行動計量学会チュートリアル資料

石田基広

2019 09 03 (資料 08/27 版)

目次

1 チュートリアル概要

最近 R 界隈では tidyverse という分析フレームワークが浸透しています。tidyverse では、データの前処理と分析方法を統一的な原理にもとづき一般化しようとしていると言えます。本チュートリアルでは、この原理にならって、改めて R におけるデータ処理と分析の手順を確認します。同時に、機械学習周りの手法、テキスト分析の方法と事例を紹介します。

残念ながら、時間の関係上、チュートリアルでは、R における tidyverse の原理、機械学習、テキスト処理それぞれについて、簡単な概要だけしか示すことができないと思います。またチュートリアルの時間中に、受講者の方々自身に操作いただく時間を設ける余裕もありません。基本的に、教室に投射するスライドにもとづいて話を進めていきます。チュートリアルでは、以下のページで紹介する分析を実行するのに必要な基礎知識を解説したいと考えています。

https://ishidamotohiro.github.io/tutorial2019/NLP.html
https://ishidamotohiro.github.io/tutorial2019/NLP2.html

1.1 チュートリアル資料

この PDF ファイルとは別に、本チュートリアルで利用したスクリプトファイル（コードを記載した拡張子 R のファイル）とデータを以下に公開します。スクリプト類は、今後修正される可能性があります。なお、スクリプトファイル類は現在、パスワード付きで圧縮しています。大会実行委員会より指示のあるパスワードで解凍してください。

https://github.com/IshidaMotohiro/tutorial2019

ファイル名を個々にクリックしてダウンロードできますが、サイト右端の緑色の download ボタンを押すと、全部をまとめて圧縮ファイルとしてダウンロードできます。Excel ファイルを除いて、残りはすべて UTF-8 形式で読みこまなければエラーになります。Windows の RStudio では、File メニューの「Reopen with Encoding」で CP932 を選ぶと文字化けが直りますので UTF-8 ファイルのままお使いください。チュートリアルで投射するスライドも、このサイトに公開予定です。

https://ishidamotohiro.github.io/tutorial2019/Osaka_part1.html
https://ishidamotohiro.github.io/tutorial2019/Osaka_part2.html

なお、本チュートリアルの一部は、年末に刊行予定の拙著『R によるテキストマイニング入門 -応用編-』森

北出版に掲載される予定です。

2　R におけるデータ操作

2.1　オブジェクト

データなど操作の対象、変数ともいう。統計解析アプリケーションとしての R では、操作対象としてのデータ（Excel でいうワークシート）が重要なオブジェクトである。

以下は、3 つの日本語テキストそれぞれに形態素解析を適用し、単語文書行列（実際はデータフレーム）としたデータオブジェクトを、サイトから、手もとのパソコンにダウンロードして読み込んでいる。

```
# データの準備
download.file("https://github.com/IshidaMotohiro/tutorial2019/blob/master/df.Rdata?raw=true",
        destfile= "df.Rdata")
load("df.Rdata")# オブジェクトを読み込む
# オブジェクトを表示する
df
```

2.2　関数

オブジェクトを操作する機能、メソッドともいう。

オブジェクトに関数を適用する例：データフレーム df に関数 head() を適用。

```
head(df)
```

TERM	POS1	POS2	文書 1	文書 2	文書 3
、	記号	読点	2	5	3
。	記号	句点	2	2	2
ある	動詞	自立	0	0	2
いく	動詞	非自立	0	1	0
いる	動詞	非自立	1	1	0
う	助動詞	*	0	1	0

2.3　データの型（構造）

2.3.1　ベクトル

複数の数値や文字をまとめたもの。

例：１から１００までの整数、A から Z までの２６文字

```
LETTERS
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

2.3.2 データフレーム

複数のベクトルを列単位で束ねたもの。Excel のワークシートに相当。よってデータ分析ではデータフレームの操作に慣れる必要がある。

```
# 先程ダウンロードしたオブジェクトの「クラス」を確認
class(df)
```

[1] "data.frame"

2.4 基本的なデータフレーム操作

特定の列を指定するには **オブジェクト名＄列名** という命令を使う。（以下、出力は省略する。）

2.4.1 列の操作：＄演算子

```
df $ 文書1 # ある列を表示（抽出）
```

2.4.2 条件抽出：添字演算子（カギ括弧）

カギ括弧内部をカンマで前と後ろに区切り、前が行の指定、後ろが列の指定

```
df [ 1:3, 1:2 ]        #1~3行のレコードの1, 2列を表示（抽出）
```

2.4.3 条件抽出：検索 ＝＝や＞といった演算子

```
#TERM列の要素が「理工」と一致することを＝＝演算子で表現
df [ df $ TERM == "理工", ]
# 行（レコード）の3つの列の合計値が「5 を超える」
df [ rowSums( df[ , c("文書1", "文書2", "文書3")] ) > 5 , ]
```

2.4.4 列の追加変更：＜ーの左に「オブジェクト名＄新規列名」と指定

```
df$合計 <- rowSums ( df[, c("文書1", "文書2", "文書3")] ) # 合計という列を追加
df$標準化 <- scale ( df$合計) # 他の列を加工した値を新しく列に
```

2.4.5 列の要約：mean などの要約関数を適用する

```
mean(df $ 合計)  #「合計」列の平均値
```

3 tidy なデータ操作

Hadley Wickham https://vita.had.co.nz/papers/tidy-data.pdf

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

```
install.packages("tidyverse")
```

3.1 なぜ tidyverse なのか

- R の基本機能ではコードのカッコが多くなって分かりにくい（処理そのものには影響はないが）。

例：df データで「出版」という単語の出現回数の最大値を表示する。

```
max(df[df$TERM == "出版", c("文書1", "文書2", "文書3")])
```

```
[1] 2
```

tidyverse のポリシーでは以下のようになる。

```
tidy_df %>% filter(TERM == "出版") %>% summarise(MAX = max(FREQ))
```

```
  MAX
1   2
```

- 基本機能に沿った操作では目的ごとに一時ファイル (temp) を作成することが多くなる。

データフレームの変数に変換を施してから解析を適用する場合、念のため元データの複製を作って、変更を加えることがあるが、これを繰り返していると、分析対象とすべきオブジェクトを取り違えることがある。

```
# オリジナルのコピー
copied <- cars
# コピーの変数を対数化
copied$dist <- log(copied$dist)
# 元データを指定して分析してしまう
copied_lm <- lm(dist ~ speed, data = cars)
```

tidyverse のポリシーでは以下のようになる。

```
cars %>% mutate(dist = log(dist)) %>% lm(dist ~ speed, data = .)
```

3.2 データの形式

3.2.1 messy data

行列形式に近く、目視に便利。また機械学習系のパッケージの関数は、この形式を入力とする。

```
head(df)
```

TERM	POS1	POS2	文書 1	文書 2	文書 3
、	記号	読点	2	5	3
。	記号	句点	2	2	2
ある	動詞	自立	0	0	2
いく	動詞	非自立	0	1	0
いる	動詞	非自立	1	1	0
う	助動詞	*	0	1	0

3.2.2 tidy data

tidyverse 流のデータ処理に適切な形式。グラフィックスパッケージ ggplot2 で想定されている形式。また、ここでは通常のデータフレームを、その拡張形式である tibble に変換するが、これは必須ではない。

