応用プログラミング3

第13回　日本語テキストマイニング（1）

専修大学ネットワーク情報学部
田中健太

1．課題の振り返り

1

2. テキストデータの価値

2

今回と次回、2回にわたってテキストマイニングについて取り上げます。これまで、数値データを中心に分析してきましたが、文章（テキスト）も分析の対象として魅力的です。

2.1　テキストデータとは

- 表構造データ、JSONなどの半構造化データとは異なる、「自然言語で書かれた文章」がテキストデータ
- アンケートの回答やSNSの書き込み、文学作品などさまざまなテキストデータが研究・分析の対象になる

	A	B
1	10.かわさきアプリ（ポータル）をダウンロードしたことがない、または削除（アンインストール）した理由を教えてください。	11.かわさきアプリ（ポータル）について、改善してほしい点や今後、追加されると便利だと思う情報や機能があれば教えてください。
14	川崎市ホームページをみているのと変わらない。また、他のアプリのインストールを勧められる為。アプリを入れた趣味がない。わかりにくい。	ホームページそのままではなく、もっとわかりやすいレイアウト、デザインにして欲しい。情報も少ない。
15	川崎アプリがでてこなかった。	子供連れで出掛けられる場所を教えて欲しい。 雨の日など特に。 子どもの企画、子育ての企画などもっと興味を持てるものが欲しい。 保育園などに関する要望など言える場が欲しい。
16	災害情報をみるのに、災害アプリをダウンロードしないといけないなどさまざまな場面で手間がかかる。	アプリを１個にまとめること
17	最近になりやっとスマホに機種変したから。	
18	今後インストールする予定	
19	今の所必要ないため	図書館、市民館、ミュージアム、アリーナ等の情報も掲載してほしい
20	購入からまだ２カ月しかたっていないので利用していない	なし
21	機能の割に動作が重い、容量が大きい、単なる入口でしかないから。	動作速度 災害アプリなどとの統合
22	基本的に何かを起動するだけで、必要ない。 これに容量を使うのがもったいない。	
23	関連項目全てを別アプリでダウンロードしなくてはならないのが面倒くさい。	川崎アプリだけで、子育てやゴミや避難や病院…全部見れてここのアプリだと思います。個別にダウンロードするのは手間がかかるし、川崎アプリをダウンロードする理由がありません。 アプリ一つだけで全部見られるようにしてほしいです。
	とにかく重いし、有効な情報がない。いまはごみ分別のアプリだけ入れてます。これも起動が重いけど。	動作を軽くして、もっと情報量を増やしてほしい。 緊急時に、GPSを利用して現在地から最も近い避難所や給水所がわかるといい。ただし、これも重いと使い物にならない。 また、普段でもどこにどのような災害時に使用できる施設があるか調べられるとよい。

出典：川崎市　かわさきアプリのアンケート（オープンデータ）
https://www.city.kawasaki.jp/170/page/0000085154.html#opendata_dataset_2

3

テキストデータとは、自然言語で書かれた文章（テキスト）を、ファイルやデータベースに記録したものです。近年ではTwitterの投稿やWebサイトの本文などが最たるものですが、他にもアンケートの自由記述や文学作品、新聞記事や会議の議事録など、身の回りにはテキストが溢れています。これらを、一定のルール（1つの投稿を1行に、など）に従って記録することで、データとして分析できるようになります。

2.2　テキストデータの収集

- Web APIを利用することで取得できるデータもある
- マーケティングリサーチ会社に依頼してアンケートを広く実施することもできる
- 学生・研究者はさまざまなコーパスも利用できる

```
# 国会会議録検索システム　検索用APIの使用例
library(jsonlite)
keyword <- URLencode("チャットGPT")
query <- paste0("https://kokkai.ndl.go.jp/api/speech?any=", keyword,
"&recordPacking=json")
res <- fromJSON(query, simplifyVector = TRUE)
```

国立情報学研究所　情報学研究データリポジトリ https://www.nii.ac.jp/dsc/idr/

4

テキストデータの収集方法は、最もシンプルな、「紙でアンケートを取って手入力する」から、Googleフォームなどを使って電子的にアンケートを取る、Web APIで取得するなど、さまざまです。

研究やビジネスにおいて大規模にテキストを収集したい場合は、マーケティングリサーチ会社に依頼して、登録会員などに対してアンケートを実施することもできます。規模や設問数によりますが、数万円から実施できます。(ただし、「アンケートに答えてポイントが欲しい人」が対象になるので、どこまで客観性のある調査になるかは注意が必要です。筆者の勤務経験から)

また、すでに研究機関などが大規模な調査を行っていて、その結果をコーパス (Corpus) として公開している場合もあります。基本的には、研究者のみに提供されるため、学生は指導教員などを経由して利用申請することになるでしょう。

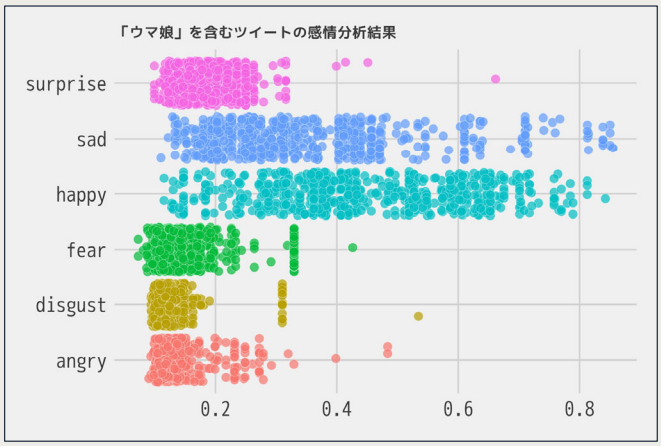
3. テキストマイニングの概要

5

　ここからは、テキストデータを分析すること、テキストマイニングについて紹介します。もちろん、1、2回の授業で説明しつくせるものではないので、あくまで概要を述べるにとどまりますが、全体的な流れを知っておきましょう。

