データサイエンス
第2回

情報理工学系研究科
創造情報学専攻
中山 英樹

本日の内容

- 統計基礎
  - 検定、推定
- データ加工、操作
- (多変量解析)

統計基礎

- 巷でよくある風潮
  - データサイエンティスト = ビッグデータを扱う人？
  - とにかく全データを使えばご利益があるらしい？

  - 間違ってはいないが、目的と手段が逆転している

- 正しくデータを使い、判断や意思決定を行うためには
  統計的検定や信頼区間の理解が重要

身近でよくあるパターン

- 修論締切まであとx日…
  - ①「新しいシステム作ったけど何か評価しないと…
    とりあえずラボのみんなに頼んで主観評価しよう！」
    - 従来システム：”2, 3, 1, 2, 3” （平均2.2点）
    - 提案システム：”3, 2, 4, 3, 3” （平均3.0点）
    - 勝った！(完)
  - ② 10人にアンケートをとり○○と△△の関係を調べたところ、
    相関係数0.6と一定の相関が見られた。

帰無仮説、対立仮説、p値

帰無仮説：　何も起きない　（e.g. 二つの変数に相関はない）
対立仮説：　何かが起こる（e.g. 二つの変数に相関がある）

- 1. 帰無仮説が真であると仮定する
- 2. 検定統計量を計算（簡単な場合では、標本の平均など。統計量の分布が分かっている必要がある。）
- 3. 統計量とその分布からp値を求める。p値は、帰無仮説が真であると仮定した場合に検定統計量が得られる確率
- 4. p値が小さすぎる場合、帰無仮説を支持しない状況証拠になる
  → 帰無仮説を棄却する。

ことば

- 標本 (sample)
- 標本の大きさ (標本サイズ, sample size)
- 標本数
- 母数
- 大数の法則
- 中心極限定理

正しく理解できていますか？

中心極限定理

- 標本を抽出する母集団が平均μ、分散σ^2の確率分布に従う時、母集団の従う確率分布がどのような分布であっても、抽出するサンプルサイズn が大きくなるにつれて標本平均の分布は平均μ 、分散 σ^2 / n　の正規分布 に近づく

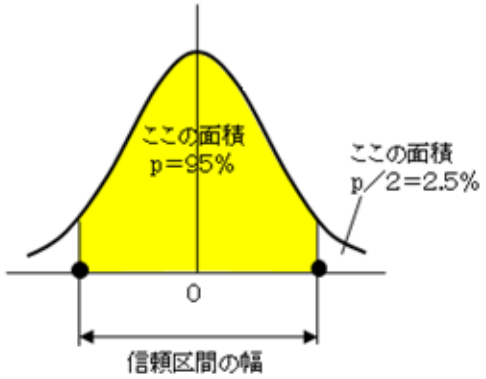
※母集団に平均、分散が存在する必要あり

平均値の検定 (※母集団の分散 σ^2 が既知の場合)

母集団の標本がある。この標本から、母集団の平均が特定の値 μ になり得るかどうかを知りたい

帰無仮説: 母集団の平均は μ である
対立仮説: 母集団の平均が μ 以外になる

- 標準正規分布を用いた検定（中心極限定理を利用）



http://www.kogures.com/hitoshi/webtext

$$x = \frac{\mu_0 - \mu}{\sigma / \sqrt{n}}$$

μ_0 標本の平均

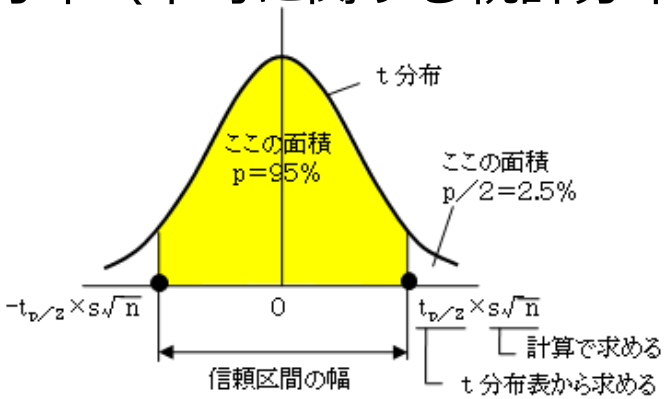
n 標本サイズ

平均値の検定 (t検定)

