応用プログラミング3

第14回　日本語テキストマイニング（2）

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

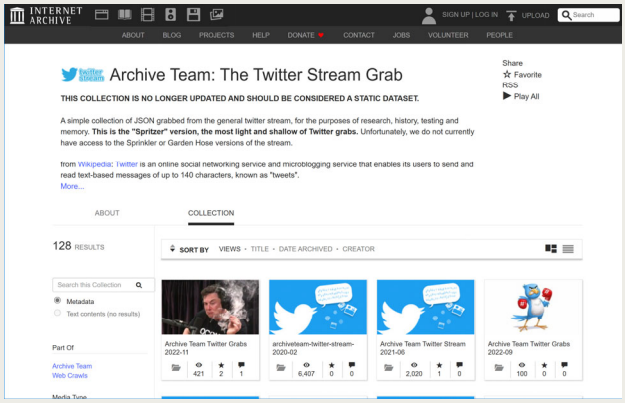
2. テキストデータの収集

2

今回も、前回に引き続きRによるテキストマイニングの方法を紹介していきますが、先に補足として、テキストデータの収集方法について紹介します。

2.1　Xデータのアーカイブ

- Internet ArchiveプロジェクトでStreaming APIで収集したデータをアーカイブしている
- 数GB単位で巨大だが、ダウンロードして展開すればrtweetパッケージの parse_stream() 関数で読み込める
- Xのデータを分析する練習としては、これを使えばよいのでは



https://archive.org/details/twitterstream

3

テキストデータのソース（収集元）として、X（旧Twitter）は非常に魅力的です。しかし、2023年以降、XはAPIを有料化し、現実的に個人では利用できなくなりました。もし、研究やビジネスにおいて料金を支払った上でAPIを利用する時のために、データの分析方法を学びたい、という場合は、すでに取得され、公開されているアーカイブデータを利用するとよいでしょう。（筆者も2023年になって知りましたが）Internet Archiveプロジェクトが、不定期ですが1か月ごとのSample Stream API（ランダムに一部のポストを取得し続けるストリーミングAPI）で取得したデータを公開しています。

- 参考: The Twitter Stream Grab https://archive.org/details/twitterstream

インターフェースがややわかりにくいですが、"202X-XX" とある項目をクリックし、右の "TAR" から日別のアーカイブをtar形式でダウンロードできます。それぞれ、3GB弱と巨大で、物理的な距離もあるので、ダウンロードには時間がかかります。tarファイルの中はさらに1分ごとのgzファイルになっているので、展開します。このアーカイブデータは、Twitter API v1で取得していたようで、rtweetパッケージの parse_stream() 関数で読み込めます。

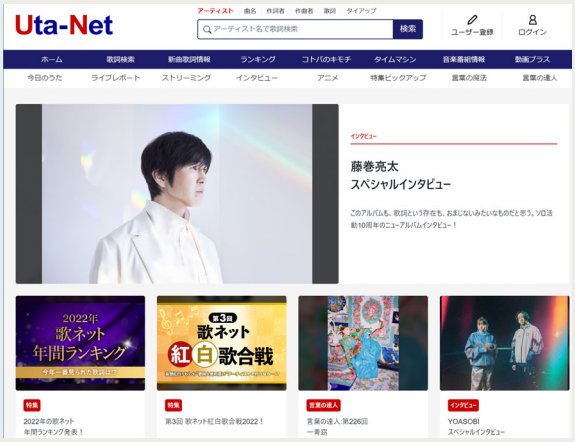
2.1　Xデータのアーカイブ

```
# 公開されている年月日のデータを指定してダウンロードする（重い、遅い）
download.file("https://archive.org/download/archiveteam-twitter-stream-
YYYY-mm/twitter-stream-YYYYmmdd.tar")
# ダウンロードしたtarファイルを展開する（Rでやる必要はない）
untar("twitter-stream-YYYYmmdd.tar")
# 1分ごとにgz圧縮されたJSONファイルを展開する（Rでやる必要はない）
library(R.utils)
gunzip("YYYYmmdd/YYYYmmddHHMMSS.json.gz")
# ツイートデータを読み込む
library(rtweet)
df <- parse_stream("YYYYmmdd/YYYYmmddHHMM00.json")
```

4

2.2　歌詞データの収集

- 歌詞サイトは右クリックやコピペ禁止になっていることが多いが、HTMLなのでスクレイピングできる
- アクセス間隔を空けるなど、サイトの負荷にならないよう注意する
- そもそも歌詞は独特な表現が多く、テキストマイニングしづらい



https://www.uta-net.com/

5

次に、J-POPを中心とした楽曲の歌詞を取得する方法を紹介します。基本的には、第12回で紹介したWebスクレイピングなので、いろいろな歌詞サイトから取得できますが、ここでは「歌ネット」(https://www.uta-net.com/) を例にします。…というのは、Pythonでスクレイピングした例があったからですが。

- 参考: 歌ネット（Uta-Net）から歌詞をスクレイピングする
  https://zenn.dev/robes/articles/00e86185677fb5

なお、歌ネットではアクセス元のIPアドレスとUserAgentの組み合わせなどで制限をしているようで、Google ColabやPosit Cloudなど外国のネットワークからは接続拒否される可能性があります。その場合も、UserAgentを一般的なWebブラウザのものに設定すれば接続できます。

ここでは、プログラムの詳しい説明はしませんが、すでに紹介したhttrパッケージとrvestパッケージを用いて、複数のアーティストの歌詞を取得し、1つのデータフレームに統合する例を示しています。実際にハンズオンで使用する pops_lyrics.csv には、10人・グループのアーティストの歌詞データを格納しています。