```
library(tidyverse)
tidy_df <- df %>% gather(key = Doc, value = FREQ, 文書1, 文書2, 文書3)
tidy_tb <- as_tibble(tidy_df) # この操作は必須ではない
tidy_tb
```

TERM	POS1	POS2	Doc	FREQ
、	記号	読点	文書 1	2
。	記号	句点	文書 1	2
ある	動詞	自立	文書 1	0
いく	動詞	非自立	文書 1	0
いる	動詞	非自立	文書 1	1
う	助動詞	*	文書 1	0

3.3 tidy data に対する操作

パイプ演算子 %＞%を中心に操作し、基本的に添字（カギ括弧）は使われない。

なお、念のため messy な形式のデータフレーム（ RMeCab の docDF 関数のデフォルト出力である）df に対する処理も併記する。

3.3.1 列を抽出する select()

- tidy な形式の場合に「文書 1」のレコードのみ取り出す方法。

Doc 列が「文書 1 」であるレコードを抽出する

```
tidy_tb %>% filter( Doc == "文書1" )
```

TERM	POS1	POS2	Doc	FREQ
、	記号	読点	文書 1	2
。	記号	句点	文書 1	2
ある	動詞	自立	文書 1	0
いく	動詞	非自立	文書 1	0
音楽	名詞	一般	文書 1	0
高い	形容詞	自立	文書 1	0

- messy な形式の場合は「列名」として指定する

```
df %>% select( 文書1 )
```

3.3.2 列の検索 filter()

単語が「理工」であるレコード、あるいは品詞が「名詞ないし形容詞ないし動詞」であるレコードを抽出する方法。

- tidy データの場合

```
tidy_tb  %>% filter( TERM == "理工" )
```

TERM	POS1	POS2	Doc	FREQ
理工	名詞	一般	文書 1	1
理工	名詞	一般	文書 2	0
理工	名詞	一般	文書 3	2

```
tidy_tb  %>% filter( POS1 %in% c("名詞", "形容詞", "動詞") )
```

- messy な場合も同じ

```
df %>% filter( TERM == "理工" )
df %>% filter( POS1 %in% c("名詞", "形容詞", "動詞") )
```

3.3.3 列の追加 mutate

tyd_tb データには、記号の「。」について、3 文書ごとに頻度が集計されている。いま文書 3 つを統合した頻度を知りたいとする。

```
tidy_tb %>% filter(TERM == "。")
```

TERM	POS1	POS2	Doc	FREQ
。	記号	句点	文書 1	2
。	記号	句点	文書 2	2
。	記号	句点	文書 3	2

- tidy データの場合

文書の違いを無視して、単語（TERM 列）でまとめて合計する。

```
tidy_tb  %>% group_by(TERM, POS1, POS2) %>% summarize(FREQ = sum(FREQ))
```

TERM	POS1	POS2	FREQ
、	記号	読点	10
？	記号	一般	1
。	記号	句点	6
方々	名詞	一般	2
役割	名詞	一般	1
理工	名詞	一般	3

- messy な場合

新たに「合計」という列を追加する。ただし、横（行）方向に合計をとる必要があるので rowSums() を使う。また、要素が数値である列だけが対象となる。select_if() と is.numeric() を併用することで、「数値の列を選択する」ことを表現している。(rowwise() で横方向の合計を指定しても実現できるが、rowwise() の利用には慎重であるほうが良いと思われる)。

```
df %>% mutate(合計 = rowSums(select_if(., is.numeric)))%>% head()
# df %>% rowwise() %>% mutate(合計 = sum(文書1, 文書2, 文書3)) %>% head()
```

TERM	POS1	POS2	文書 1	文書 2	文書 3	合計
、	記号	読点	2	5	3	10
。	記号	句点	2	2	2	6
ある	動詞	自立	0	0	2	2
いく	動詞	非自立	0	1	0	1
いる	動詞	非自立	1	1	0	2
う	助動詞	*	0	1	0	1

3.3.4 列の要約 group_by() + summarise()

文書「列」ごとに総語数を数える

- tidy データの場合

```
tidy_tb %>% group_by(Doc) %>% summarise(FREQ = sum(FREQ))
```

Doc	FREQ
文書 1	46
文書 2	67
文書 3	83

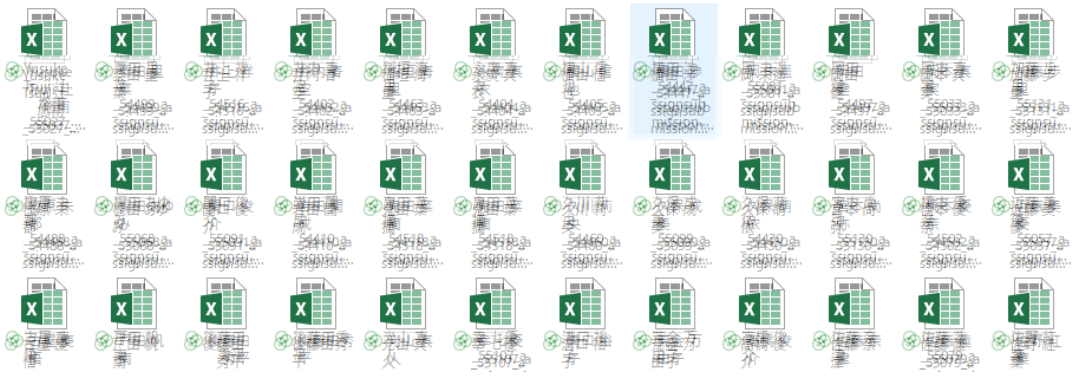
- messy な場合

```
# 数値が記録された列を指定して合計を求める
df %>% summarize_if(is.numeric, sum)  %>% head()
```

文書 1	文書 2	文書 3
46	67	83

4　退屈なことは R にやらせよう

数百の Excel ファイルを探索処理する事例：



課題をして提出された Excel ファイルごとに、特定のセルに入力された式を抽出して採点し、提出者ごとに採点した結果を Excel ファイルで出力する。

配布ファイルの seiseki_Demo.R を参照

5　ggplot2

ggplot2 の説明に gapminder パッケージを利用する。gapminder については https://www.gapminder.org/ を参照。

```
install.pacakges("gapminder")
```

5.1　作図の手順

ggplot2 パッケージでの作図の基本は、データ、その変数の役割、グラフの種類を足し算で指定していくことにある。

- データを ggplot() に渡す

```
df %>% ggplot()
```

- 変数（列）の役割を指定 aes()

```
+ aes(x = Col1, y = Col2, group = Col3, color = Col4)
```

- グラフの種類を指定: geom

```
+ geom_point, geom_bar, geom_histogram, geom_line, geom_boxplot
```

具体的な作成方法は Osaka_part2.R を参照

6 探索的分析

レコードがいくつかのグループに分かれている場合、そのそれぞれについて個別に要約したい場合、あるいは仮に分析しておきたい場合がある。

また、回帰分析の結果は、通常はメッセージとして表示されるが、この出力自体を整形しデータフレームとして利用したい。

6.1 回帰分析の結果を表形式に整える

gapminder パッケージから、試行のためデータを日本、中国、韓国にしぼる。

```
install.packages("gapminder")
```

```
library(gapminder)
gapminder <- gapminder %>% filter(country %in% c("Japan", "China", "Korea, Rep."))
```

6.2 国別に回帰

日本についてのみ、「平均余命」を「一人あたり GDP」で回帰。filter() で絞り込んだデータを、回帰を実行する lm() に渡す。

```
gap_lm <- gapminder %>% filter(country == "Japan") %>% lm(data=., lifeExp ~ gdpPercap)
summary(gap_lm)
```

```
Call:
lm(formula = lifeExp ~ gdpPercap, data = .)