母集団の標本がある。この標本から、母集団の平均が特定の値 μ になり得るかどうかを知りたい

帰無仮説: 母集団の平均は μ である
対立仮説: 母集団の平均が μ 以外になる

- t分布（平均に関する統計分布）を用いた検定



http://www.kogures.com/hitoshi/webtext

$$t = \frac{\mu_0 - \mu}{s_0 / \sqrt{n}}$$

μ_0, s_0 標本の平均、標準偏差

n 標本サイズ

$$f(t) = \frac{\Gamma\left((\nu+1)/2\right)}{\sqrt{\nu\pi}\,\Gamma\left(\nu/2\right)}\left(1+t^2/\nu\right)^{-(\nu+1)/2} \qquad \nu = n-1$$

平均値の検定 (t検定)

- t.test 関数 (R)

  *p*値を求める。慣例的に、*p*>0.05 であれば帰無仮説を棄却できない
  = 平均が μ_0 であることに不利な証拠を示していない（有意水準5%)
  (対立仮説が証明されたわけではないことに注意)

> x<-rnorm(50, mean=100, sd=15) #平均100, 標準偏差15の正規分布からランダムに50点サンプリングし、標本とする

> t.test(x, mu=110)

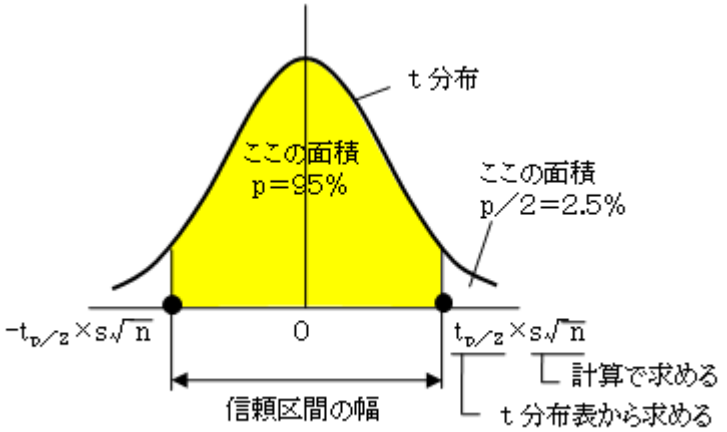
> t.test(x, mu=100)

~ p-value = 0.5224

```
> t.test(x,mu=110)

        One Sample t-test

data:  x
t = -3.3702, df = 49, p-value = 0.001472
alternative hypothesis: true mean is not
equal to 110
95 percent confidence interval:
  96.59886 106.61056
sample estimates:
mean of x
 101.6047
```



t検定の前提

- 基本的には、母集団が正規分布に従っている必要がある
- ただし、標本サイズが大きくなれば、中心極限定理により実用上十分な近似精度となることが示される
  - n>30 が一つの目安とされる（多ければ多いほどよい）

- 中心極限定理
  - 母集団の分布がどんな分布であっても、標本サイズを大きくすると、標本平均と真の平均の誤差は近似的に正規分布に従う
    - （厳密には母集団に関する条件あり）

2つの標本の平均を比較する

- ①「新しいシステム作ったけど何か評価しないと…
  とりあえずラボのみんなに頼んで主観評価しよう！」
  - 従来システム：”2, 3, 1, 2, 3” （平均2.2点）
  - 提案システム：”3, 2, 4, 3, 3” （平均3.0点）

2つの母集団からそれぞれ選びだした標本がある。これらから、
2つの母集団の平均が同じになりえるかどうかを知りたい

母集団： ターゲットユーザー全員（それぞれのシステムごとに）
標本： ラボのみんな（本当それでよいだろうか？）
帰無仮説： 2つの母集団の平均は同じ である
対立仮説： 2つの母集団の平均は異なる