2.2　歌詞データの収集

```
## # A tibble: 10 × 5
## # Groups:   id [10]
##       id artist          title              lyric
##    <dbl> <chr>           <chr>              <chr>
##  1     1 AKB48           アイスのくちづけ   花柄のカーテンを開ければ 降り注ぐ夏の太陽 …
##  2     2 乃木坂46        ここにはないもの   夜になって　街の喧騒も闇に吸い込まれて…
##  3     3 櫻坂46          I'm in             腕を思い切り伸ばしてみる 僕のこの指の先は…
##  4     4 日向坂46        あくびLetter       マホガニーのテーブルで何度も書き直すLetter…
##  5     5 米津玄師        アイネクライネ     あたしあなたに会えて本当に嬉しいのに…
##  6     6 キャンディーズ  哀愁のシンフォニー あなたの目が私を見て 涙うかべてたその顔がつらい…
##  7     7 さだまさし      愛                 あなたに会いたい 泣きたいほど会いたい…
##  8     8 吉田拓郎        あゝ青春           ひとつひとりじゃ淋しすぎる ふたりじゃ息さえもつまる部屋…
##  9     9 サンボマスター  アイ ウォン チュー 神様 街では無実の子が殺められています…
## 10    10 BUMP OF CHICKEN アカシア           透明よりも綺麗な　あの輝きを確かめにいこう…
```

```r
library(tidyverse)
library(httr)
library(rvest)

base_url <- "https://www.uta-net.com"

urls <- c("https://www.uta-net.com/artist/6636/0/1/", "https://www.uta-net.com/artist/6636/0/2/", "https://www.uta-net.com/artist/12550/", "https://www.uta-net.com/artist/29512/")

# User Agentを偽装する
pseudo_user_agent <- "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"

# データフレームの初期化
df <- data.frame(id = numeric(), artist = character(), title = character(), lyric = character(), url = character())



for (i in 1:length(urls)) {
    res <- GET(urls[i], user_agent(pseudo_user_agent))
    html <- content(res)
    links <- html %>% html_elements("td.sp-w-100")
    for (link in links) {
        href <- link %>% html_element("a") %>% html_attr("href")
        song_url <- paste0(base_url, href, collapse = "")
        song_res <- GET(song_url, user_agent(pseudo_user_agent))
        song_html <- content(song_res)
        song_title <- song_html %>% html_element("h2") %>% html_text2()
        artist_name <- song_html %>% html_element("h3") %>% html_text2()
        lyric <- song_html %>% html_element("div#kashi_area") %>% html_text2() %>% str_replace_all("¥¥n+", " ")
        tmp_df <- data.frame(id = 0, artist = artist_name, title = song_title, lyric = lyric, url = song_url)
        df <- df %>% bind_rows(tmp_df)
        print(paste0("now processing '", artist_name, "', '", song_title, "'"))
        Sys.sleep(1)
    }
}

df[["id"]] <- as.numeric(factor(df[["artist"]], levels = c("AKB48", "乃木坂46", "櫻坂46")))

write_csv(df, "pops_lyrics.csv")
```

3. さまざまな分析手法

8

ここから、前回の続きとして、いくつかのテキストデータの分析手法を紹介していきます。

3.1　ワードクラウド

- 形態素の出現頻度を文字の大きさとして表現したグラフィックス
- 頻度以外の情報はないが、「映える」のでよく使われる
- wordcloud, wordcloud2, ggwordcloudパッケージなどで作成できる

近年、メディアなどで「テキストデータを分析」といった場合にまず取り上げられるのが、ワードクラウド (Word Cloud) です。ワードクラウドは、形態素の出現頻度を文字の大きさや色で表現するグラフィックスです。実際のところ、「大きく表示される形態素がたくさん出現している」という以上の情報はないのですが、見た目にインパクトがあり、色使いや3D表現などによって、「映える」グラフィックスを作ることができるので、ビジネスにおいても多用されます。

Rには、ワードクラウドを作成するためのパッケージがいくつか存在します。古くから使われているwordcloudパッケージ、HTML + JavaScriptで (多少) インタラクティブに操作できるwordcloud2パッケージ、ggplot2の拡張としてのggwordcloudパッケージが広く使われています。今回は、それぞれを使ったプログラムの例を紹介します。なお、以降のプログラムは、「10人・グループの歌詞からまとめてワードクラウドを作成する」ために、並列処理 (`map()` 系関数) を行っています。そのためやや複雑になっていますが、本質は形態素と頻度を格納したデータフレームを、ワードクラウドを作成する関数に与える、というだけです。

はじめに、wordcloudパッケージを紹介します。

- CRAN - Package wordcloud https://cran.r-project.org/package=wordcloud
- 参考1: Rでウェブ解析:ワードクラウドを作ろう！「wordcloud」パッケージをお勧めします https://www.karada-good.net/analyticsr/r-79/