Residuals:
   Min     1Q Median     3Q    Max
-2.674 -0.985  0.409  1.019  1.564

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.37e+01   8.32e-01    76.6  3.5e-15 ***
gdpPercap   6.28e-04   4.11e-05    15.3  2.9e-08 ***
---
codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 1.38 on 10 degrees of freedom
Multiple R-squared: 0.959,Adjusted R-squared: 0.955
F-statistic: 233 on 1 and 10 DF, p-value: 2.95e-08

6.3 出力を表に変換

出力はメッセージなので、これを表形式にするため broom パッケージの tidy() で整形する。

```
library(broom)
gap_lm %>% tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	63.6846238724022	0.831637931641735	76.5773438768991	3.52132441939716 (-15)
gdpPercap	0.000627704039531759	4.11123598810475 (-05)	15.2680128639642	2.94880714616723 (-08)

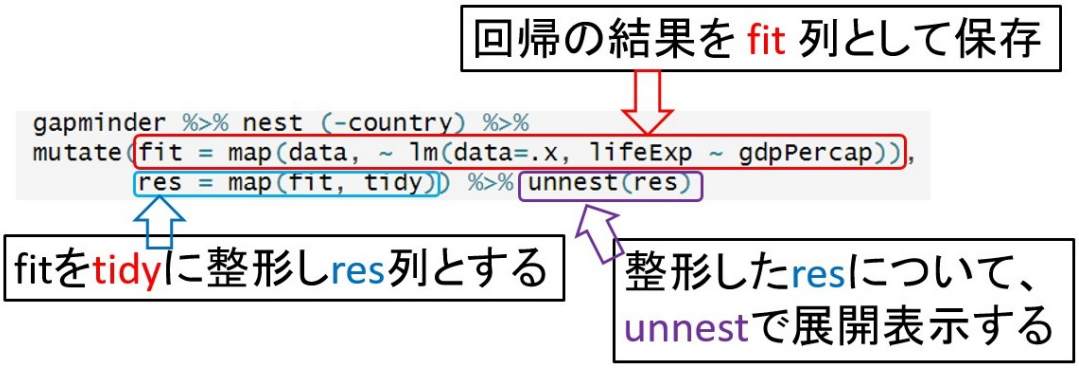
6.4 各水準について一度で回帰分析を実行

日本に加え、中国、韓国についても回帰分析を実行する。

unnest() で国ごとにデータ分けて分析を実行し、最後に unnest() で結果を結合する。

```
gapminder %>% nest (-country) %>%
        mutate(fit = map(data, ~ lm(data=., lifeExp ~ gdpPercap)),
        res = map(fit, tidy)) %>% unnest(res)
```

country	term	estimate	std.error	statistic	p.value
China	(Intercept)	53.9948550661662	3.35135731414049	16.1113393783301	1.755996862126 (-08
China	gdpPercap	0.00523432416911913	0.00168900469653678	3.0990583861915	0.011268716571204
Japan	(Intercept)	63.6846238724022	0.831637931641735	76.5773438768991	3.52132441939716 (-
Japan	gdpPercap	0.000627704039531759	4.11123598810475 (-05)	15.2680128639642	2.94880714616723 (-
Korea, Rep.	(Intercept)	55.2967086685328	1.9669400104735	28.113063120426	7.53337521123418 (-
Korea, Rep.	gdpPercap	0.00118095597110944	0.000178172810297752	6.62814920602026	5.8674767871742 (-0



6.4.1　purrr::map のチルダについて

map ではリストの要素ごとに関数を適用する。

ここで map(data, ~lm(data=.x, lifeExp ~ gdpPercap)) にある最初のチルダは「無名関数」を作成するために使われる。すなわち lm(data=., lifeExp ~ gdpPercap) は function(x){lm(data=x, lifeExp ~ gdpPercap)} と展開される。これにより ~data=. のドットには、この無名関数を実行するたびにデータ（リストの要素）を割り当ててもらえる。ところが、ここで冒頭のチルダなしに map(data, lm(data=., lifeExp ~ gdpPercap)) と書いてしまうと、引数の data=. のドットにリストの要素が割り当られなくなってしまうのである。

7　機械学習

機械学習では、一般に以下の手順がとられる。

- データを訓練用（および検証用）とテスト用に分ける
  - データを標準化（主成分化）する
- 分析手法に固有のハイパーパラメータの指定
- 訓練データにモデルを当てはめ検証する
- 推定されたモデルにテストデータを適用

7.1　テキストデータに対する機械学習

Twitter データの内容を予測するモデルを作成してみる。ここでは、奈良先端科学技術大学院大学鈴木優氏のデータを利用 (http://bigdata.naist.jp/~ysuzuki/data/twitter/ ;Tweets そのものは利用者が自身で取得する)。このデータは、個々のツィートの内容が肯定的か否定的かを、人手による評価が含まれている。ここから、iPhone についてのツィートデータを抽出して利用する。その上で、データを学習し、出現する単語から、その内容が肯定的か否定的かの二値判定を行う。

- 肯定的か否定的か ~ Word1 + Word2 + ... + Word9999 + Word10000

このデータのテキスト部分に形態素解析を実行し、行に status_id を、また列に単語を取り、成分がそれぞれの語の頻度であるデータフレームを用意する。さらに列には肯定的か否定的かを表す Y 変数が加えら

れている。（データの生成方法は、説明が煩雑になるため、省略しています。興味のある方は近著『R によるテキストマイニング入門 応用編』森北出版をご参照ください。）データフレームは以下のようにしてダウンロードできる。なお、チュートリアルで利用するデータでは、データフレームの列数を絞り込んでいる。また stopwords とよばれる助詞などの高頻度語は、内容の判別に貢献しないと思われるので削除してあり、tf-idf などによる頻度の調整も行っていない。

```
download.file("https://github.com/IshidaMotohiro/tutorial2019/blob/master/iPhone.Rdata?raw=true",
              destfile= "iPhone.Rdata")
load("iPhone.Rdata")# iPhoneデータフレームがロードされる
```

データ行数（ツィートの数）と列数（単語の数）を確認する。

```
iPhone %>% dim()
```

[1] 5344 1437

肯定的 (TRUE) の割合を確認する。

```
iPhone %>% select(Y) %>% table()
```

.
FALSE TRUE
 3503 1841

余談だが、例えば、データにおいて「肯定」の回答が占める割合が極端に少ないような場合、つまりデータがアンバランスな場合は、割合を調整した分析の適用を検討すべきである。詳細は ROSE パッケージ https://cran.r-project.org/web/packages/ROSE/index.html などの解説を参照されたい。

7.2 判別の手法

機械学習の文脈では、二値（あるいは多値）のクラスを判別する手法としてよく知られているのは以下であろう。カッコ内は、よく利用されているパッケージである。

- ロジスティック回帰 (glmnet)
- サポートベクターマシン (kernlab)
- k-近傍法 (class)
- 決定木 (rpart)
- ナイーブベイズ (e1071)
- ランダムフォレスト (randomForest)
- バギング (adabag)
- XGboost (xgboost)

チュートリアルでは、解釈のしやすいロジスティック回帰を取り上げる。また、実行方法を比較するという意味で、サポートベクターマシンについても取り上げる。

7.3　ロジスティック回帰

ロジスティック回帰については R に最初から入っている stats::glm() が利用可能である。以下のように glm() で family に ”binomial” を指定するだけで実行できる（ただし、すべてのデータを使っているので、結果が返るまでに数分がかかる）。