※標本は母集団からランダムサンプリングしないと正確な統計にならない
また、母集団は正規分布に従うことを仮定

2つの標本の平均を比較する

- t.test 関数
  - *p*値を求める。慣例的に、*p*<0.05の場合は平均がおそらく異なる（同じになる可能性は十分小さい）ことを意味し、*p*>0.05の場合は平均が異なる証拠にならない

```
> x<-c(2, 3, 1, 2, 3)
> y<-c(3, 2, 4, 3, 3)
> t.test(x,y,paired=TRUE)
```

```
> t.test(x,y,paired=TRUE)

        Paired t-test

data:  x and y
t = -1.206, df = 4, p-value = 0.2943
alternative hypothesis: true difference in
means is not equal to 0
```

相関の検定

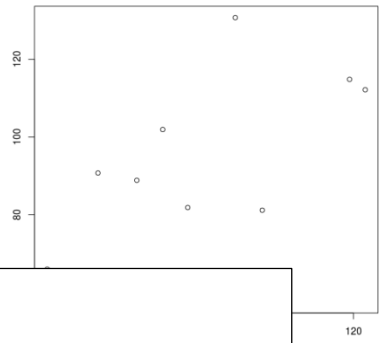
帰無仮説：　2つの母集団に相関はない
対立仮説：　2つの母集団に相関がある

- cor.test 関数 (デフォルトではピアソン相関を求める)

```
> x<-rnorm(50, mean=100, sd=15)
> y<-x+rnorm(50, sd=15)　# xに正規分布によるノイズを加える
> cor.test(x[1:10],y[1:10])　# 最初の10点で検定してみる
```

```
> cor.test(x, y)　#50点では？
```

~ p-value = 2.448e-05




```
> cor.test(x[1:10],y[1:10])

        Pearson's product-moment correlation

data:  x[1:10] and y[1:10]
t = 1.9674, df = 8, p-value = 0.08468
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.09148762  0.88313774
sample estimates:
      cor
0.5710318
```

一般に、サンプル数が少ない時は相関係数の絶対値は大きくなりやすいが、有意になりにくいので注意。

比率の検定、信頼区間

TrueとFalseからなる母集団の値の標本がある。この標本データに基づき、母集団のTrueの本当の比率の信頼区間を求めたい

- prop.test(x, n, p)　#nが標本サイズ、xがTrueの個数
  - 95%信頼水準の信頼区間を求める

```
> prop.test(1e+09, 3e+09,0.333)
   0.3333 ~ 0.3334
> prop.test(1e+05, 3e+05,0.333)
   0.3316 ~ 0.3350

> prop.test(1e+05, 3e+09)
   3.313e-05 ~ 3.354e-05
> prop.test(10, 3e+05)
   1.694e-05 ~ 6.352e-05
```

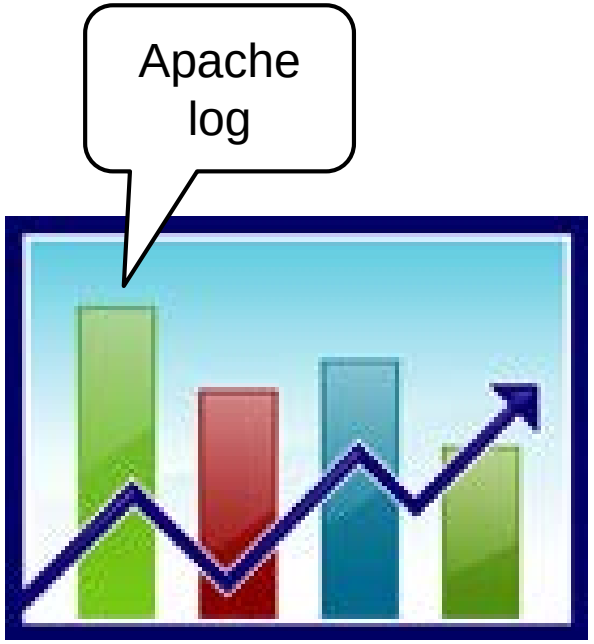
```
> prop.test(1e+9, 3e+9)

        1-sample proportions test with continuity correction

data:  1e+09 out of 3e+09, null probability 0.5
X-squared = 333333333, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3333165  0.3333502
```