3.1　テキストマイニングとは

- テキストデータを分析し、何らかの知見を得ること
- 数値や選択肢にしにくいヒトの感情などを知る手段として用いられる
- 単純なトレンドの把握や、テキストの感情（ポジネガ）分析などが行われる

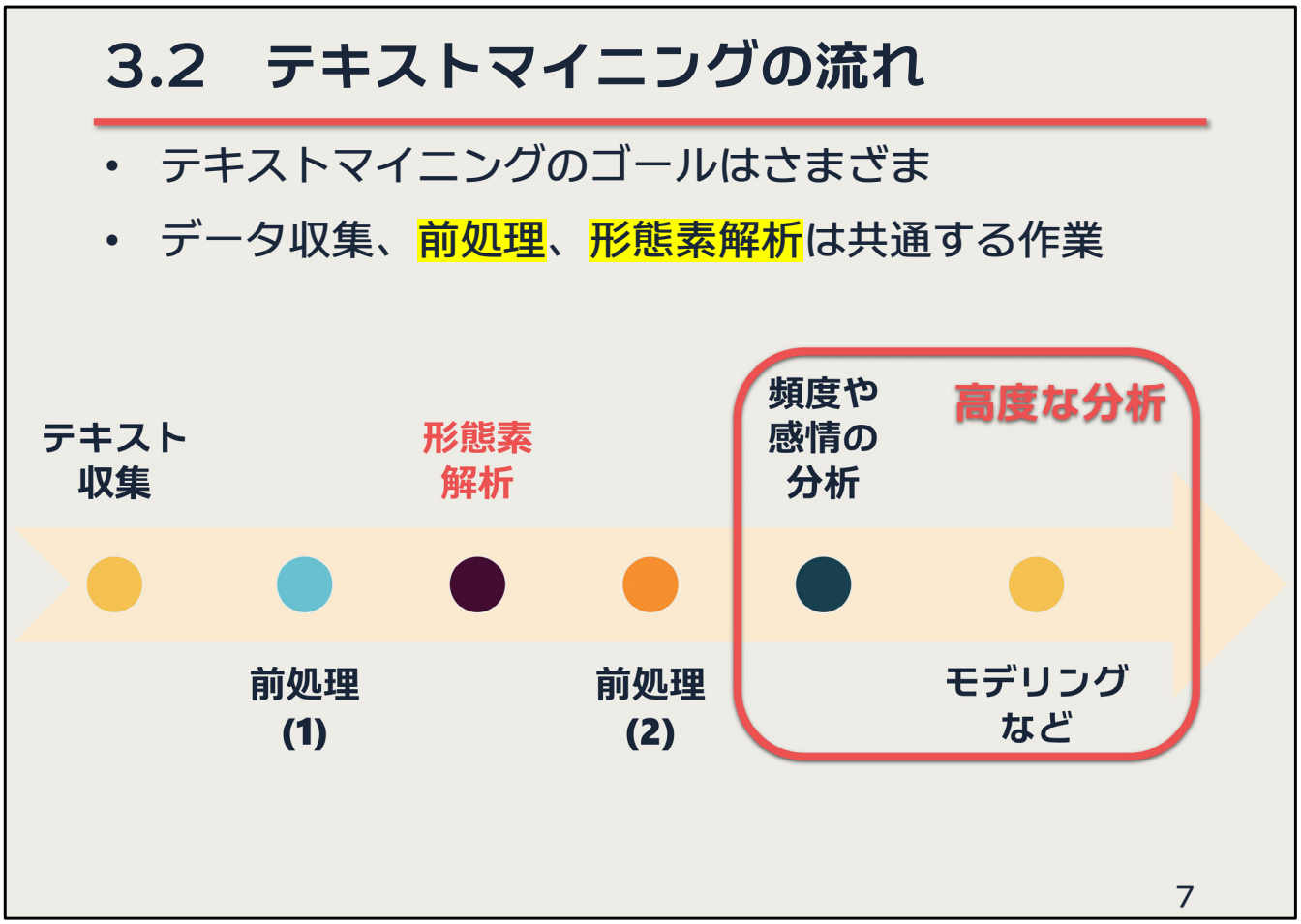


ツイートの感情分析をした例

6

テキストマイニング（Text mining）は、テキストデータから何らかの知見を得る分析アプローチの総称です。マイニング（Mining）は、元は採鉱、採掘といった意味ですが、「データに潜む価値を見つけ出す（掘り出す）」といった文脈で使われています。

SNSの投稿や日記、アンケートの自由記述欄には、数値による計測が難しい、アンケートの選択肢では表現しにくいヒトの意識や感情などが含まれています。それらを、テキストデータから「採掘する」ことで、現在のトレンドを把握したり、あるトピックについて、ヒトがポジティブ、ネガティブなどどのような感情を抱いているかを分析できます。

