5_linalg_statistics

2017 年 3 月 16 日

1 Introduction to Data Science with Julia

2 目次

3 線形代数

Julia では線形代数の計算が標準機能として備わっています。
行列 A と行列 B の掛け算は

```
A * B
```

と書くだけです。

```
In [1]: A = reshape(1:9, 3, 3)
        B = rand(1:10, 3, 3)
        @show A
        @show B
        @show A * B
```

```
A = [1 4 7; 2 5 8; 3 6 9]
B = [3 6 2; 5 5 1; 1 6 3]
A * B = [30 68 27; 39 85 33; 48 102 39]
```

```
Out[1]: 3 × 3 Array{Int64,2}:
         30   68  27
         39   85  33
         48  102  39
```

逆行列や行列式、固有値、固有ベクトルなども簡単に計算することが出来ます。

```
In [2]: @show C = rand(3, 3)
        inv(C) # 逆行列
```

```
C = rand(3,3) = [0.0316126 0.605445 0.0169347; 0.222539 0.544538 0.875881; 0.725021 0.29
```

```
Out[2]: 3 × 3 Array{Float64,2}:
         -0.686293  -0.00696504   1.41133
          1.71241   -0.0321798   -0.0647877
         -0.890242   1.16348     -0.318304
```

```
In [3]: det(C) # 行列式
```

```
Out[3]: 0.3692088373360544
```

```
In [4]: eigvals(C) # 固有値
```

```
Out[4]: 3-element Array{Complex{Float64},1}:
           1.17899+0.0im
         -0.295143+0.475444im
         -0.295143-0.475444im
```

```
In [5]: eigvecs(C) # 固有ベクトル
```

```
Out[5]: 3 × 3 Array{Complex{Float64},2}:
         -0.419653+0.0im  -0.0956797+0.556813im  -0.0956797-0.556813im
         -0.782415+0.0im     -0.4028-0.375644im     -0.4028+0.375644im
         -0.460129+0.0im    0.614368+0.0im         0.614368-0.0im
```

```
In [6]: trace(C) # トレース
```

```
Out[6]: 0.5887077924636039
```

```
In [7]: rank(C) # ランク
```

```
Out[7]: 3
```

```
In [8]: x = [1, 2, 3]
        y = [4, 5, 6]
        dot(x,y) # 内積
```

```
Out[8]: 32
```

```
In [ ]:
```

線形方程式 $Ax = y$ を解く場合には

```
A \ y # ￥ はバックスラッシュ
```

とします。行列 A の逆行列は inv(A) として求められるので

```
inv(A) * y
```

としても同様の結果が得られますが、A y の方がより計算精度が上がります。

```
In [9]: A = rand(3,3)
        y = rand(3)
        A \ y
```

```
Out[9]: 3-element Array{Float64,1}:
         -0.309406
          0.761443
         -0.538079
```

```
In [ ]:
```

その他の行列計算に関しては公式ドキュメントを読んでください。

目次に戻る

4 統計量の計算

4.1 量的データ (numerical data)

Julia では平均や分散などを計算する関数が標準で備わっています。

```
In [10]: # 平均 (mean)
         @show x = rand(5)
         mean(x)
```

```
x = rand(5) = [0.353628,0.903524,0.159742,0.810534,0.950088]
```

```
Out[10]: 0.6355032675516282
```

```
In [11]: # 分散 (variance)
         @show x = rand(5)
         var(x) # 補正をなくす場合は、 var(x, corrected=false) とする
```

```
x = rand(5) = [0.885045,0.751102,0.742,0.48118,0.592068]
```

```
Out[11]: 0.024419059408298135
```

```
In [12]: # 標準偏差 (standard deviation)
         @show x = rand(5)
         std(x) # 補正をなくす場合は、 std(x, corrected=false) とする
```

```
x = rand(5) = [0.101201,0.790782,0.429018,0.421794,0.422853]
```

```
Out[12]: 0.24410191862505906
```

```
In [ ]:
```

```
In [13]: # 中央値 (median)
         x = 1:5
         median(x)
```

```
Out[13]: 3.0
```

```
In [14]: # 第 1 四分位点（lower quartile）
         quantile(x, 1/4)
```

```
Out[14]: 2.0
```

```
In [15]: # 第 1 四分位点, 中央値, 第 3 四分位点（upper quartile）
         quantile(x, [1/4, 1/2, 3/4])
```

```
Out[15]: 3-element Array{Float64,1}:
          2.0
          3.0
          4.0
```

```
In [16]: # 最大値
         maximum(x)
```

```
Out[16]: 5
```

```
In [17]: # 最小値
         minimum(x)
```

```
Out[17]: 1
```

```
In [18]: # 最小値と最大値を同時に計算
         extrema(x)
```

```
Out[18]: (1,5)
```

```
In [19]: # 相関係数
         x = 1.0:1.0:12.0
         y = x .+ randn(length(x))
         cor(x,y)
```

```
Out[19]: 0.9835687891956586
```

```
In [ ]:
```