基本的には、`wordcloud()` 関数に、形態素と頻度からなるデータフレームやベクトルを与えるだけです。見た目をカスタマイズする主なオプションとして、`scale = c(文字の最大サイズ, 最小サイズ)` および `colors = 文字色のベクトル` があります。その他は、デフォルトのままでよいと思いますが、ドキュメントなどを参照してください。

```r
library(tidyverse)
library(gibasa)
df <- read_csv("./pops_lyrics.csv")
res <- df %>% select(-url) %>% gibasa::tokenize(., text_field = "lyric", docid_field = "title") %>%
prettify()

## ストップワードを定義する
mystopwords <- unique(c(stopwords::stopwords(language = "ja", source = "stopwords-iso")[1:124],
  stopwords::stopwords(language = "ja", source = "marimo")[11:194],
  "WOW", "Wow", "wow", "WO", "NO", "Hey", "Yeah", "AH", "OH", "Oh", "oh", "Na", "LA", "La",
"L¥u30fbA"))

res2 <- res %>% filter(POS1 %in% c("名詞", "動詞", "形容詞")) %>%
  filter(!POS2 %in% c("接尾", "非自立")) %>%
  mutate(token = if_else(is.na(Original), token, Original)) %>% # 原形があれば原形を選択
  filter(!token %in% mystopwords) %>%
 # ひらがな1文字、アルファベット1文字の形態素を除去
  filter(str_detect(token, "[^¥¥p{Hiragana}{1}|¥¥p{Latin}{1}]")) %>%
 # 辞書の原形が変なので
  mutate(token = if_else(token == "BIUTIFULビューティフル", "ビューティフル", token)) %>%
  group_by(id, artist, token) %>%
  summarise(count = n()) %>% # 頻度を算出
  mutate(prob = count / n()) %>% # アーティストごとに曲数が違うので頻度から出現率も算出
  select(artist, token, count, prob) %>%
  arrange(artist, desc(prob)) %>%
  top_n(100) %>% # 上位100語を抽出
  ungroup() # グループ化を解除して普通のデータフレームにする

library(wordcloud)

### グループごとにワードクラウドを作成し、画像として保存するため、
### 一連の処理を関数化する
make_wordcloud <- function(df) {
  # ファイル名を設定
  filename <- paste0("wordcloud_", str_replace_all(unique(df$artist), " ", "_"), ".png")
  tmp_df <- data.frame(token = df$token, prob = df$prob) # 形態素と出現率だけからなるデータフレーム
を作成
  ragg::agg_png(filename=filename, width = 1024, height = 1024, pointsize = 18, res = 96)
  par(family = "IPAexMincho")
  wordcloud(df$token, df$prob, scale = c(4, 1.5), colors = mypalette) # ワードクラウドを作成
  dev.off()
}

mypalette <- brewer.pal(8, "Dark2") # RColorBrewerパッケージのパレットから文字色を8色選択

### dplyrパッケージの group_map() 関数は、group_by() したグループ全体に
### 対して引数に指定した関数を適用する。パイプ経由のデータは .x として渡す。
### https://dplyr.tidyverse.org/reference/group_map.html
res2 %>% group_by(id) %>% group_map(~ make_wordcloud(.x))
```

3.1　ワードクラウド（wordcloud2）

```
library(wordcloud2)
library(webshot2)
library(htmlwidgets)

make_wordcloud2 <- function(df) {
  filename <- paste0("wordcloud2_", str_replace_all(unique(df$artist),
       " ", "_"), ".png")
  tmp_df <- data.frame(token = df$token,  count = as.numeric(cut(df$count,
       breaks = 10, labels = 1:10)))
  wc <- wordcloud2(tmp_df, fontFamily = "IPAexMincho", size = 2.5,
       minSize = 0.5)
  saveWidget(wc, "wc.html", selfcontained = FALSE) # HTMLとして保存
  # ヘッドレスでスクリーンショットを取得
  webshot("wc.html", filename, delay = 5, vwidth = 1280, vheight = 1280)
}

res2 %>% group_by(id) %>% group_map(~ make_wordcloud2(.x))
```

11

wordcloud2パッケージは、HTMLとJavascriptを使ってワードクラウドを作成します。こちらも、基本的には `wordcloud2()` 関数に形態素と頻度のデータフレームを与えれば作成できます。RStudioを使っている場合、"Viewer" ペインに表示され、マウスオーバーで頻度を確認できます。見た目をカスタマイズするオプションは多数用意されていますが、基本的には文字サイズを調整する `size` や `minSize` に、画面を見ながら適当な値を指定すれば、見栄えの良いワードクラウドが作成できるでしょう。他に、指定した画像や文字の形に形態素を配置するなど、もっと凝ったこともできますが、それらについてはドキュメントを参照してください。

- CRAN - Package wordcloud2 https://cran.r-project.org/package=wordcloud2
- 参考2: Rでお遊び：インタラクティブなワードクラウドが面白い「wordcloud2」パッケージ https://www.karada-good.net/analyticsr/r-538/

また、wordcloud2パッケージの出力を保存するには、HTMLを画像に変換する必要があります。ブラウザでスクリーンショットを撮ってもよいですが、ここではhtmlwidgetsパッケージの `saveWidget()` 関数でオブジェクトをファイルとして保存し、さらにwebshot2パッケージの `webshot()` 関数で、ヘッドレス（ウィンドウを開かず）にChromeブラウザでスクリーンショットを取得して保存しています。

- 参考3: 【R言語/Linux】コマンドでHTMLファイルをPDFファイルに変換する流れ - (O+P)ut https://www.mtioutput.com/entry/rlang-html-pdf
- 参考4: Rで解析：webページをスクリーンショット「webshot2」パッケージ https://www.karada-good.net/analyticsr/r-739/

3.1　ワードクラウド（ggwordcloud）

```
library(ggwordcloud)

### 文字色はランダムに選択する
res2[["colour"]] <- sample(mypalette, nrow(res2), replace = TRUE)

make_ggwordcloud <- function(df) {
  filename <- paste0("ggwordcloud_", str_replace_all(unique(df$artist), " ",
"_"), ".png")
  tmp_df <- data.frame(token = df$token, prob = df$prob)
  wc <- ggplot(df, aes(label = token, size = prob, colour = colour)) +
    geom_text_wordcloud(fontface = "bold") +
    scale_size(range = c(2, 14)) +
    theme_minimal()
  ggsave(filename, wc, device = ragg::agg_png, width = 1600, height = 1280,
units = "px", bg = "white")
}

### どうやっても明朝体にできないのであきらめる
res2 %>% group_by(id) %>% group_map(~ make_ggwordcloud(.x))
```

12

ggwordcloudパッケージは、ggplot2の拡張として、ワードクラウドを作成する geom を提供します。

- CRAN - Package ggwordcloud https://cran.r-project.org/package=ggwordcloud

使い方は、はじめに aes() 関数の引数に label = 形態素, size = 頻度や出現率 を指定します。その他、色なども適宜指定します。そして、geom_text_wordcloud() 関数を使うだけです。その他、形態素の配置や重なり、全体の形状などをさまざまにカスタマイズできますが、詳細はドキュメントを参照してください。

