかを簡易的な目安にすることがあるが，コーパス言語学では 1.65 以上であれば，二つのタームの共起は偶然ではないと考えるようである (**?**).

一方，MI は情報科学の相互情報量にもとづく指標である．相互情報量とは，ある記号が出現することが，別の特定の記号の出現を予測させる度合いを意味する．コーパス言語学では，二つのタームの独立性を図る指標として使われている．この指標が大きい場合，二つのタームは独立ではなく，一方のタームが出現していればその共起語が現れる可能性が高いことになる．MI はコーパス言語学で以下の式で定義されている．なお対数の底は 2 である．

(共起回数 ÷ 共起語の期待値) の対数

共起語の期待値は，その共起語のテキスト全体での頻度をテキストの総トークン数で割り，その値にウィンドウ内の総語数を乗じた値である．「蓮池」の場合であれば，中心語の「極楽」との共起回数が 4 であり，一方，「蓮池」の期待値は，そのテキスト全体での頻度 4 をテキストの総語数 1808 で割り，これにウィンドウ内での総語数 $3 \times 2 \times 10$ を乗じることで求まる．R を使うなら，次の式で計算することができる．

```
> log2( 4 / ((4/1808) * 10 * 3 * 2))
```

MI は低頻度語を強調する傾向があるため，テキスト全体での出現頻度は低くとも，専門語のようにテキストを特徴付ける単語を抽出するのに役立つといわれる．MI の値が 1.58 を越える場合，二つのタームの間に共起関係があると考えられる (**?**)．ただし低頻度語を強調するため，極端に頻度が少ないタームを MI の値で評価するのは好ましくないとされる．一般には T の方がバランスの取れた指標として用いられているようである．

いずれにせよ，T も MI も，正規分布あるいは語のランダム性を仮定している指標であるので，その数値の大小を厳密に比較するのではなく，大まかな目安と考えるべきであろう．

さて **RMeCab** パッケージで T と MI を求めてみよう．そのためには collocate() 関数の出力であるオブジェクト (いまの場合は res) を第 1 引数として，collocate() 関数で指定した中心語を node 引数に，同じく span に前後の語数を指定して collScores() 関数を実行する．

```
> res2 <- collScores(res, node = "極楽", span = 3)
> res2[25:33,]
           Term Span Total         T       MI
25         極楽   10    10        NA       NA
26           様    2     7 1.2499520 3.105933
27         蓮池    4     4 1.9336283 4.913288
28         蜘蛛    2    14 1.0856905 2.105933
29         行く    1     4 0.8672566 2.913288
30         釈迦    2     7 1.2499520 3.105933
31           間    1     3 0.9004425 3.328326
32 [[MORPHEMS]]   31   413        NA       NA
33   [[TOKENS]]   70  1808        NA       NA
```

極楽は中心語であるので各指標は計算されず NA で埋められる．同じく [[MORPHEMS]] と [[TOKENS]] の指標も NA とされている．この場合，T 値は約 1.9 であり，2 に近い．「極楽」と「蓮池」の共起回数は有意に多いと考えてよいであろう．また MI に関しても約 4.9 であり，先ほど記述した基準の 1.58 を大きく越えている．従って，二つのタームの間には強い共起の関係があると考えてよいだろう．

索引