アプリケーション例（比率の検定）

- ディスプレイ広告



ページビュー予測問題

広告商品（契約）の例

2012/4/1~4/15の間、ファイナンスのページを訪れた、
東京在住で不動産に興味ある30代の男性に100万回表示

ページビュー数予測

- 基本的には単純な時系列予測の問題だが・・・
  - 考慮すべき属性の組み合わせ数は膨大（数100億～兆！）
    - 性別
    - 年齢
    - 掲載場所 （サービスの種類） （ 10^3 ）
    - 地域 （ 10^3 ）
    - 興味カテゴリ（ 10^3 ）
    - ドメイン、時間帯、・・・
  - これら全てについてログをとり、予測モデルを構築・管理することは到底不可能

ページビュー数予測

- 実際のアプローチ
  - ユーザ全体の予測モデルを作る（SARIMAなど）
  - 全体の予測に、過去ログ中のユーザ分布の割合をかけて、属性ごとの予測値を得る

$$\hat{N}(\text{女性}, \text{化粧品}) = \hat{N}(\text{全体}) \times p(\text{女性}, \text{化粧品})$$

  - ログ中で、ある属性を持ったユーザの割合が重要

割合に基づくPV数予測

- 古典的実装: 組み合わせを無視
  - **割合を項目ごとにばらしてもっておく**

$$\hat{N}(\text{女性}, \text{化粧品}) = \hat{N}(\text{全体}) \times p(\text{女性}, \text{化粧品}) \cong \hat{N}(\text{全体}) \times \frac{N_{old}(\text{女性})}{N_{old}(\text{全体})} \times \frac{N_{old}(\text{化粧品})}{N_{old}(\text{全体})}$$

$$\frac{N_{old}(\text{女性}, \text{化粧品})}{N_{old}(\text{全体})}$$

本当はこちらを
使いたいが…

  - 組み合わせを考慮すると割合表の大きさが膨大になる

ログの数え上げ

- 毎日30億ページビュー
- 全部見る必要はなさそうだが、どれくらいサンプリングするべき？

- 直感的には
  - “男”, “女”など，大きい属性を見るには少しでよさそう
  - “男+30代+車+○○県○○町+…”
    小さい場合は大丈夫だろうか？

Sample-based approach

- Forecasting high-dimensional data [Agarwal et al., SIGMOD'10]
  - ランダムにログをサンプリングし、メモリ上に保持
    - 2000万サンプル、8GB
  - 問い合わせの際、条件を満たすログをon-siteに数え上げる
    - 数十msec
- Bitmap indexing https://sdm.lbl.gov/fastbit/

	1	2	3	4	5
男性	1	0	0	0	1
女性	0	0	1	1	0
東京	1	0	0	0	0
大阪	0	1	1	0	0
自動車	1	1	0	0	0
化粧品	0	0	1	1	0

count
"女性=1 AND 化粧品=1"

実験

- Yahoo! Inc. のログデータ
  - 一週間分のログで割合計算
  - 2000万サンプル、8GB
- 提案手法（ログカウント）が最もよい予測精度
- 全数に比べるとごくわずかのサンプルであるが、このビジネス要件の範囲内においては十分