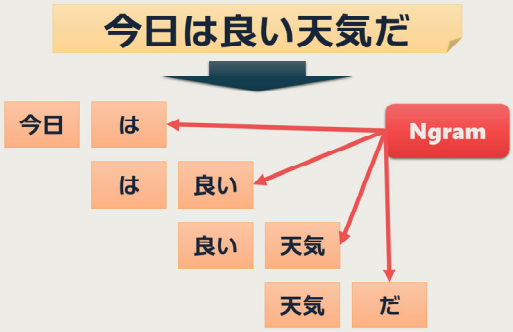
- ggwordcloud: a word cloud geom for ggplot2 https://cran.r-project.org/web/packages/ggwordcloud/vignettes/ggwordcloud.html

なお、CRANのggwordcloudパッケージは2019年から更新されていないようで、GitHubでの活動も低調（アップデートの気配はあるが、差分を見ると軽微）なので、そのうちフェードアウトするかもしれません。

今回は、選択肢として3つのパッケージを紹介しましたが、結果としては、wordcloud2パッケージを使うのが、いちばん「映える」結果になるのではないでしょうか。

3.2　Ngramと共起

- 「共に起きる」つまりよく一緒に出現する語句の組み合わせが共起
- 形態素単体の頻度を見ているだけではわからない「文脈」を検討できる
- 形態素単位とNgram単位で分析することがある
- Ngramはn個の形態素のセット（"今日-は-良い" など）
- 共起関係をもとに「共起ネットワーク図」を作成できる



13

ここからは、共起 (Collocation) 分析を取り上げます。名前の通り、「ある形態素とよくいっしょに出てくる形態素」を調べるアプローチです。共起関係を見ることで、単にどの形態素がトレンドなのかということだけでなく、それがポジティブな文脈で使われているのか、ネガティブな文脈で使われているのかなどを検討できます。

共起を分析する際には、形態素からNgramを作成することが一般的です。Ngramは、隣接するN個の形態素のペア、グループのことです。ここではgibasaパッケージ、同作者のテキスト前処理のためのaudubonパッケージとtidyverseを組み合わせたプログラム例を紹介します。本格的 (?) な共起分析では、直接隣り合って出現するものだけでなく、隣接する3から5語の組み合わせも分析します。詳細は、以下の参考URLを参照してください。

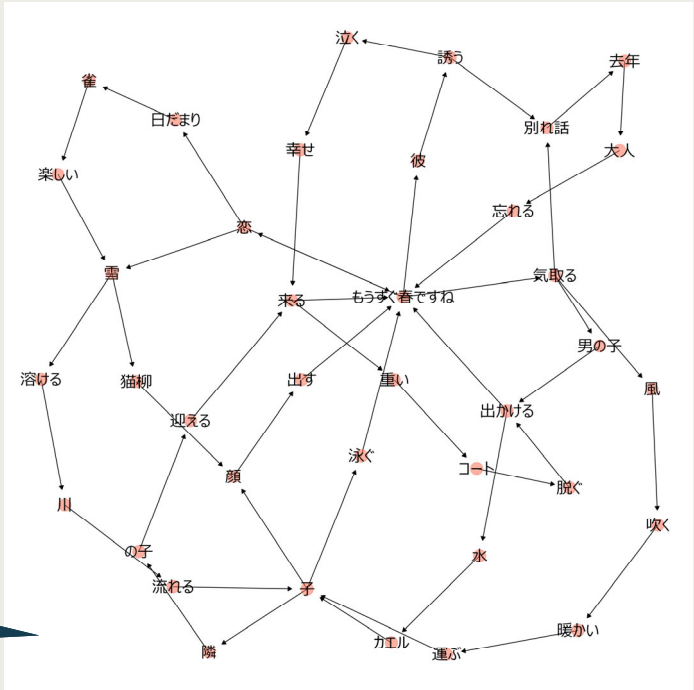
- 参考1: コロケーションと語彙分析
  http://www.ic.daito.ac.jp/~mizutani/mining/collocation.html
- 参考2: Chapter 5 コロケーション | RとMeCabによる日本語テキストマイニングの前処理 https://paithiov909.github.io/textmining-ja/collocation.html

このプログラムでは、はじめにこれまで同様、形態素解析を行い、不要な形態素を除去したり修正する前処理を行っています。次にaudubonパッケージの `ngram_tokenizer(n = 個数)` 関数を使い、N = 2のNgramを抽出するオブジェクトを作成しています。この関数は、gibasaパッケージの `tokenize()`, `prettify()` 関数で処理した結果を、n 個ごとに縦に「窓」をずらして抽出し、Ngramを作成します。今回は、この処理を関数化し、アーティストごとにグループ化したデータフレームに対して適用しています。なお、ここでは `group_modify()` 関数を使っています。`group_map()` 関数がリストを返す（ここまでの例では画像出力だったので問題なし）のに対し、`group_modify()` 関数はデータフレームを返します。

```r
library(tidyverse)
library(gibasa)
library(audubon)
# 歌詞データを読み込む
df <- read_csv("./pops_lyrics.csv")
mystopwords <- unique(c(stopwords::stopwords(language = "ja", source =
"stopwords-iso")[1:124],
  stopwords::stopwords(language = "ja", source = "marimo")[11:194],
  "WOW", "Wow", "wow", "WO", "NO", "Hey", "Hey!", "Yeah", "AH", "OH", "Oh",
"oh", "Na", "LA", "La", "L¥u30fbA"))

res <- df %>%
  select(-url) %>%
  gibasa::tokenize(., text_field = "lyric", docid_field = "title") %>%
  prettify(col_select = c("Original", "POS1", "POS2")) %>%
  # 原形があれば原形を選択
  mutate(token = dplyr::if_else(is.na(Original), token, Original)) %>%
  filter(POS1 %in% c("名詞", "動詞", "形容詞")) %>%
  filter(!POS2 %in% c("接尾", "非自立")) %>%
  filter(!token %in% mystopwords) %>%
  filter(str_detect(token, "[^¥¥p{Hiragana}{1}|¥¥p{Latin}{1}]")) %>%
  mutate(token = if_else(token == "BIUTIFULビューティフル", "ビューティフル",
token)) %>% # 辞書の原形が変なので
  mutate(token = if_else(token == "＝LOVE", "LOVE", token)) %>% # 同上
  # "-" を含む形態素は後の処理で問題が出るので、スペースに置換する
  mutate(token = str_replace_all(token, "-", " ")) %>%
  select(id, artist, doc_id, sentence_id, token)

# Ngramを作成するオブジェクトを生成
bigram <- ngram_tokenizer(n = 2)

make_collocate <- function(df) {
  df %>%
    group_by(artist, doc_id) %>%
    # N = 2のNgramを作成
    summarise(token = bigram(token, sep = "-"), .groups = "keep") %>%
    count(token) %>%
    select(token, n) %>%
    # Ngramが「単語 - 単語」として出力されるので、1列ずつに分割する
    separate(token, c("N1", "N2"), sep = "-")
}

ndf <- res %>% group_by(id) %>% group_modify(~ make_collocate(.x))
View(ndf)
```