```
iPhone_glm <- glm(Y ~ ., data= iPhone, family = binomial())
# summary(iPhone_glm)
```

この場合、約 1,500 個ある単語について係数が推定されることになる。

一般的な回帰の手順に従えば、ここで各説明変数、すなわち出現単語のすべてについて P 値を確認し、有効な変数を探ることになるが、目視で確認するのは現実的ではない。そこで、以下のように broom パッケージの整形関数である tidy() を併用して、P 値が 0.05 未満の説明変数を抽出する。

```
library(broom)
iPhone_glm %>% tidy() %>% filter(p.value < .01) %>% arrange(p.value) %>%
  mutate_if(is.numeric, round,3)#
```

term	estimate	std.error	statistic	p.value
(Intercept)	-0.677	0.057	-11.793	0
画質	3.61	0.471	7.661	0
ん	-0.799	0.116	-6.873	0
認証	1.762	0.303	5.822	0
デカ	-1.275	0.248	-5.14	0
s	0.358	0.074	4.824	0
最高	2.609	0.617	4.227	0
手	-0.844	0.225	-3.756	0
感じ	1.257	0.365	3.441	0.001
性能	3.726	1.141	3.267	0.001
ピンク	1.688	0.525	3.217	0.001
音質	3.285	1.049	3.133	0.002
反応	-1.134	0.413	-2.75	0.006
バッテリー	-0.542	0.198	-2.738	0.006
S	0.345	0.13	2.658	0.008
回線	1.564	0.592	2.641	0.008

「画質」の係数がプラスであるのに対して、「デカ（さ）」がマイナスであることなどが分かる。

7.3.1　予測と混合行列

モデルによる肯定的か否定的かの予測と、実際の評価の一致度は以下のように確認できる。predict() で予測を行い、混合行列で予測の精度を見る。ここでは glm() に与えたデータを予測している。

```
pred <- predict(iPhone_glm, type="response")
# 予測値は肯定的となる確率値なので丸めて 1 か 0 にする
pred <- round(pred)
```

```
table(予測 = pred, 実測=iPhone$Y)
```

FALSE	TRUE
3339	897
164	944

予測の精度を測る方法については後述するが、ここでは正確度（Accuracy) を出す。精度は、要するに全体の予測のうち正解の割合である。全データを使って学習したモデルでは、予測精度が 0.80 になっている。

```
(3339 + 944) / ( 3339 + 944 + 164 + 897)
```

[1] 0.801

7.4 モデルの汎化性能

ただし、新しいデータへの予測を重視する場合、モデルは投入されたデータに特化しており（バリアンスが高い)、新しいデータを予測する性能が劣る（バイアスが高い)。過適合 (overfitting) といわれるが、これを避けるのに利用される手順は主に 2 つある。

- データを訓練（モデル構築用）データと、テスト（モデルの精度を調べる）データに分ける
- モデルの（ハイパー）パラメータを調整する

7.5 データの分割

データを訓練用とテスト用に分ける。一番単純な方法はデータ数をもとに、その半分、あるいは 7 割の整数をランダムに取り出し、これを添え字として分けることである。

```
N <- iPhone %>% NROW()
index <- sample(N, N * 0.5)
index %>% NROW()
train_data <- iPhone[index, ]
test_data <- iPhone[-index, ]
```

ただし、この単純な方法では目的変数である「肯定・否定」の割合が、訓練データとテストデータで極端に異なっている可能性がある。

これに対処する簡単な方法は caret パッケージの createDataPartition() を利用することである。

```
library(caret)
set.seed(123) # 再現性のため乱数の種を設定
index <- createDataPartition(y = iPhone$Y, p = 0.7, list = FALSE)
```

```
training <- iPhone[ index,  ]
training$Y %>% table()
```

.	Freq
FALSE	2453
TRUE	1289

同様に訓練データを用意する。

```
testing  <- iPhone[-index, ]
testing$Y  %>% table()
```

.	Freq
FALSE	1050
TRUE	552

いずれも元データにおける「肯定・否定」と同じ割合で抽出されている。

7.6 モデルのパラメータ

ロジスティック回帰で過適合を防ぐ方法として、回帰モデルに罰則項をいれた方法がある。通常の回帰分析では目的変数 y を、例えば $x_1 + x_2 + x_3$ の線型結合で説明しようとする（ただし、実際にはこれに切片と誤差が加わる）。この際、$x_1 + x_2 + x_3$ の係数に制約を加えるのが Lasso, Ridge, Elastic Net として知られる手法である。

$$min\left[\frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - x_i^T\beta) + \lambda P_\alpha(\beta)\right] \tag{1}$$

$$P_\alpha(\beta) = \sum_{j=1}^{p}\left[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right] \tag{2}$$

7.6.1 Lasso 回帰

Lasso 回帰は上の式で α を 1 とした場合にあたり、係数の絶対値の和をペナルティとした推定方法である。これを L1 ノルム（正則化）と呼ぶ。また λ は正則化パラメータと呼ばれ、大きいほどペナルティ（罰則項）の影響が強くなり、この結果、変数のうちいくつかのパラメータは 0 になる。すなわち、有効な説明変数の数が少なくなる。

7.6.2 Ridge 回帰

一方、Ridge 回帰は上の式で α を 0 とした場合で、係数の 2 乗和をペナルティとした推定方法である。これを L2 ノルムと呼ぶ。Ridge 回帰では説明変数の回帰係数を 0 の方向に縮小するが、完全に 0 にはしない。

7.6.3 Elastic Net 回帰

Elastic Net 回帰では、回帰係数の 2 乗和と絶対値の和を両方を取り込み、パラメータ α によって、その割合を調整する。

7.7 glmnet パッケージ

正則化を取り込んだ回帰分析を実行できるパッケージに glmnet がある。glmnet パッケージで Lasso 回帰を行う場合は、関数の引数としてパラメータ alpha に 1 を指定する。Ridge 回帰の場合は引数 alpha に 0 を指定する。またパラメータ lambda を指定する必要があるが、これは交差法で推定する。Elastic Net 回帰の場合は、この alpha についても推定が行われることになる。glmnet では cv.glmnet() を使うと交差法によって最適なモデルを推定する。cv.glmnet() ではデフォルトでデータを 10 分割し、そのうち 9 つでモデルを作成し、残り 1 つで結果を検証することを繰り返す。

実際に試してみよう。その前にデータを glmnet に適切な形式に変換する。glmnet では目的変数と説明変数を別に指定しなければならない（いわゆるモデル式 Y ~ X はサポートされていないが、別に glmnetUtils パッケージを導入するとモデル式も使える）。説明変数を行列に、また目的変数をベクトルに変えて用意するのがシンプルである。