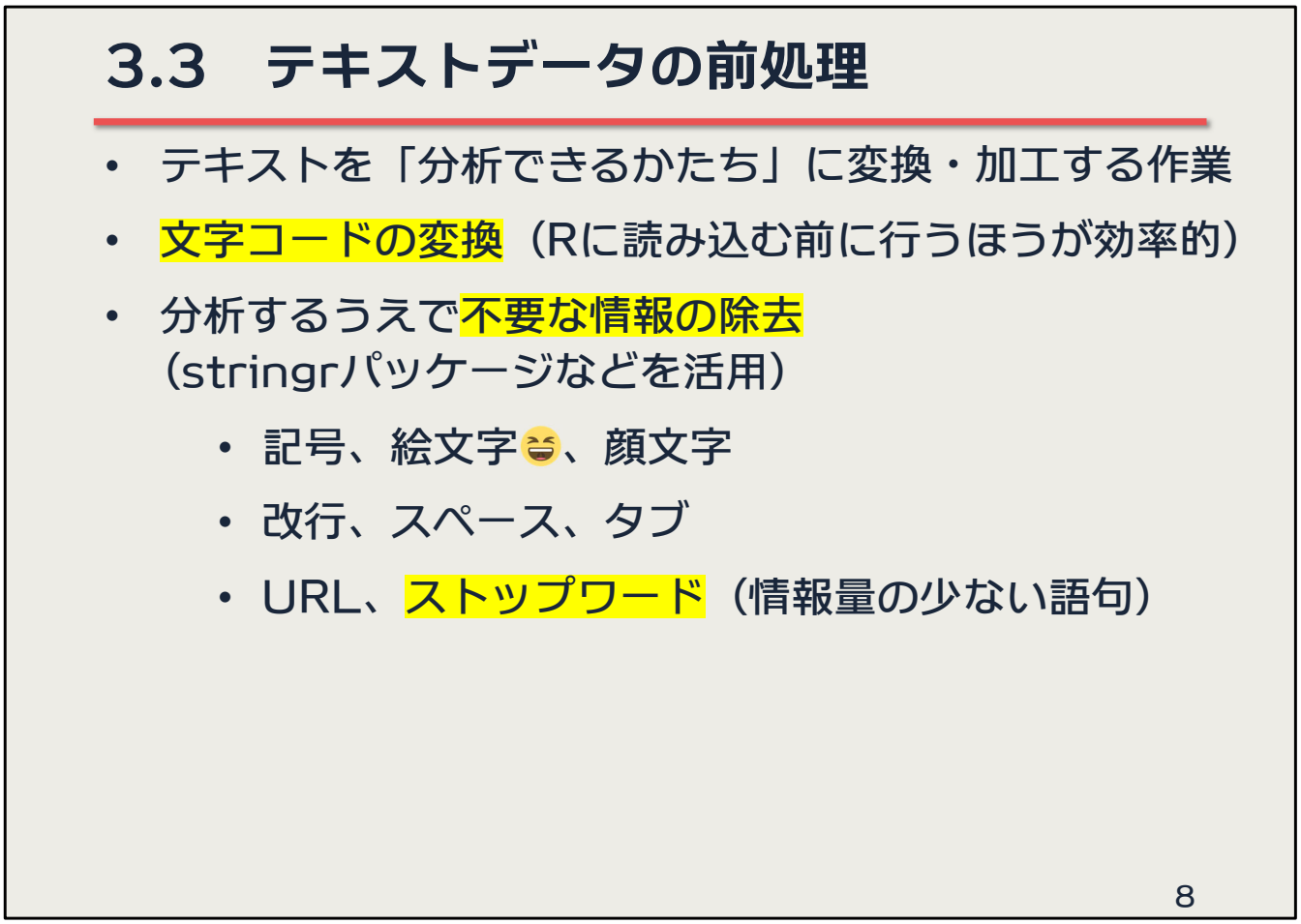


ここでは、テキストマイニングを実行する際の基本的な流れを示しています。当然ですが、はじめにテキストデータを集める必要があります。前述のように、SNSのAPIやオープンデータ、アンケートを実施するなどして、十分な量と質のテキストを収集します。1文の長さにも注意が必要です。数単語だけのツイートの羅列から、深い洞察を得ることは難しいでしょう。

収集したテキストは、そのままでは「データ」として活用可能な状態ではありません。さまざまな前処理を適用する必要があります。代表的な処理に形態素解析があります。詳細はこの後紹介します。

その後、データに対してどのような分析を行うかは、さまざまです。単語（形態素）の頻度をカウントすることで、トレンドを把握したり、形態素の頻度を特徴量に、テキストのカテゴリを目的変数として、「このような形態素が多く出現するテキストはクレームである」といった分類モデルを作成することもあります。

いずれのプロセスも、Rで容易に実行可能なパッケージが多く提供されており、それらを活用することで、さまざまなテキストマイニングを行うことができます。



テキストデータを分析するには、表構造のデータや数値データ以上にさまざまな前処理が必要です。はじめに、テキストによっては、文字コードの変換を行います。Rは2022年4月にリリースされた4.2.0以降、どのOSでも標準の文字コードがUTF-8になりました。そのため、UTF-8以外で作成されたテキストを扱うには、UTF-8に変換して読み込みます。読み込みの際に `read.csv(..., fileEncoding="文字コード")` などとして変換することもできますが、ファイルが多数ある場合、先にOSのコマンドなどで変換したほうが簡単でしょう。

次に、分析するうえでは不要な文字情報を除去します。例えば、記号（！？＞＜@＊など）や絵文字（😂👹）などは、人間は意味を読み取ることができますが、コンピューターにとっては理解が難しいものです。そのため、正規表現などを使って除去します。また、改行や(単語の区切りなど意味のあるもの以外の) スペース、タブなどもテキストの切れ目を誤認させることがあるので、取り除きます。そして、URLや、「ストップワード」と呼ばれる情報量の少ない単語も除去します。ストップワードとは、「あれ」「それ」といった指示語など、その単語だけでは意味がない言葉です。ストップワードについては、自分でリストアップすることもできますが、例えば京都大学の研究成果として公開されているファイルを使うことができます。