3.3　ネットワーク分析

- Ngram（共起）の前後関係を有向グラフとして表現する手法
- igraphパッケージでグラフを作成する
- 可視化はggnetworkパッケージなどで行う
- テキストの構造や要点の把握ができる



キャンディーズ「春一番」のネットワーク

15

形態素間の共起関係を可視化する手法として、ネットワーク分析があります。形態素をノード、前後関係をエッジであらわした有向グラフで表現します。Rでは、igraphパッケージとggnetworkパッケージを組み合わせて描きます。

- 参考1: R+igraph https://sites.google.com/site/kztakemoto/r-seminar-on-igraph---supplementary-information
- 参考2: Rによるネットワークグラフ http://data-science.tokyo/R-J/R-J5-04.html
- CRAN - Package ggnetwork https://cran.r-project.org/package=ggnetwork

形態素間の有向グラフを作成するには、Ngramを分割し、それぞれの形態素と頻度からなるデータフレームを、`graph.data.frame()` 関数に与えます。特に歌詞の場合、そのままでは同じ形態素の繰り返し（ループ）が多く発生するので、`simplify()` 関数を使ってループなどを除去します。関数名が他のパッケージと被りやすいので、`igraph::simplify()` と明示します。

続いて、ネットワーク図を描画するために、ggnetworkパッケージの `ggnetwork()` 関数を使います。この際、igraph形式のオブジェクトのままでは処理ができないため、intergraphパッケージの `asNetwork()` 関数を使い、もうひとつRでグラフ構造を表現する際に使われるnetwork型のオブジェクトに変換します。また、グラフを2次元に描画する際のレイアウトについては、以下を参考にしてください。一般的には、`"fruchtermanreingold"`（Fruchterman-Reingold）アルゴリズムを使うことが多いです。

- 参考3: Chapter 2 igraph package | Introduction to Network Analysis Using R https://yunranchen.github.io/intro-net-r/igraph.html

3.3　ネットワーク分析

```
library(igraph)
library(intergraph)
library(ggnetwork)

# 共起ネットワークを描く
make_collocate_graph <- function(df) {
  filename <- paste0(str_replace_all(unique(df$artist)," ","_"), ".png")
  # 上位50個のNgramだけ描く
  df <- df %>% select(N1, N2, n) %>% arrange(desc(n)) %>% top_n(50)
  n <- graph.data.frame(df) %>% igraph::simplify(.) # ループなどを除去する
  n2 <- ggnetwork(asNetwork(n), layout = "fruchtermanreingold")
  p <- ggplot(n2, aes(x = x, y = y, xend = xend, yend = yend)) +
    geom_edges(color="black",arrow=arrow(length=unit(6,"pt"),type="closed")) +
    geom_nodes(color = "tomato", size = 6, alpha = 0.5) +
    geom_nodetext(aes(label = vertex.names), size = 6, check_overlap = TRUE) +
    theme_blank(base_family = "IPAexGothic")
  ggsave(filename, p, width = 4096, height = 4096, units = "px", bg = "white")
}

ndf %>% group_map(~ make_collocate_graph(.x))
```

16

ggnetwork() 関数でggnetworkパッケージを使って描画可能な形式に変換したオブジェクトを、ggplot2の文法で可視化します。かなり細かいカスタマイズが可能ですが、ここではシンプルな例を示しています。

有向グラフの矢印をエッジと言い、geom_edges() 関数で描画します。arrow オプションで矢印のタイプをカスタマイズできます。

有向グラフの節点をノードと言い、geom_nodes() 関数で描画します。これは、形態素そのものではなく、〇の要素です。

ノード上のテキスト（今回の場合、形態素）は geom_nodetext() 関数で描画します。aes() 関数と label 引数を組み合わせて、表示したいテキストを指定します。今回のデータ形式では、asNetwork() 関数で変換したオブジェクトの vertex.names 列に形態素が入っているので、指定します。check_overwrap = TRUE とすることで、できるだけ文字の重なりがないように調整してくれます。

このように記述することで、見栄えの良いネットワーク図が作成できます。詳細なカスタマイズについては、以下のドキュメントを参考にしてください。

- 参考: ggnetwork: Network geometries for ggplot2 https://cran.r-project.org/web/packages/ggnetwork/vignettes/ggnetwork.html