```
# 訓練データを整形して行列にする
train_X <- training %>% select(-Y) %>% as.matrix(dimnames = list(NULL, colnames(.))) #列名（単語）を残す
# 目的変数のベクトル
train_Y <- training$Y
# テストデータを整形して行列にする
test_X <- testing %>% select(-Y) %>% as.matrix(dimnames = list(NULL, colnames(.))) #列名（単語）を残す
# 目的変数のベクトル
test_Y <- testing$Y
```

訓練データに Lasso 回帰を適用する。alpha パラメータに 1 を指定することで L1 正則化が指定される。alpha を指定しない場合は Elastic 回帰が実行され alpha も推定される。

なお、パラメータ探索（複数の候補を実行して比較する）によりパソコンの負荷が高まるので、並列化を行っておくほうが良い。以下では、利用しているパソコンで利用可能なコアから 2 を引いた数を指定している。コアを全部投入すると、OS そのものが反応しなくなることがあるので注意されたい。

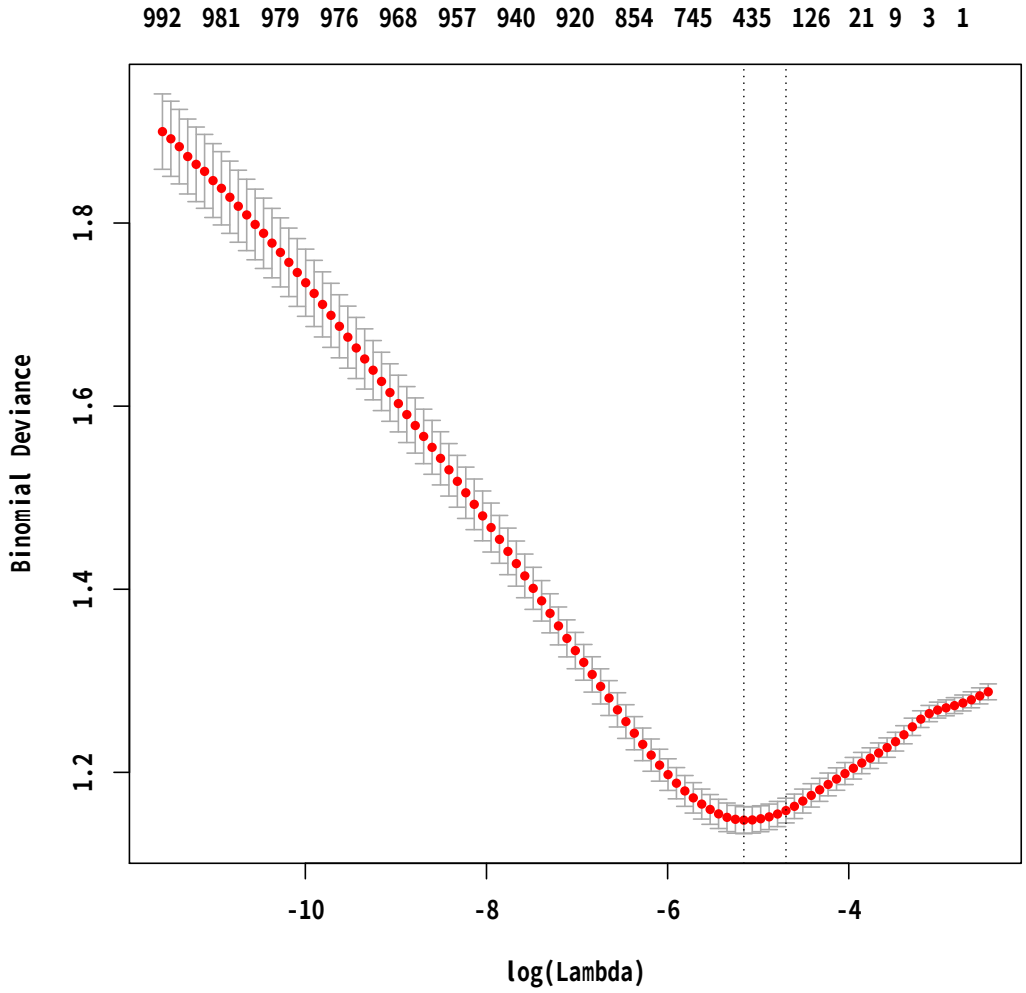
```
# install.packages("doParallel")
library(doParallel)
core <- parallel::detectCores() - 2
cl <- makePSOCKcluster(core)
registerDoParallel(cl)

#  並列演算の必要がなくなれば以下を実行
## stopCluster(cl)
```

```
library(glmnet)
lasso <- cv.glmnet(x = train_X, y = train_Y, alpha = 1, family= "binomial", parallel = TRUE)
```

分析結果が保存されたオブジェクトからは次のようなグラフが得られる。

Listing 1　パラメータと説明変数の数

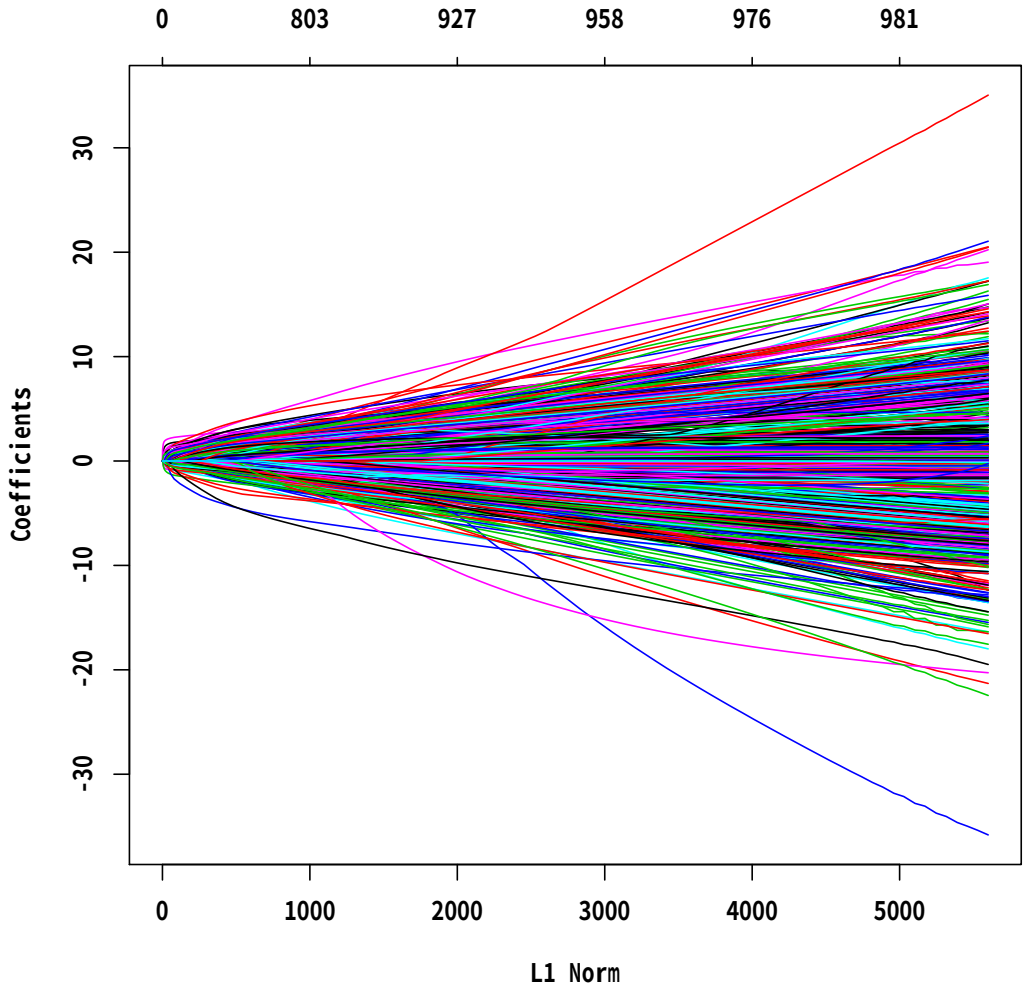
```
plot(lasso)
```



モデルのパラメータである λ（の対数）の値に対応するデビアンスの大きさが表示されている。

Listing 2　lambda の対数と、デビアンスの大きさ