mean や var などで 2 次元配列を引数に取ると要素全体の平均などになりますが、第 2 引数に 1 と指定すると列ごとの平均、 2 を指定すると行ごとの平均になります。

```
In [20]: x = [1 2 3
              4 5 6
              7 8 9]
         @show mean(x)
         @show mean(x, 1) # 列ごとの平均
```

```
@show mean(x, 2) # 行ごとの平均
```

```
mean(x) = 5.0
mean(x,1) = [4.0 5.0 6.0]
mean(x,2) = [2.0; 5.0; 8.0]
```

```
In [ ]:
```

目次に戻る

StatsBase パッケージを使うと統計量の計算がさらにやりやすくなります。

```
In [21]: using StatsBase
```

summarystats を使うと、平均と五数要約（five-number summary）を一度に計算できます。

```
In [22]: @show x = rand(10)
         result = StatsBase.summarystats(x)
```

```
x = rand(10) = [0.849495,0.102439,0.979398,0.132961,0.448198,0.801706,0.613595,0.511562,
```

```
In [23]: typeof(result)
```

```
Out[23]: StatsBase.SummaryStats{Float64}
```

```
In [24]: fieldnames(result) # 各データへアクセスするための名前
```

```
Out[24]: 6-element Array{Symbol,1}:
          :mean
          :min
          :q25
          :median
          :q75
          :max
```

```
In [25]: result.mean
```

```
Out[25]: 0.5376001014419926
```

```
In [26]: result.q25
```

```
Out[26]: 0.33350458271318084
```

四分位範囲 (Interquartile Range, IQR) を計算するには iqr を使います。

```
In [27]: StatsBase.iqr(x)
```

```
Out[27]: 0.42811837334235747
```

Z-score の計算も出来ます。

```
In [28]: StatsBase.zscore(x)
```

```
Out[28]: 10-element Array{Float64,1}:
           1.04662
          -1.46026
           1.48253
          -1.35784
          -0.300004
           0.886252
           0.255013
          -0.0873765
           0.348236
          -0.81317
```

```
In [ ]:
```

目次に戻る

4.2 質的データ (categorical data)

質的データを調べるときには countmap や proportionmap を使うと便利です。共通要素が何個（何割）あるのかがわかります。

返り値は各要素を key とする辞書です。

```
In [29]: @show x = rand('a':'c', 10)
         StatsBase.countmap(x) # 共通要素の個数
```

```
x = rand('a':'c',10) = ['a','a','b','b','b','a','b','b','b','b']


Out[29]: Dict{Char,Int64} with 2 entries:
           'b' => 7
           'a' => 3
```

```
In [30]: abc = StatsBase.proportionmap(x) # 共通要素の全体の割合
```

```
Out[30]: Dict{Char,Float64} with 2 entries:
           'b' => 0.7
           'a' => 0.3
```

```
In [31]: abc['a']
```

```
Out[31]: 0.3
```

目次に戻る

4.3 度数分布

まずは Plots を使ってヒストグラムを書いてみましょう。

```
In [32]: import Plots
         Plots.gr(leg=false)
```

```
Out[32]: Plots.GRBackend()

In [33]: nd = randn(1000) # 正規分布に従う乱数 1000 点
         Plots.histogram(nd)
```

bin を変える場合は

```
nbins = 20
```

や

```
nbins = -5:0.5:5
```

などとします。分割数を指定する場合は前者で、分割幅を決めたい場合は後者を使うと便利です。

```
In [34]: Plots.histogram(nd, nbins = 20) # 20 分割

In [35]: Plots.histogram(nd, nbins = -5.0:0.5:5.0) # bin 幅を 0.5 にし、-5 〜 5 の範囲でプロッ

In [36]: Plots.histogram(nd, nbins = -5.0:0.5:5.0, norm=true) # 正規化
```

Plots を使ってヒストグラムを書けばどのような分布なのかということはわかりますが、各 bin の中に何サンプルあるのかという具体的な数値はわかりません。

具体的な数値を知りたい場合は

```
fit(Histogram, nd, nbins = 20)
or
fit(Histogram, nd, -5.0:0.5:5.0)
```

などとします。

結果は

```
StatsBase.Histogram{Int64,1,Tuple{FloatRange{Float64}}}
edges:
  -3.5:0.5:3.0
weights: [1,2,13,46,84,149,173,206,164,83,46,27,6]
closed: right
```