4. テキストデータの機械学習

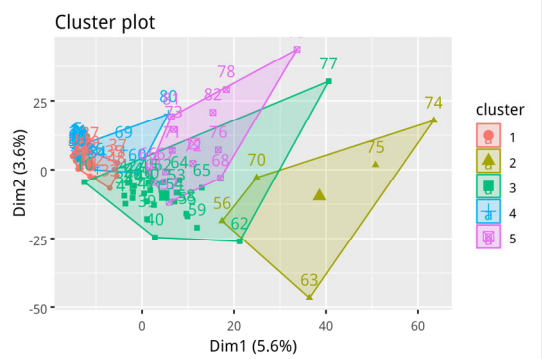
17

ここからは、テキストデータ（を加工したもの）を特徴量に用いる機械学習のアプローチを紹介します。ビジネスにおいては、企業が保有する大量のテキストデータをグループ化したり、製造現場の日報やシステム開発における各種ドキュメントからトラブルの予兆を検出するなど、さまざまな分析が行われています。

- 参考1: 新田, 森, 2022, 機械学習を用いたプロジェクト異常予測の実践, 国際P2M学会誌, 16巻 (2021-2022) 2号, p. 109-127
  https://www.jstage.jst.go.jp/article/iappmjour/16/2/16_109/_article/-char/ja/
- 参考2: 日立、AIでプロジェクトの品質不良や工期遅延リスクを早期発見
  https://ledge.ai/hitachi-solutions-project/

4.1　クラスタリング

- 文書ごとに形態素の出現頻度をカウントしたBoW (Bag of Words) を特徴量として、クラスタリングできる
- 顧客からの問い合わせや論文のグループ分けなど、さまざまな用途で使われる
- 組み込みの kmeans() 関数とfactoextraパッケージで可視化する



日本の首相の所信表明演説をクラスタリングした結果

18

はじめに教師なし学習であるクラスタリングを取り上げます。クラスタリングの考え方、手法については第9回で取り上げました。

テキストデータに対しても、形態素解析して作成したBoWから、KMeans法などを使い、「こういう形態素がよく使われがちなテキストのグループ」に分けることができます。

ビジネスにおいては、製品やサービスについて寄せられたアンケートなどをグループ分けし、改良方針やマーケティングプランを考える際などに使われます。今回は、国会（衆議院）における内閣総理大臣の所信表明演説を収集したコーパスを使い、「首相の演説をグループ化」してみました。

- 参考: 所信表明演説コーパス
  https://github.com/tetlabo/GeneralPolicySpeechOfPrimeMinisterOfJapan

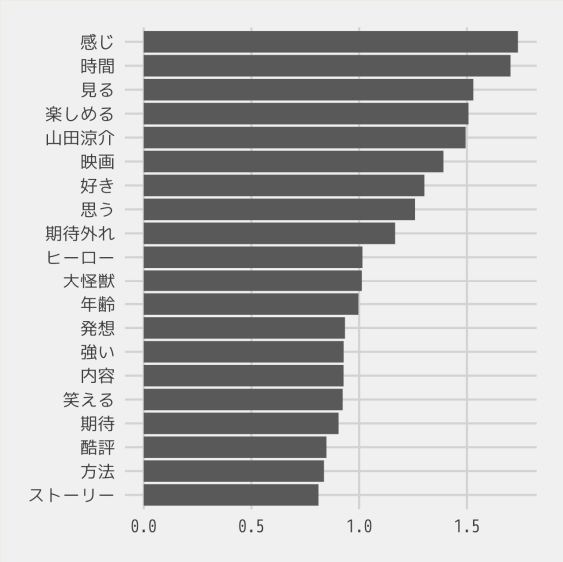
プログラムとしては、これまで同様gibasaパッケージなどを用いて形態素解析し、ストップワードを除去したうえで、縦に並んだ形態素を横方向に展開（`tidyr::spread()`）し、文書（演説）ごとに頻度をカウントしたBoWを作成します。この際、演説は1つずつテキストファイルになっているので、readtextパッケージの `readtext()` 関数を使い、1つのデータフレームにまとめて読み込んでいます。

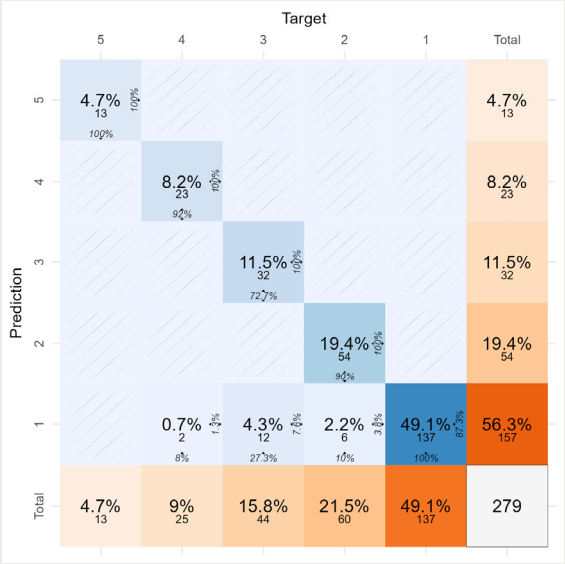
このBoWを特徴量として、組み込みの `kmeans()` 関数でクラスタリングしています。また、結果の可視化のためにfactoextraパッケージの `fviz_cluster()` 関数を使い、主成分分析で2次元に圧縮した時の各テキストの空間的配置をプロットしています。

```
# 所信表明演説コーパス
(https://github.com/tetlabo/GeneralPolicySpeechOfPrimeMinisterOfJapan) を使用
library(tidyverse)
library(readtext)
library(zipangu)
library(gibasa)

files <- paste0("./speeches/", list.files("./speeches/", pattern = ".txt"))
texts <- readtext(files)

mystopwords <- unique(c(stopwords::stopwords(language = "ja", source = "stopwords-
iso")[1:124],
  stopwords::stopwords(language = "ja", source = "marimo")[11:194],
str_conv_zenhan(0:12, to = "zenkaku"), "〇", "¥u30fb", "（拍手）"))

bow_df <- gibasa::tokenize(texts) %>%
  prettify() %>%
  filter(POS1 %in% c("名詞", "動詞")) %>%
  filter(!POS2 %in% c("接尾", "非自立")) %>%
  filter(!token %in% mystopwords) %>%
  # 原形があれば原形を選択
  mutate(token = if_else(is.na(Original), token, Original)) %>%
 # 誤って分類される記号類を除く
  filter(str_detect(token, "[¥¥p{Hiragana}|¥¥p{Katakana}|¥¥p{Han}|¥¥p{Latin}]")) %>%
  filter(!str_detect(token, "[0-9]+月[0-9]+日")) %>%
  mutate(token = str_conv_normalize(token, "nfkc")) %>%
  select(doc_id, token) %>%
  group_by(doc_id, token) %>%
  summarise(count = n()) %>%
  spread(key = token, value = count, fill = 0)

# k-means法によるクラスタリング

library(factoextra)

res <- kmeans(bow_df[, 2:ncol(bow_df)], centers = 5)

bow_df[["クラスタ番号"]] <- res[["cluster"]]

# 最適なクラスタ数の判断
fviz_nbclust(bow_df[, 2:ncol(bow_df)], kmeans, method = "wss")

# 主成分分析で2次元に圧縮した時のクラスタの分布
# 某元専修大学助教授 👽 の演説だけ極めて異質であることがわかる
fviz_cluster(res, bow_df[, 2:ncol(bow_df)])
```