```
plot(lasso$glmnet.fit)
```



こちらは、正則化の強さ（右にいくほど弱くなる）と、係数が 0 近くに調整される説明変数の数が示されている。

推定された最良のモデルでのパラメータを確認する。

```
# 推定されたモデルパラメータ
lasso$lambda.min
# 各説明変数（単語）の係数
x <- coef(lasso, s = lasso$lambda.min)
x %>% as.matrix() %>% tail()
```

```
[1] 0.00575
          1
貰い      0
問い合わせ 0
也        0
```

野外　　0
厄年　　0
薬指　　0

出力でドットは係数が 0 に調整されていることを表す。

説明変数のうち、係数が 0 に調整されなかった言葉を確認してみよう。

```
library(broom)
x %>% as.matrix() %>% tidy() %>% filter(X1 > 0) %>% tail()
```

```
# A tibble: 6 x 2
  .rownames     X1
  <chr>      <dbl>
1 毎年       1.39
2 夢         1.47
3 面積       0.0438
4 網         1.22
5 目的       1.64
6 目標       0.893
 警告メッセージ:
'tidy.matrix' is deprecated.
See help("Deprecated")
```

7.7.1　予測と混合行列

訓練データに適用した最良のモデルを使ってテストデータでの予測精度を確認しよう。

```
lasso_preds <- predict(lasso, newx = test_X, type = "class")
```

caret パッケージの confusionMatrix() で予測の精度を確認する。

```
confusionMatrix(table(lasso_preds, test_Y), positive = "TRUE")
```

```
Confusion Matrix and Statistics

           test_Y
lasso_preds FALSE TRUE
      FALSE  1006  434
      TRUE     44  118

               Accuracy : 0.702
                 95% CI : (0.679, 0.724)
    No Information Rate : 0.655
    P-Value [Acc > NIR] : 4.69e-05
```

Kappa : 0.206

Mcnemar's Test P-Value : < 2e-16

Sensitivity : 0.2138
Specificity : 0.9581
Pos Pred Value : 0.7284
Neg Pred Value : 0.6986
Prevalence : 0.3446
Detection Rate : 0.0737
Detection Prevalence : 0.1011
Balanced Accuracy : 0.5859

'Positive' Class : TRUE

分割したデータセットのみを学習したモデルを使った精度は 0.70 となった（ここで試行したデータセットでは、説明変数、つまり単語をかなり削っているためもあり、精度はよくない。

7.7.2 ROC/AUC

ここでロジスティック回帰では肯定的になる確率が内部で計算されており、その値が閾値を超えると肯定的（すなわち 1）と予測される。デフォルトで閾値は 0.5 であるが、この閾値を 0 から 1 まで調整した場合の偽陽性率と真陽性率をグラフで表すことで、予測の精度を確認する。

ROC は、横軸に [1-特異度] (偽陽性率) を、また縦軸に [感度]（真陽性率）を取ったグラフである。

また、曲線下の面積を AUC という。AUC の面積が面積が大きい場合、陰性と陽性のそれぞれが正しく判断される度合いが強い。

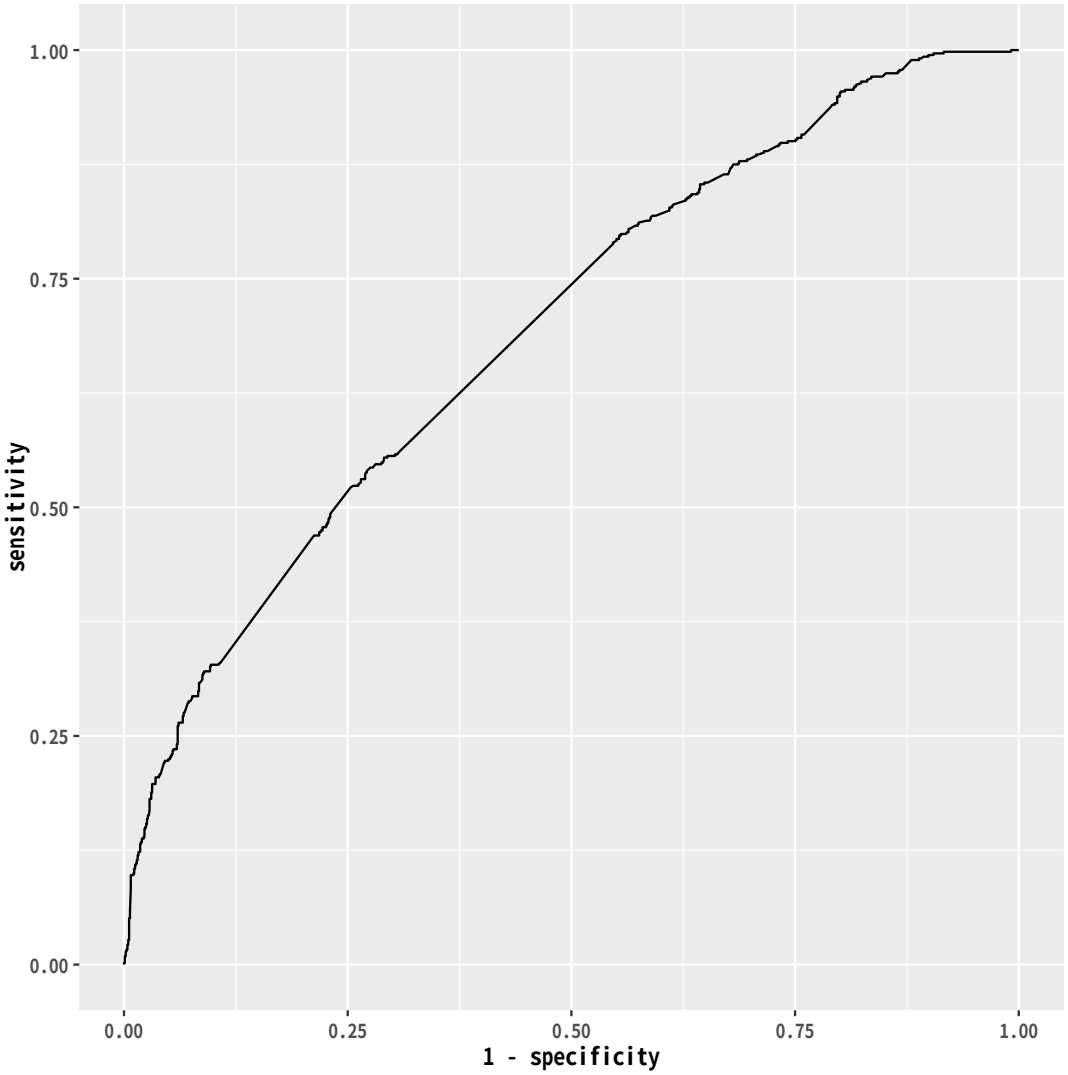
```
library(pROC)
lasso_response <- predict(lasso, s = "lambda.min", newx = test_X, type = "response")
lasso_roc <- roc(test_Y, as.numeric(lasso_response))
lasso_roc
```

Call:
roc.default(response = test_Y, predictor = as.numeric(lasso_response))

Data: as.numeric(lasso_response) in 1050 controls (test_Y FALSE) < 552 cases (test_Y TRUE).
Area under the curve: 0.693

Listing 3 ROC