- slothlib http://svn.sourceforge.jp/svnroot/slothlib/CSharp/Version1/SlothLib/NLP/Filter/StopWord/word/

なお、ストップワードを単純なマッチングで除去すると、意味のある単語を「破壊」してしまうことになります。そのため、次に取り上げる形態素解析を先に行い、その結果からストップワードを除去することが一般的です。

日本語テキストを分析するためには、形態素解析（Morphological Analysis）が必要です。形態素解析は、テキストを「意味を持つ最小単位」である形態素に分割する処理です。形態素解析は、形態素解析器という専用のソフトウェアを使って行います。形態素解析器は何種類かあり、ここでは著名なオープンソースソフトウェアを挙げています。特に広く使われるのが、MeCabです。Windowsでは、64bit OS用に改変した「野良ビルド」が使われることが多いです。macOSやLinuxではパッケージ管理システムからインストールできます。

Chasen（茶筅）は、MeCab作者も所属していたNAIST（奈良先端科学技術大学院大学）の研究成果です。現在ではMeCabが上位互換なので、積極的に利用することはないでしょう。

Juman, JUMAN++は、東京大学の研究成果で、MeCab, Chasenとは異なる言語モデルでの解析を行います。Rから利用するのは簡単ではないようです。以上のソフトウェアは、10年から20年前に開発されたものですが、近年新たに開発されたのが、ワークスアプリケーションズ社が公開したSudachiです。ディープラーニングなど、現代のニーズに対応し、現在も更新が続いています。

近い将来、Sudachiが主流になるかもしれませんが、現在はMeCabが広く使われており、本講義でもMeCabをRから使う方法を紹介します。

- MeCab: https://taku910.github.io/mecab/
- MeCab 64bit Windows版: https://github.com/ikegami-yukino/mecab/releases/tag/v0.996.2
- Sudachi: https://nlp.worksap.co.jp/

3.5　テキストデータの分析

- 分析するには、数値に変換する必要がある
- 基本的なアプローチとして、形態素の頻度をカウントする
- 形態素と頻度の対応表をBoW（Bag of Words）という
- その他、形態素ごとにあらかじめ算出した感情価を割り当てることも

形態素	頻度
楽しい	150
良かった	117
面白い	99
・・・	・・・

BoWに基づいた分析「ワードクラウド」のイメージ

10

テキストデータを分析するには、テキストを何らかの数値に変換する必要があります。最も典型的なアプローチとして、テキスト中の形態素の頻度をカウントし、その度数分布をテキストの特徴とする方法があります。そのような、形態素と頻度の対応表のことを、BoW (Bag of Words) と言います。BoWを作成することで、複数のテキストの特徴を比較したり、テキストのジャンルなどを目的変数としたモデリングのための特徴量（説明変数）として用いることができます。

また、形態素ごとに、あらかじめ「その形態素がどの程度ポジティブか / ネガティブか」などをあらわす感情価という数値を求めておき、それを付与することもあります。形態素単位の感情価を合計したり平均することで、テキストの感情を表現できます。感情価については、以前は形態素ごとの算出が極めて困難でしたが、この1、2年で生成AI研究の過程で得られたディープラーニングモデルを用いることで容易になりました。また実世界での用例を学習しているため、実態に沿った評価ができるようになりました。ただし、Pythonでの実装が主流で、Rのパッケージはあまり存在しません。最終回で紹介するreticulateパッケージを使い、RからPythonを呼び出して利用することになるでしょう。

- CRAN - Package transforEmotion
  https://cran.r-project.org/web/packages/transforEmotion/index.html
- Transformersによる日本語の感情分析を試す | npaka
  https://note.com/npaka/n/n1766c894c1f2

4. Rによるテキストマイニング

11

ここからは、Rでテキストマイニングをするにはどうしたらよいかを紹介していきます。毎度のことながら、高度な理論的な説明は一切ないので、「パッケージを使えばこんなことができる」というように概要を捉えていただければ十分です。

4.1　テキストマイニングのためのRパッケージ

- 他にも、山ほどある
- rtweet: RからTwitter APIにアクセスする
- RMeCab: RからMeCabを呼び出す
- RcppMeCab: RからMeCabを呼び出す（C++版）
- rjavacmecab: RからMeCabを呼び出す（Java経由）
- gibasa: RからMeCabを呼び出す（C++版）
- sudachir: RからSudachiを呼び出す
- tidytext: tidyなテキストマイニングを行うための フレームワーク
- udpipe: UDPipeライブラリの辞書を利用するための パッケージ
- quanteda: テキストデータの量的分析のためのフレームワーク
- syuzhet: 感情分析を行うためのパッケージ

12

Rでテキストマイニングを行うためのパッケージは、多数公開されています。形態素解析器を呼び出すためのパッケージは、日本特有のものですが前述のMeCabやSudachiに対応したものが複数あります。また、形態素解析からその他の前処理、定量的分析、形態素間のつながりなどの専門的な分析までをカバーするフレームワークもいくつかあります。

やはり、Posit社のスタッフが中心になって開発しているtidytextパッケージが、これからの主流になっていくものと思います。しかし、"tidyなRプログラミング" の文化と言語統計学などの研究領域は少し異なるので、今後もいくつかのフレームワークが並列的に使われていくかもしれません。

- tidytext: https://juliasilge.github.io/tidytext/
- udpipe: https://bnosac.github.io/udpipe/en/index.html
- quanteda: https://quanteda.io/index.html
- syuzhet: https://github.com/mjockers/syuzhet