4.2　回帰 / 分類

- 文書のジャンル分けなど、「正解」（目的変数）がある場合に、形態素の出現頻度を特徴量としてモデリングできる





20

回帰や分類といった教師あり学習も、テキストデータを用いて行うことができます。前記同様に、BoWを作成し、適切な目的変数（例えば人間が判定したジャンルなど）を付与すれば、「こういう形態素からなるテキストがどのジャンルにあたるか」をモデリングできます。

プログラム例はあまりに長大なので資料には載せませんが、映画作品についてのAmazonのレビューをスクレイピングし、5段階の星と形態素の関係をあらわす分類モデルを作成しています。ここではランダムフォレストでモデルを作っていますが、その際にrangerパッケージ（第9回参照）を使っています。randomForestパッケージの `randomForest()` 関数では、日本語の形態素を列名にしたデータを与えた際に、文字コードの関係かエラーになることが多かったため、「使ってみたらエラーなく処理できた」rangerパッケージを採用しました。rangerパッケージと組み合わせて、最適なハイパーパラメーターを探索するためのtuneRangerパッケージも使用しています。

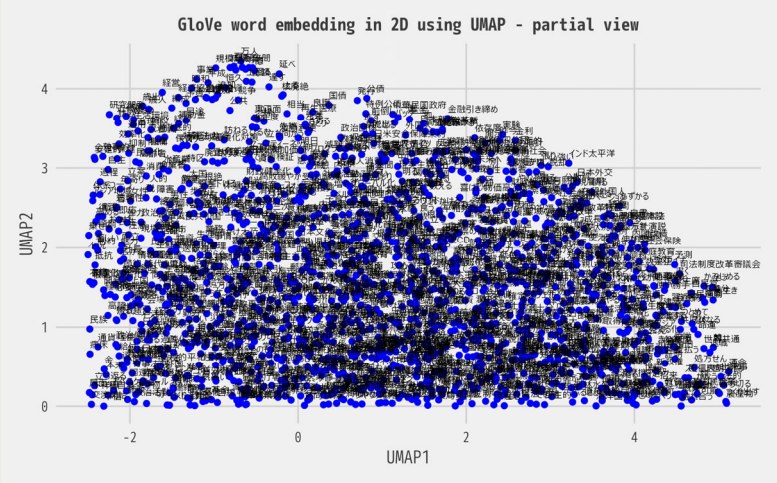
- CRAN - Package ranger https://cran.r-project.org/package=ranger
- CRAN - Package tuneRanger https://cran.r-project.org/package=tuneRanger

また、分類タスクの場合、精度を可視化して確認する方法として、混同行列（Confusion Matrix）などを使います。ggplot2で混同行列を容易に描画するためのパッケージとして、cvmsがあったので使用しています。実際には、cvmsパッケージは交差検証（第10回参照）など機械学習モデルの評価をtidyverseの文脈で行うためのさまざまな機能を提供しています。

- CRAN - Package cvms https://cran.r-project.org/package=cvms

4.3　テキストの分散表現

- 形態素を文の中での出現パターン（共起）から任意の次元のベクトルとして表現する分散表現が、2010年代から利用されるようになった
- Rではtext2vecパッケージなどが利用できる



21

最後に、より高度なテキストデータの機械学習手法として、分散表現（Word embeddingとも）を取り上げます。ある形態素の特徴を表現する方法として、テキスト中に出現するすべての形態素を行方向と列方向に並べ、該当する要素だけ1としたベクトルで表現するOne Hot Encodingがあります。しかし、この表現は1つを除きすべての要素が0となり、極めて非効率（スパース）です。また、扱うテキストが増えると膨大な次元数のベクトルとなり、計算資源を浪費します。さらに、そのように表現したところで、特に使い道がありません。

そこで、ある形態素が出現した時に別の形態素が共起する確率をニューラルネットワークで学習し、その中間層の設計次第で、形態素の特徴を任意の次元で表現できる分散表現というアイディアが生まれました。

- 参考1: 単語・句の分散表現の学習 https://www.slideshare.net/naoakiokazaki/ss-55494101
- 参考2: 絵で理解するWord2vecの仕組み - Qiita https://qiita.com/Hironsan/items/11b388575a058dc8a46a

4.3　テキストの分散表現

```
library(tidyverse)
library(text2vec)

# 作成済みの分散表現を読み込む
word_vectors <- readRDS("./word_vectors.rds")

# パリからフランスを引いて日本を足すと...
res <- word_vectors["paris", drop = FALSE] -
  word_vectors["france", , drop = FALSE] +
  word_vectors["japan", , drop = FALSE]
cos_sim <- sim2(x = word_vectors, y = res, method = "cosine", norm = "l2")

head(sort(cos_sim[,1], decreasing = TRUE), 10)
```