```
ggroc(lasso_roc, legacy.axes = TRUE)
```



以上がロジスティック回帰での手順であるが、分析手法を変えるために新たなパッケージを利用するとすれば、違う手順が必要になる。

7.8 サポートベクターマシン

例えば、サポートベクターマシンを使う場合の手順は以下のようになる。ここでは kernlab1 パッケージを使い、ガウシアン・カーネル (rbfdot) で推定を行う。

引数 type には、判別の種類を指定する。ここでは「分類」を意味する C-svc を与えている。

```
library(kernlab)
```

```
svm_kernlab  <- ksvm(Y ~ ., data = training,  scaled = FALSE,
          type = "C-svc", kernel = "rbfdot", cross = 10)
```

結果を確認する。モデルに特有のパラメータ C とガウシアンカーネルのハイパーパラメータ sigma が推定されていることに注意されたい。

```
svm_kernlab
```

Support Vector Machine object of class "ksvm"

SV type: C-svc (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma = 0.555555555555556

Number of Support Vectors : 3021

Objective Function Value : -1915
Training error : 0.180385
Cross validation error : 0.301

```
svm_preds <- predict(svm_kernlab, test_X)
```

```
confusionMatrix(table(svm_preds, test_Y), positive = "TRUE")
```

Confusion Matrix and Statistics

```
          test_Y
svm_preds FALSE TRUE
    FALSE  1018  445
    TRUE     32  107
```

               Accuracy : 0.702
                 95% CI : (0.679, 0.725)
    No Information Rate : 0.655
    P-Value [Acc > NIR] : 3.75e-05

                  Kappa : 0.199

 Mcnemar's Test P-Value : < 2e-16

            Sensitivity : 0.1938
            Specificity : 0.9695
         Pos Pred Value : 0.7698
         Neg Pred Value : 0.6958

Prevalence : 0.3446
Detection Rate : 0.0668
Detection Prevalence : 0.0868
Balanced Accuracy : 0.5817

'Positive' Class : TRUE

8 caret パッケージの活用

分割したデータに対して、分析方法ごとに選んだパッケージに用意された関数を適用する作業は、試行する手法が多数に登る場合、手間であり、またミスもしやすい。もちろん、手法に特有のパラメータもあるため、これらは手作業で調整する必要があるのだが、せめてモデルの当てはめとテストデータへの適用は共通化したい。

こうした場合、caret パッケージや mlr パッケージ、さらに tidymodels パッケージを援用することで、作業が混乱するのを避けることができるかも知れない。

caret には訓練データを適用する train() が用意されている。例えば、train() では、以下のようにパラメータ、分析方法、データとその前処理（例えば標準化する、主成分にするなど）を指定できる。

これらの引数のうち method には分析の手法を指定するが、ここには手法を、また tuneGride に、手法特有のパラメータ（の探索範囲）を指定する。また、一般に機械学習の手法を適用するには説明変数を標準化する（決定木やランダムフォレストでは必要ない）。あるいは、説明変数の主成分を利用する場合もある。train() では preProcess 引数で指定できる。

```
caret::train(x = 説明変数
       , y = 目的変数,
       , method = 分析手法,
       , tuneGrid = モデルパラメータの範囲, # あるいはtuneLength=10など
       , trControl = 交差法の指定,
       , preProcess = データの加工指定
)
```

ちなみに tuneLength という引数に適当な整数を指定すると、パラメータについて適当な組み合わせを設定してくれる。

caret でのモデリングについては https://topepo.github.io/caret/index.html に詳しい。

8.0.1 caret による Lasso 回帰

最初に実行した Lasso 回帰を caret 流に直せば以下のようになる。ここでは α の範囲を 0 から 1 まで 0.01 刻みの数列を、また λ については 10^{-10} から 1 (10^1) までの数列を指定している（以下、実際に実行するとかなりの時間がかかります）。preProcess を指定することで説明変数の標準化を行うことができるが、ここでは利用しない。

なお、オリジナルの cv.glmnet() では説明変数と目的変数を分けて指定する必要があったが、caret の train() ではモデル式 Y ~ X が使える。

```
caret_lasso  <- train(Y ~ ., data = training, method = "glmnet"
```

```
    , tuneGrid = expand.grid(alpha   = seq(0, 1, by = 0.01), lambda = 10^(0:5 * -1))
    , trControl = trainControl(method = "cv", number = 10)
   #, preProcess = c("center", "scale")
)
```

最良のモデルを抽出する。

```
caret_lasso$bestTune
```

```
    alpha lambda
202  0.33   0.01
```

```
coef(caret_lasso$finalModel, caret_lasso$finalModel$lambdaOpt) %>% head()
```

```
[1] -0.747 -0.929  1.511  0.000  0.000  0.000
```

予測精度を確認しよう。

```
caret_ls_preds <- predict(caret_lasso, testing)
```

```
confusionMatrix(table(testing$Y, caret_ls_preds), positive = "TRUE")
```

```
Confusion Matrix and Statistics

       caret_ls_preds
        FALSE TRUE
  FALSE   961   89
  TRUE    395  157

               Accuracy : 0.698
                 95% CI : (0.675, 0.72)
    No Information Rate : 0.846
    P-Value [Acc > NIR] : 1

                  Kappa : 0.23

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.638
            Specificity : 0.709
         Pos Pred Value : 0.284
         Neg Pred Value : 0.915
             Prevalence : 0.154
```

```
        Detection Rate : 0.098
  Detection Prevalence : 0.345
     Balanced Accuracy : 0.673

      'Positive' Class : TRUE
```

8.0.2 caret によるサポートベクターマシン

同じく、caret の train() でサポートベクターマシンのモデルを訓練してみよう。ロジスティック回帰の場合と異なるのは、method と tuneGrid の指定である。これは先に rbfdot を指定したが、caret::train() では "svmRadial" とする。指定できるメソッドについては https://topepo.github.io/caret/train-models-by-tag.html で確認できる。また、サポートベクターマシンのパラメータ C と、カーネルのハイパーパラメータ sigma について、探索範囲を tuneGrid に指定する。

このように、caret を使うことで、パッケージ間の違いをある程度吸収することができる（なお、以下は実行結果が返ってくるまでに数時間がかかるので注意）。

```
caret_kernlab  <- train(Y ~ ., data = training, method = "svmRadial"
    , tuneGrid = expand.grid(C = 1:10, sigma = seq(0.0, 0.9, by = 0.1))
    , trControl = trainControl(method = "cv", number = 10)
   #, preProcess = c("center", "scale")
)
```

推定されたパラメータを確認する。