4.2　テキストデータの前処理

- 一般的に、テキストマイニングは名詞・動詞・形容詞・形容動詞を対象とすることが多い
- それ以外の品詞は情報量が少ないので除去する
- 記号、絵文字、顔文字なども除去する
- 完璧な除去は不可能（頻度が小さくなれば良い）
- 各フレームワークにはそれらの前処理を担当する関数がある
- stringrパッケージの `str_replace_all()` 関数などを使い地道に指定する

13

テキストデータの前処理は、数値や表構造データに比べてはるかに複雑です。形態素ひとつひとつについて、必要か不要かを判断し、取捨選択するプログラムを記述する必要があります。また、扱うテキストによっても、出現する形態素のパターンが異なるので、前処理の方法（プログラム）が変わってきます。

とはいえ、「定番」と言えるような処理もありますので、それらを紹介します。まず、テキストマイニングでは一般的に動詞、名詞、形容詞を中心に分析します。他に形容動詞なども加わることがありますが、助詞や助動詞、副詞などの品詞からはあまり興味深い情報は得られないので、形態素解析の結果得られた品詞情報を用いて除去します。また、記号や絵文字なども除去します。絵文字を分析する事例もありますが、ビジネスではそこまですることは多くありません。なお、不要な品詞、文字列を完全に除去することは不可能ですので、妥協することも必要です。

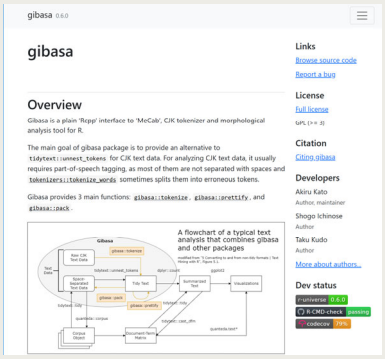
具体的な処理の方法については、次のページにも例を示していますが、tidytextなどテキストマイニングのためのフレームワークには、よくある前処理を関数化したものが含まれています。一方、文字単位での処理には、stringrパッケージの `str_replace_all()` 関数などを使うことが多いでしょう。

4.2　テキストデータの前処理

```
text <- "..."
# URLをあらわす正規表現
url_regex <- "http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*¥¥(¥¥),]|
              (?:%[0-9a-fA-F][0-9a-fA-F]))+"

parsed_text <- text %>%
    str_replace_all(url_regex, "") %>% # URLを除去する
    # 日本語の場合
    # ひらがな、カタカナ、漢字、数字、"ー" だけを残す
    str_replace_all("[^¥¥p{Hiragana}|¥¥p{Katakana}|¥¥p{Han}|¥¥p{N}|ー]", " ") %>%
    str_replace_all(" +", " ") %>% # 連続するスペースを1つにまとめる
    str_replace_all("^ | $", "") %>% # 行頭や行末のスペースを除去する
    str_replace_all("¥¥n", "") %>%
    # 形態素解析（後述）してスペース区切りで出力する
    map(~paste(flatten_chr(RMeCabC(., mypref = 1)), collapse=" ")) %>%
    flatten_chr() %>% as_tibble()
```

4.3　RMeCabパッケージなどによる形態素解析

- Rから形態素解析器を呼び出すためのパッケージが複数ある
- 少し前まではRMeCab一択だったが、現在ではgibasaなども利用できる
- RMeCabは形態素解析と基本的な加工、集計ができる
- gibasaはモダンで高機能なパッケージ（ただしまだ歴史が浅い）



15

ここから、Rから形態素解析器を呼び出して形態素解析などを行うことができるパッケージをいくつか紹介します。前述のように、RMeCabを使うことが一般的ですが、MeCabを別のアプローチで呼び出すパッケージもいくつか開発されており、自分のプログラミングスタイルに合うものを選びましょう。今回は、MeCab系パッケージとSudachiを使う例を紹介します。

4.4　RMeCabパッケージによる形態素解析

- RMeCabC() 関数でベクトルに対する形態素解析ができる
- ほとんどの処理は docDF() 関数のオプションで指定できる
- 工夫すればtidy dataの文脈に乗せることもできる

```
library(RMeCab)

txt <- "RMeCabは、「Rでテキストマイニング」というジャンルにおける定番中の
定番です。徳島大学の石田基広教授によって開発されています。"

# mypref = 1 で原形を返す
unlist(RMeCabC(txt, mypref = 1))
##          名詞          助詞          記号
##     "RMeCab"          "は"          "、"
##          記号          名詞          助詞
##          "「"           "R"          "で"...
```

16

RMeCab（http://rmecab.jp/wiki/index.php?RMeCab）は、「Rでテキストマイニング」というジャンルにおける定番中の定番です。徳島大学の石田基広教授によって開発されています。MeCabには、他のプログラムから呼び出すためのC言語のインターフェースがあり、RMeCabはそれを使い、R→C言語→MeCabという流れでアクセスしています。RMeCabパッケージは多数の関数を提供しており、単純に形態素解析だけを行うものから、後述の頻度分析や共起分析を行うためのものまで、幅広くカバーしています。ただ、多くの機能はdocDF() 関数のオプションとして実装されており、docDF() 関数だけでほとんどの処理を実行できます。一時期、更新が停滞し、大量のテキストを与えた際に不安定になったり、tidy dataや新語辞書（注）などとの相性が悪かった頃もありましたが、現在は再度活性化し、十分な機能と性能を有しています。