のようになります。ここで edges は範囲と分割幅、weights が度数を表します。 各値は 変数名.edges や 変数名.weights をすることで抜き出すことが出来ます。

```
In [37]: ndfreq1 = fit(Histogram, nd, nbins = 20) # 分割数を指定

Out[37]: StatsBase.Histogram{Int64,1,Tuple{FloatRange{Float64}}}
         edges:
           -3.5:0.5:3.5
         weights: [3,6,12,43,79,166,188,179,162,80,50,26,5,1]
         closed: right

In [38]: ndfreq2 = fit(Histogram, nd, -5.0:0.5:5.0) # 幅と範囲を指定

Out[38]: StatsBase.Histogram{Int64,1,Tuple{FloatRange{Float64}}}
```

```
          edges:
            -5.0:0.5:5.0
          weights: [0,0,0,3,6,12,43,79,166,188,179,162,80,50,26,5,1,0,0,0]
          closed: right
```

```
In [39]: ndfreq2.edges
```

```
Out[39]: (-5.0:0.5:5.0,)
```

```
In [40]: ndfreq2.weights
```

```
Out[40]: 20-element Array{Int64,1}:
             0
             0
             0
             3
             6
            12
            43
            79
           166
           188
           179
           162
            80
            50
            26
             5
             1
             0
             0
             0
```

求めた結果をヒストグラムと一緒に描写してみます。

```
In [41]: Plots.histogram(nd, nbins = -5.0:0.5:5.0)
         x = [(ndfreq2.edges[1][i] + ndfreq2.edges[1][i+1])/2 for i in 1:length(ndfreq2.
         Plots.plot!(x, ndfreq2.weights, marker=:circle)
```

```
In [ ]:
```

最頻値 (mode) を求める場合は mode を使います。

```
In [42]: @show a = rand('A':'C', 10)
         StatsBase.mode(a)
```

```
a = rand('A':'C',10) = ['A','B','A','B','A','A','A','A','C','C']
```

```
Out[42]: 'A'
```

```
In [ ]:
```

目次に戻る

5　回帰直線

Julia で回帰直線の切片、傾きを求めるには linreg 関数を使います。

```
In [43]: ?linreg
```

```
search: linreg linearindices LineNumberNode
```

```
Out[43]:
```

```
linreg(x, y)
```

Perform simple linear regression using Ordinary Least Squares. Returns `a` and `b` such that `a + b*x` is the closest straight line to the given points `(x, y)`, i.e., such that the squared error between `y` and `a + b*x` is minimized.

Examples:

```
using PyPlot
x = 1.0:12.0
y = [5.5, 6.3, 7.6, 8.8, 10.9, 11.79, 13.48, 15.02, 17.77, 20.81, 22.0, 22.99]
a, b = linreg(x, y)           # Linear regression
plot(x, y, "o")               # Plot (x, y) points
plot(x, a + b*x)              # Plot line determined by linear regression
```

See also:

`\, cov, std, mean`

```
In [44]: # y = a + b * x
         x = 1.0:12.0
         y = [5.5, 6.3, 7.6, 8.8, 10.9, 11.79, 13.48, 15.02, 17.77, 20.81, 22.0, 22.99]
         a, b = linreg(x, y)
```

```
Out[44]: (2.5559090909090916,1.696013986013986)
```

求めた結果を使ってプロットしてみます。

```
In [45]: Plots.plot(x,y, linetype=:scatter)
         Plots.plot!(x, a + b*x)
```

```
In [ ]:
```

目次に戻る

6　アンスコムの例

データ分析において可視化することがいかに重要かということを知ることの出来る良い例とアンスコムの例 (Anscombe's quartet) というものがあります。

アンスコムの例は 4 つのデータセットからなり、それぞれのデータセットの平均や分散、回帰直線などはほとんど同じなのに、散布図にすると似ても似つかない分布になるという面白い例です。

アンスコムの例のデータセットは統計ソフト R のデータセットから読み込むことが出来ます。

```
In [46]: using RDatasets
         anscombe = RDatasets.dataset("datasets","anscombe")
```

```
Out[46]: 11 × 8 DataFrames.DataFrame
         | Row | X1 | X2 | X3 | X4 | Y1    | Y2   | Y3    | Y4   |
         |-----|----|----|----|----|-------|------|-------|------|
         | 1   | 10 | 10 | 10 | 8  | 8.04  | 9.14 | 7.46  | 6.58 |
         | 2   | 8  | 8  | 8  | 8  | 6.95  | 8.14 | 6.77  | 5.76 |
         | 3   | 13 | 13 | 13 | 8  | 7.58  | 8.74 | 12.74 | 7.71 |
         | 4   | 9  | 9  | 9  | 8  | 8.81  | 8.77 | 7.11  | 8.84 |
         | 5   | 11 | 11 | 11 | 8  | 8.33  | 9.26 | 7.81  | 8.47 |
         | 6   | 14 | 14 | 14 | 8  | 9.96  | 8.1  | 8.84  | 7.04 |
         | 7   | 6  | 6  | 6  | 8  | 7.24  | 6.13 | 6.08  | 5.25 |
         | 8   | 4  | 4  | 4  | 19 | 4.26  | 3.1  | 5.39  | 12.5 |
         | 9   | 12 | 12 | 12 | 8  | 10.84 | 9.13 | 8.15  | 5.56 |
         | 10  | 7  | 7  | 7  | 8  | 4.82  | 7.26 | 6.42  | 7.91 |
         | 11  | 5  | 5  | 5  | 8  | 5.68  | 4.74 | 5.73  | 6.89 |
```