```
caret_kernlab$bestTune
```

```
  sigma C
2   0.1 1
```

テストデータで予測を行い、精度を確認する。

```
caret_svm_preds <- predict(caret_kernlab, testing)
```

```
confusionMatrix(table(testing$Y, caret_svm_preds), positive = "TRUE")
```

```
Confusion Matrix and Statistics

       caret_svm_preds
        FALSE TRUE
  FALSE  1022   28
  TRUE    470   82

               Accuracy : 0.689
                 95% CI : (0.666, 0.712)
```

No Information Rate : 0.931
P-Value [Acc > NIR] : 1

Kappa : 0.15

Mcnemar's Test P-Value : <2e-16

Sensitivity : 0.7455
Specificity : 0.6850
Pos Pred Value : 0.1486
Neg Pred Value : 0.9733
Prevalence : 0.0687
Detection Rate : 0.0512
Detection Prevalence : 0.3446
Balanced Accuracy : 0.7152

'Positive' Class : TRUE

9 tidymodels

caret 開発者による tidy な機械学習モデルとして注目されている。処理手順に「料理」のプロセスを思い起こさせる名前が付けられている。

9.1 tidymodels で glmnet

始めにデータを分割するため、initial_split() を利用する。

```
library(tidymodels)
splitted_data <- initial_split(iPhone, p = 0.5, strata = c('Y'))
training_data <- training(splitted_data)
test_data <- testing(splitted_data)
```

交差法を使う場合は vfold_cv() を指定する（以下の説明では交差法の指定は省いている）。

```
# train_data <- train_data %>% vfold(v = 10)
```

次に recipe() で分析の方針を定義する。これをレシピという。レシピによって、目的変数と説明変数の対応（モデル）を指定する。また、変数の変換を行いたければ step_log() や step_center() step_scale() を指定する。レシピが用意できたら、訓練用データに prep() で下ごしらえの方針を指定する。

```
rec <- recipe(Y ~ ., data = training_data)
# 説明変数を標準化する場合
# rec <- rec %>% step_center(all_predictors()) %>%
#   step_scale(all_predictors())
```

```
rec_dat <- rec %>% prep(training = training_data)
```

下ごしらえの方針を立てた後、訓練用データとテスト用データのそれぞれに適用する。訓練データは juice() で、またテストデータは bake() で味付けする。これでデータの準備が完了したことになる。

```
train_juced <- rec_dat %>% juice()
test_baked <- rec_dat %>% bake(test_data)
```

その上で、分析方針に適切な機械学習の手法を選択する。ここでは logistic_reg() でロジスティック回帰を指定し、Lasso 回帰のためのパラメータを設定する。また解析のエンジンには glmnet を利用する。

```
glmnet_model_tidy <- logistic_reg(mixture = 0, penalty = 10^(0:10 * -1)) %>%
        set_engine("glmnet")
```

準備が整ったらモデルの学習を始める。

```
lasso_tidy <- glmnet_model_tidy %>% fit(Y ~ ., data = train_juced)
```

モデルの推定が行われたら、ペナルティの値ごと予測を行う。これには multi_predict() を使う。

```
preds  <- test_baked %>% select(Y) %>% bind_cols(fitted = multi_predict(lasso_tidy, test_baked))
preds
```

```
# A tibble: 2,671 x 2
  Y     .pred
  <fct> <list>
 1 FALSE <tibble [11 × 2]>
 2 TRUE  <tibble [11 × 2]>
 3 FALSE <tibble [11 × 2]>
 4 TRUE  <tibble [11 × 2]>
 5 TRUE  <tibble [11 × 2]>
 6 TRUE  <tibble [11 × 2]>
 7 TRUE  <tibble [11 × 2]>
 8 TRUE  <tibble [11 × 2]>
 9 TRUE  <tibble [11 × 2]>
10 TRUE  <tibble [11 × 2]>
# … with 2,661 more rows
```

このデータフレームは .pred 列に、さらにデータフレーム (tibble) が入れ子になっており、ペナルティの値ごとに予測値を求めた結果になっている。最初のレコードに対する予測値を確認するには次のようにすればよい。

```
preds$.pred[[1]]
```

```
# A tibble: 11 x 2
      penalty .pred
```

```
          <dbl> <fct>
 1 0.0000000001 FALSE
 2 0.000000001  FALSE
 3 0.00000001   FALSE
 4 0.0000001    FALSE
 5 0.000001     FALSE
 6 0.00001      FALSE
 7 0.0001       FALSE
 8 0.001        FALSE
 9 0.01         FALSE
10 0.1          FALSE
11 1            FALSE
```

この出力について、さらにペナルティを 0.001 とした場合の予測精度だけを出力するには次のように添字を加える。

```
preds$.pred[[1]][3, ]
```

```
# A tibble: 1 x 2
     penalty .pred
       <dbl> <fct>
1 0.00000001 FALSE
```

ただし、このままでは使いにくいので、unnest() を使って .$pred 列を展開してしまう。

```
preds_flat <- preds %>% mutate(.pred = map(.pred, bind_rows)) %>% unnest()
```

penalty 列の値ごとグループ化して、metrics() で、それぞれについて予測精度を求める。

```
preds_flat %>% group_by(penalty) %>% metrics(Y, .pred)
```

```
# A tibble: 22 x 4
         penalty .metric  .estimator .estimate
           <dbl> <chr>    <chr>          <dbl>
 1 0.0000000001 accuracy binary         0.684
 2 0.000000001  accuracy binary         0.683
 3 0.00000001   accuracy binary         0.682
 4 0.0000001    accuracy binary         0.683
 5 0.000001     accuracy binary         0.683
 6 0.00001      accuracy binary         0.684
 7 0.0001       accuracy binary         0.689
 8 0.001        accuracy binary         0.692
 9 0.01         accuracy binary         0.695
```

```
10 0.1          accuracy binary         0.702
# ... with 12 more rows
```

9.1.1 tidymodels でサポートベクター

kernlab ＊また *caret で実行したサポートベクターマシンを tidymodels で実現するには svm_rbf() を指定する。

```
svm_tidy <- svm_rbf(mode = "classification",
       cost = 1:10, rbf_sigma = seq(0.0, 0.9, by = 0.1)) %>%
          set_engine("kernlab")

svm_tidy_fit <- svm_tidy %>% fit(Y ~ ., data = train_baked)
```

9.1.2 tidymodels でランダムフォレスト

tidymodels ではランダムフォレストを実行するメソッド rand_forest() が用意されている（厳密には parsnip パッケージで実装されている）ので、以下のような指定になる。set_engine() には、例えば ranger パッケージを指定することもできる。

```
rf_tidy <- rand_forest(mode = "classification", trees = 20,
          min_n = 100, mtry = 1000) %>%
    set_engine("randomForest", num.threads =
          parallel::detectCores()/2, seed = 123)
rf_tidy_fit <- rf_tidy %>% fit(Y ~ ., data = train_baked)
```

10 参考文献

松村優哉，湯谷啓明，紀ノ定保礼，前田和寛『R ユーザのための RStudio［実践］入門—tidyverse によるモダンな分析フローの世界—』技術評論社

石田基広『R によるテキストマイニング入門 第二版』森北出版

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani『R による 統計的学習入門』朝倉書店

Sebastian Raschka , Vahid Mirjalili『[第 2 版]Python 機械学習プログラミング 達人データサイエンティストによる理論と実践』impress top gear