注: MeCabに付属する辞書は、2007年に作成されたもので、それ以降にあらわれた新しい言葉を知りません。そのため、SNSなどの分析では、意図した結果が得にくい場合があります。そこで、定期的に辞書を更新し、最新のトレンドに対応するmecab-ipadic-NEologdプロジェクトがありましたが、2020年半ばに更新が停止してしまいました。以降は後継のプロジェクトもなく、最終版のNEologd辞書を使うというのが一般的です。

- 参考: GitHub - neologd/mecab-ipadic-neologd https://github.com/neologd/mecab-ipadic-neologd/blob/master/README.ja.md

なお、Posit Cloudにおいては、クラウド特有の環境からMeCabのインストールに工夫が必要で、NEologd辞書についてはメモリ不足でインストールできません。

4.4　RMeCabパッケージによる形態素解析

```
# RMeCabText(): ファイル名を指定して形態素解析する
# 返り値は形態素ごとにリストに格納されたベクトルとして出力される
RMeCabText("sample_texts/sample_text01.txt")

## file = sample_texts/sample_text01.txt
## [[1]]
##  [1] "ワールドカップ" "名詞"           "固有名詞"       "一般"
##  [5] "*"              "*"              "*"              "ワールドカップ"
##  [9] "ワールドカップ" "ワールドカップ"
##
## [[2]]
##  [1] "アルゼンチン" "名詞"   "固有名詞"     "地域"         "国"
##  [6] "*"            "*"      "アルゼンチン" "アルゼンチン" "アルゼンチン"
...
```

17

ここで、RMeCabが提供する関数をいくつか紹介します。前のページでは、RMeCabC()関数を取り上げていますが、これはベクトルを引数に与えると形態素解析した結果を、リストに格納された名前付きベクトル、という少しややこしい構造で返します。リストを解除（flatten）する unlist() 関数を使うと一連のベクトルになります。mypref = 1 とオプションを指定すると、形態素をテキスト中の出現形から、辞書に登録されている原形に変換して出力します。

RMeCabText() 関数は、ファイルを引数に取って、形態素解析の結果を返します。特に指定すべきオプションはありません。RMeCabは関数ごとに出力形式が異なり、やや不便です。

docDF() 関数は、前述のように多くの機能をオプションで指定できる、ある種の「万能関数」です。引数には、ファイルやデータフレームを指定できます。また、フォルダーのパスを指定すると、中のテキストファイルをすべて読み込んで処理します。ここでは、主要なオプションを使用した例を示しています。

詳細は、RMeCabパッケージのドキュメントを参照してください。ただし、更新されていないのか、実際の動作と一致しない記述もあります。

- RMeCabFunctions http://rmecab.jp/wiki/index.php?RMeCabFunctions

4.4　RMeCabパッケージによる形態素解析

```
# docDF(): ある種の万能関数。入力はファイルやフォルダー（複数ファイル）、
# データフレームに対応。type = 1 で形態素単位の処理、Genkei = 0 で
# 出現形を原形に変換する、nDF = 1 で出力をデータフレームにする
res <- docDF("sample_texts/sample_text01.txt",type = 1,Genkei = 0,nDF = 1)
```

```
res
```

```
##       N1 POS1     POS2 sample_text01.txt
## 1 1978年 名詞 固有名詞                1
## 2 1986年 名詞 固有名詞                1
## 3   19日 名詞 固有名詞                1
## 4      2 名詞       数                2
## 5      3 名詞       数                2
## 6   35歳 名詞 固有名詞                1
...
```

4.4　RMeCabパッケージによる形態素解析

```
# pos = 品詞（大分類）で、処理対象を限定
res <- docDF("sample_texts/sample_text01.txt", type = 1,
   pos = c("名詞", "動詞", "形容詞"), Genkei = 0, nDF = 1)

res

##       N1 POS1     POS2 sample_text01.txt
## 1 1978年 名詞 固有名詞                1
## 2 1986年 名詞 固有名詞                1
## 3   19日 名詞 固有名詞                1
## 4      2 名詞       数                2
## 5      3 名詞       数                2
## 6   35歳 名詞 固有名詞                1
...
```

4.4　RMeCabパッケージによる形態素解析

```
# N = 2以上でN-gramを抽出。N-gramは形態素の組み合わせのこと
res <- docDF("sample_texts/sample_text01.txt", type = 1, Genkei = 0,
   nDF = 1, N = 3)

res
```

```
##       N1   N2           N3           POS1                    POS2  sample_text01.txt
## 1 1978年   の     自国開催 名詞-助詞-名詞 固有名詞-連体化-固有名詞                   1
## 2 1986年   の メキシコ大会 名詞-助詞-名詞 固有名詞-連体化-固有名詞                   1
## 3   19日   、       ドーハ 名詞-記号-名詞   固有名詞-読点-固有名詞                   1
## 4      2   で         制す 名詞-助詞-動詞           数-格助詞-自立                   1
## 5      2 連覇           を 名詞-名詞-助詞       数-サ変接続-格助詞                   1
## 6      3   で       延長戦 名詞-助詞-名詞       数-格助詞-固有名詞                   1
...
```