22

分散表現には多くのアプローチがありますが、今回はtext2vecパッケージが対応するGloVeという手法を紹介しています。これは、ニューラルネットワークによる学習の際に、形態素の共起関係を与えることで、より高精度なベクトル表現を、高速に実行できるというものです。

- 参考3: 高速かつ高性能な分散表現Gloveについて(PyTorch実装) - Qiita https://qiita.com/GushiSnow/items/e92ac2fea4f8448491ba
- 参考4: GloVe: Global Vectors for Word Representation 読んだ | Deploy on Friday https://satopirka.com/2019/01/glove/

プログラム例では、英語版Wikipediaのテキストを200次元に圧縮したベクトルを作成、使用しています。

- 参考5: Wikipedia日本語版のダンプ https://dumps.wikimedia.org/jawiki/

分散表現では、「パリからフランスを引いて日本を足すと東京になる（首都の概念）」といった演算も可能です。

```
library(tidyverse)
library(text2vec)
# 所信表明演説コーパスを使用する
library(readtext)
library(zipangu)
library(gibasa)

files <- paste0("./texts/", list.files("./texts/", pattern = ".txt"))
texts <- readtext(files)
mystopwords <- unique(c(stopwords::stopwords(language = "ja", source = "stopwords-
iso")[1:124], stopwords::stopwords(language = "ja", source = "marimo")[11:194],
str_conv_zenhan(0:12, to = "zenkaku"), "〇", "¥u30fb", "（拍手）"))

# データが巨大すぎて一括では処理できないため、小さく分割した
# ファイル単位で処理し、結果をテキストファイルに書き込んでいく
outfile <- file("jawiki_text.txt", open = "a")

for (i in 1:nrow(texts)) {
   train_df <- gibasa::tokenize(texts[i, ]) %>%
     prettify() %>%
     filter(POS1 %in% c("名詞", "動詞", "形容詞")) %>%
     filter(!POS2 %in% c("接尾", "非自立")) %>%
     filter(!token %in% mystopwords) %>%
     # 原形があれば原形を選択
     mutate(token = if_else(is.na(Original), token, Original)) %>%
    # 誤って分類される記号類を除く
    filter(str_detect(token, "[¥¥p{Hiragana}|¥¥p{Katakana}|¥¥p{Han}|¥¥p{Latin}]")) %>%
    filter(!str_detect(token, "[0-9]+月[0-9]+日")) %>%
    mutate(token = str_conv_normalize(token, "nfkc")) %>%
    collapse_tokens(., POS1 %in% c("名詞", "動詞", "形容詞"), .collapse = " ")
    writeLines(train_df$token, outfile)
}

train_text <- readLines("./jawiki_text.txt")

# Create iterator over tokens
tokens <- space_tokenizer(train_text)
# Create vocabulary. Terms will be unigrams (simple words).
it <- itoken(tokens, progressbar = TRUE)
vocab <- create_vocabulary(it)
# 頻度の少ない語彙を削除する
vocab <- prune_vocabulary(vocab, term_count_min = 3L)
# Use our filtered vocabulary
vectorizer <- vocab_vectorizer(vocab)
# 前後10語を使う
tcm <- create_tcm(it, vectorizer, skip_grams_window = 10L)
```

```r
glove <- GlobalVectors$new(rank = 200, x_max = 10) # 200次元の分散表現を獲得する
wv_main <- glove$fit_transform(tcm, n_iter = 10, convergence_tol = 0.01,
             n_threads = 20)
wv_context <- glove$components
word_vectors <- wv_main + t(wv_context)

saveRDS(word_vectors, "word_vectors.rds")


# 語彙空間の可視化
# https://medium.com/broadhorizon-cmotions/nlp-with-r-part-2-training-word-
embedding-models-and-visualize-results-ae444043e234
library(uwot)
glove_umap <- umap(word_vectors, n_components = 2, metric = "cosine", n_neighbors =
50, min_dist = 0.1, spread = 2, n_threads = 20, verbose = TRUE)

# Put results in a dataframe for ggplot, starting with Word2Vec
df_glove_umap <- as.data.frame(glove_umap, stringsAsFactors = FALSE)

# Add the labels of the words to the dataframe
df_glove_umap$word <- rownames(word_vectors)
colnames(df_glove_umap) <- c("UMAP1", "UMAP2", "word")
df_glove_umap$technique <- "GloVe"
str(df_glove_umap)

ggplot(df_glove_umap) +
      geom_point(aes(x = UMAP1, y = UMAP2), colour = "blue", size = 2) +
      geom_text(aes(UMAP1, UMAP2, label = word), size = 2.5, vjust=-1, hjust=0) +
      labs(title = "GloVe word embedding in 2D using UMAP - partial view") +
      theme(plot.title = element_text(hjust = .5, size = 14))


ggplot(df_glove_umap[df_glove_umap$UMAP1 >= -2.5 & df_glove_umap$UMAP2 >= -2.5 &
df_glove_umap$UMAP2 >= 0,]) +
      geom_point(aes(x = UMAP1, y = UMAP2), colour = "blue", size = 2) +
      geom_text(aes(UMAP1, UMAP2, label = word), size = 2.5, vjust=-1, hjust=0) +
      labs(title = "GloVe word embedding in 2D using UMAP - partial view") +
      theme(plot.title = element_text(hjust = .5, size = 12))
```

5. まとめ

5.1　今日の内容

- さまざまな分析手法
  - ワードクラウド
  - 共起分析
- テキストデータの機械学習
  - クラスタリング
  - 回帰 / 分類
  - word2vec / doc2vec / …

5.2　次回までの課題

- Posit Cloudプロジェクト内の `14_text_mining_02_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: 2024-01-15 23:59

27