ここで、読み込んだデータは DataFrame という配列に似たものです。詳細は次回。まずは、4 つのデータをプロットしてみます。

DaatFrame 型は各列を

```
anscombe[:X1]
```

のように列名で抜き出すことが出来ます。

```
In [47]: Plots.scatter(anscombe[:X1], anscombe[:Y1], xlims=(0,20))
```

```
In [48]: Plots.scatter(anscombe[:X2], anscombe[:Y2], xlims=(0,20))
```

```
In [49]: Plots.scatter(anscombe[:X3], anscombe[:Y3], xlims=(0,20))
```

```
In [50]: Plots.scatter(anscombe[:X4], anscombe[:Y4], xlims=(0,20))
```

4 つの散布図は似ても似つきません。しかし、統計量は非常に近い値を取ります。

```
In [51]: # Introducing Julia/DataFrames - Wikibooks, open books for an open world
         # https://en.wikibooks.org/wiki/Introducing_Julia/DataFrames#Plotting_Anscombe.

         # print a header
```

```
println("Column\tMeanX\tMedianX\tStdDev X\tMeanY\t\t\tStdDev Y\t\tCorr\t")
map((xcol,ycol) -> println(
    xcol,                   "\t",
    mean(anscombe[xcol]),   "\t",
    median(anscombe[xcol]), "\t",
    std(anscombe[xcol]),    "\t",
    mean(anscombe[ycol]),   "\t",
    std(anscombe[ycol]),    "\t",
    cor(anscombe[xcol], anscombe[ycol])),

    [:X1, :X2, :X3, :X4],
    [:Y1, :Y2, :Y3, :Y4]);
```

```
Column        MeanX        MedianX        StdDev X        MeanY                      S
X1        9.0        9.0        3.3166247903554        7.500909090909093        2.031568
X2        9.0        9.0        3.3166247903554        7.500909090909091        2.031656
X3        9.0        9.0        3.3166247903554        7.5000000000000001        2.030423
X4        9.0        8.0        3.3166247903554        7.50090909090909        2.0305785
```

In [ ]:

目次に戻る

7 練習問題

7.1 1.

線形方程式 $

$$\begin{cases} x + 2y = -1 \\ 3x + y = 2 \end{cases} \tag{1}$$

$ を解け

In [ ]:

7.2 2.

以下のコード

```
srand(1)
scores = rand(0:100, 100, 3)
```

を実行し、配列 scores の 1. 各列の合計を求めよ。 1. 各列の平均を求めよ。1. 各列の分散を求めよ。

In [ ]:

7.3 3.

次のコードを実行するとタイタニックの乗客乗員の情報を読み込むことが出来る。

```
using RDatasets
titanic = RDatasets.dataset("COUNT", "titanic")
```

	Survived	Age	Sex	Class
1	1	1	1	1
2	2	1	1	1
3	3	1	1	1
4	4	1	1	1
5	5	1	1	1
6	6	1	1	1

ここで各列の数字の意味は以下のとおりである。 Survived 1=survived; 0=died
Age 1=adult; 0=child
Sex 1=Male; 0=female
Class ticket class 1= 1st class; 2= second class; 3= third class

1. 生存者と死者の数をそれぞれ調べよ。
2. 男女別に生存者と死者の数をそれぞれ調べよ。
3. 年齢別に生存者と死者の数をそれぞれ調べよ。
4. チケットの階級別に生存者と死者の数をそれぞれ調べよ。
5. 1〜4 の結果を図示せよ。

```
In [52]: using RDatasets
         titanic = RDatasets.dataset("COUNT", "titanic")
         head(titanic)
```

```
Out[52]: 6 × 4 DataFrames.DataFrame
         | Row | Survived | Age | Sex | Class |
         |-----|----------|-----|-----|-------|
         | 1   | 1        | 1   | 1   | 1     |
         | 2   | 1        | 1   | 1   | 1     |
         | 3   | 1        | 1   | 1   | 1     |
         | 4   | 1        | 1   | 1   | 1     |
         | 5   | 1        | 1   | 1   | 1     |
         | 6   | 1        | 1   | 1   | 1     |
```

```
In [ ]:
```

7.4 4.

福岡市の月ごとの降水量を調べ棒グラフで図示せよ。

In [ ]:

目次に戻る