20

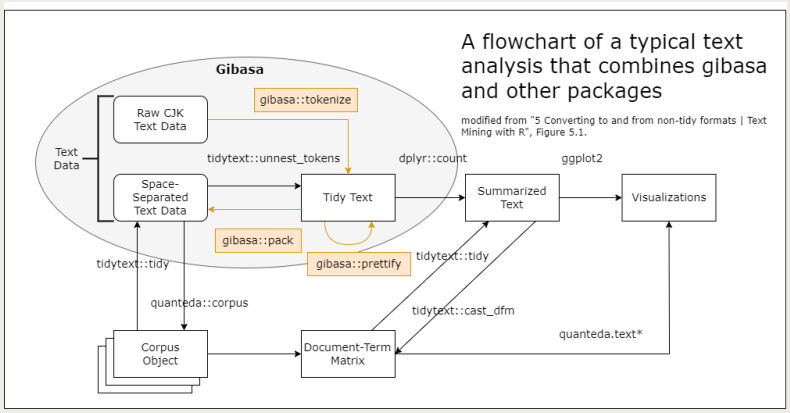
4.4　RMeCabパッケージによる形態素解析

```
# 引数にフォルダーを指定すると複数のテキストファイルを読み込んで
# 解析、出力する。出力には、どのテキストで何回出現したかの情報も付与される
res <- docDF("sample_texts", type = 1, pos = c("名詞", "動詞", "形容詞"),
   Genkei = 0, nDF = 1)

res

##     N1 POS1     POS2 long_sample_text01.txt sample_text01.txt
## 1    % 名詞     接尾                       2                 0
## 2 .com 名詞 固有名詞                       0                 0
## 3   00 名詞       数                       0                 0
## 4   04 名詞       数                       0                 0
## 5   09 名詞       数                       0                 0
## 6 0900 名詞       数                       0                 0

...
```


21

4.5　gibasaパッケージによる形態素解析

- 2023年4月にCRANに登録された
- 辞書と mecabrc ファイルがあれば動作し、MeCab自体は不要
- `install.packages("gibasa")`



gibasaと他パッケージを組み合わせたテキストマイニングのフロー

22

gibasaは、RからC++言語を利用できるRcppパッケージを用いてMeCabにアクセスするパッケージです。ユーザーとしては特に意識する必要はありませんが、Rcppを使うほうが、プログラムの設計がしやすく、性能も期待できるものと思われます。

- An Alternate Rcpp Interface to MeCab・gibasa
  https://paithiov909.github.io/gibasa/index.html
- RとMeCabによる日本語テキストマイニングの前処理
  https://paithiov909.github.io/textmining-ja/index.html
- 参考: みんなのRcpp https://teuder.github.io/rcpp4everyone_ja/

gibasaパッケージが提供する関数はRMeCabよりは少なく、形態素解析に特化しています。特によく使うのが `tokenize()` 関数です。引数にデータフレームを指定します。このページでは、readtextパッケージの `readtext()` 関数を使い、フォルダー内のファイルをまとめて読み込み、1件を1行の形式でデータフレームにしています。フレームワークを使った「モダンな」テキストマイニングでも、この「1行1文書」という形式は一般的です。なお、`tokenize()` 関数はreadrパッケージが提供する関数と名前が重なっているので、パッケージを読み込む順番などに注意が必要です。

`tokenize()` 関数で形態素解析した結果を見やすく整形して出力するのが `prettify()` 関数です。

4.5　gibasaパッケージによる形態素解析

```
library(readtext)
library(gibasa)

files <- paste0("sample_texts/", list.files("sample_texts",
       pattern = "^sample"))
df <- readtext(files) # フォルダー内のテキストファイルをまとめて読み込む
# 他のパッケージが提供する関数と名前が被るためパッケージ名を指定する
res <- gibasa::tokenize(df) # 形態素解析を行う
prettify(res)
```

```
##   doc_id            sentence_id token_id token          POS1  POS2     POS3  ...
##   <fct>                   <int>    <int> <chr>          <chr> <chr>    <chr> ...
## 1 sample_text01.txt           1        1 ワールドカップ 名詞  固有名詞 一般  ...
## 2 sample_text01.txt           1        2 アルゼンチン   名詞  固有名詞 地域  ...
## 3 sample_text01.txt           1        3 36年ぶり       名詞  固有名詞 一般  ...
## 4 sample_text01.txt           1        4 3回目          名詞  固有名詞 一般  ...
...
```

4.6　sudachiによる形態素解析

- sudachirパッケージがあるが、現状うまく動作しない
- RからPython（sudachipy）を呼び出して使うほうが簡単そう
- RからPythonを呼び出すreticulateパッケージは最終回に紹介予定

```
library(reticulate)

sudachipy <- import("sudachipy")
dict <- sudachipy$Dictionary(dict = "full")
tokenizer <- dict$create()
res <- tokenizer$tokenize(txt)

res_it <- iterate(res) # 独自構造のPythonオブジェクトをRのリストに変換する
# tidyverseな map() 系関数でもっとシンプルに書けそうですが
...
```

24

次に、Sudachiを使う例を紹介します。SudachiはJavaとPythonで並行して開発されており、Pythonではsudachipyライブラリとして利用できます。

- GitHub - WorksApplications/SudachiPy
  https://github.com/WorksApplications/SudachiPy

Rからは、Pythonを呼び出すための仕組みであるreticulateパッケージを介してsudachipyを使うのが容易そうです。ここでは、下記の記事で紹介されている方法を示しています。reticulateについては、本講義の最終回で紹介する予定です。

- Rでも、良いSudachi Lifeを送りたい - Qiita https://qiita.com/Mitz-TADA/items/ade1806dd4644bfa5a37

SudachiをRから利用するためのsudachirパッケージ（ https://uribo.github.io/sudachir/ ）もありますが、現状CRAN版もGitHub版も、Posit Cloudではうまく動作しません。

4.6 sudachiによる形態素解析

```
df <- data.frame(
    surface = sapply(res_it, function(x) x$surface()),
    base = sapply(res_it, function(x) x$dictionary_form()),
    pos1 = sapply(res_it, function(x) unlist(x$part_of_speech()[1])),
    pos2 = sapply(res_it, function(x) unlist(x$part_of_speech()[2])),
    pos3 = sapply(res_it, function(x) unlist(x$part_of_speech()[3])),
    pos4 = sapply(res_it, function(x) unlist(x$part_of_speech()[4])),
    pos5 = sapply(res_it, function(x) unlist(x$part_of_speech()[5])),
    pos6 = sapply(res_it, function(x) unlist(x$part_of_speech()[6]))
)
df
```

```
##   surface              base     pos1       pos2     pos3 pos4 ...
## 1  RMeCab            rmecab     名詞   普通名詞      一般    * ...
## 2      は                は     助詞     係助詞         *    * ...
## 3       、                 、 補助記号       読点         *    * ...
## 4       「                 「 補助記号     括弧開         *    * ...
...
```

5. さまざまな分析手法

26

ここからは、形態素解析の結果を基にしたいくつかの分析手法を紹介します。この節では、比較的クラシックな、「機械学習ブーム以前」の手法を中心に取り上げます。

今回は、そのうちの1つだけ、頻度分析について取り上げます。他は、次回（年明け）の前半に紹介します。

5.1　頻度分析

- RMeCabFreq() 関数で形態素（原形）の頻度をカウントできる
- dplyrパッケージの filter() 関数や arrange() 関数と組み合わせて「トレンドワード」を知ることができる

```
# 名詞、動詞、形容詞に絞って頻度で並べ替える
RMeCabFreq("sample_texts/long_sample_text01.txt") %>%
   filter(Info1 %in% c("名詞", "動詞", "形容詞")) %>%
   filter(!Info2 %in% c("接尾", "非自立", "数")) %>%
   filter(!Term %in% c("する", "ある", "いる", "なる", "あれ", "これ",
          "それ", "いう", "ない")) %>% # ストップワードの除去
   filter(nchar(Term) > 1) %>% # 1文字の形態素を除去
   filter(Freq >= 3) %>%
   arrange(desc(Freq))
```

27

頻度分析は、名前の通り、テキストにおける形態素ごとの出現頻度をカウントする手法です。単純ですが、SNSの投稿に対して頻度分析をした結果がいわゆる「トレンドランキング」として扱われているように、結果を見るだけでさまざまな示唆が得られます。

手順としては、形態素解析の結果を出現形で group_by() して、それぞれの頻度をカウント (n()) するだけですので、どの形態素解析器・パッケージを使ってもできますが、RMeCabには頻度分析のための RMeCabFreq() 関数があり、簡単に実現できます。なお、RMeCabFreq() 関数はファイルしか引数に指定できません。

このプログラム例では、結果のうち、助詞や副詞、一部の品詞にはあまり意味・関心がないので、後段の処理で除去します。また、出現頻度が少ない形態素も除去しています。最後に、出現頻度の降順で並べ替えて出力しています。

5.1　頻度分析

```
files <- paste0("sample_texts/",
   list.files("sample_texts", pattern = "^sample"))
df <- readtext(files)
res <- gibasa::tokenize(df)
res %>%
    mutate(pos1 = unlist(str_split(feature, ","))[1],
        pos2 = unlist(str_split(feature, ","))[2],
        pos3 = unlist(str_split(feature, ","))[3]) %>%
    group_by(token, pos1, pos2, pos3) %>%
    summarise(freq = n()) %>%
    filter(pos1 %in% c("名詞", "動詞", "形容詞")) %>%
    filter(!pos2 %in% c("接尾", "非自立", "数")) %>%
    filter(!token %in% c("する", "ある", "いる", "なる", "あれ",
          "これ", "それ", "いう", "ない", "まし", "から", "こと")) %>%
    filter(nchar(token) > 1) %>%
    filter(freq >= 3) %>%
    arrange(desc(freq))
```

28

　このページでは、gibasaパッケージを使った頻度分析の例を示しています（たぶんもっとシンプルに書けるでしょうが）。gibasaの tokenize() 関数は、複数のファイルをデータフレームに格納して与えられるので、ここではフォルダー内のテキストファイルをすべて集計して頻度を出力しています。

　gibasaもMeCabを呼び出しているので出力される情報はほぼ同じで、RMeCabの場合と同じようなフィルタリングをして集計できます。

5.1　頻度分析

```
##    token          pos1  pos2     pos3   freq
##    <chr>          <chr> <chr>    <chr> <int>
##  1 アルゼンチン   名詞  固有名詞 一般     10
##  2 優勝           名詞  固有名詞 一般     10
##  3 残高           名詞  固有名詞 一般      6
##  4 水素           名詞  固有名詞 一般      6
##  5 大会           名詞  固有名詞 一般      6
##  6 18日           名詞  固有名詞 一般      5
##  7 フランス       名詞  固有名詞 一般      5
##  8 レース         名詞  固有名詞 一般      5
##  9 ワールドカップ 名詞  固有名詞 一般      5
## 10 基金           名詞  固有名詞 一般      5
## # … with 38 more rows
```

6. まとめ

6.1　今日の内容

- テキストデータの価値
- テキストマイニングの概要
- Rによるテキストマイニング
    - テキストデータの前処理
    - 形態素解析
    - 頻度分析

31

6.2　次回までの課題

- Posit Cloudプロジェクト内の
  `13_text_mining_01_exercise.R` について、
  指示に従ってプログラムを作成してください
- 編集したファイルは、ファイル一覧でチェックを
  入れ、[more] メニューから [Export] を選択し、
  [Download] ボタンを押してダウンロードして
  ください
- ダウンロードしたファイルを、Classroomの
  課題ページから提出してください
- **提出期限: 2024-01-08 23:59**

32