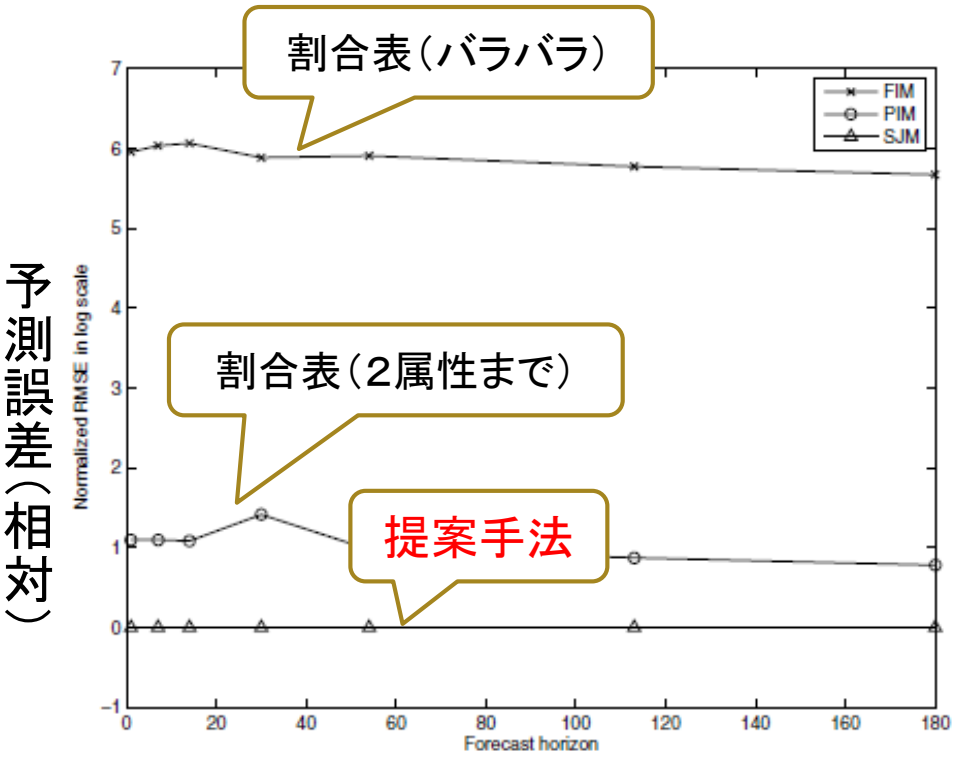


Figure 2: Count Forecast Accuracy: Varying Forecast Horizon (Y-axis shown in logscale, base e)

Rの検定いろいろ

- t.test　#平均値
- wilcox.text　#中央値
- prop.test　#比率
- shapiro.text　#正規性の検定
- cor.text　#相関
- chisq.test #カイ二乗検定
- pairwise.t.text　#グループ平均
- ks.test　#標本分布が同じかどうか

- 他多数

p値に関して

- 有意水準5%？
  - 別に根拠はないが、慣習的によく用いられている
  - 目的や前提によって水準は変えるべき
  - 分野によっても違う

- 生物、自然相手のサイエンスでは厳しくやらないと話にならない
  - ヒッグス粒子の存在は有意水準 0.00003% で検定

- Redefine statistical significance [Benjamin et al., 2017]
  - 72人の研究者が連名で、p値の敷居値を0.05から0.005にしようと提言
  - 社会科学、医療分野

p-hacking (what you must NOT do)

Six Ways to p-Hack

1. Stop collecting data once $p<.05$
2. Analyze many measures, but report only those with $p<.05$.
3. Collect and analyze many conditions, but only report those with $p<.05$.
4. Use covariates to get $p<.05$.
5. Exclude participants to get $p<.05$.
6. Transform the data to get $p<.05$.

8

Slides courtesy of Leif D. Nelson
(https://bitssblog.files.wordpress.com/2014/02/nelson-presentation.pdf)

米国統計学会の声明 (2016)

- 1. P-values can indicate how incompatible the data are with a specified statistical model.
- 2. P-values do not measure the probability that the studied hypothesis is true, or the probability that the data were produced by random chance alone.
- 3. Scientific conclusions and business or policy decisions should not be based only on whether a p-value passes a specific threshold.
- 4. Proper inference requires full reporting and transparency.
- 5. A p-value, or statistical significance, does not measure the size of an effect or the importance of a result.
- 6. By itself, a p-value does not provide a good measure of evidence regarding a model or hypothesis.

ビッグデータ

- データ量の重要さは目的によって変わる
  - とにかく全部使えばいい、というわけではない
    （ただし、一部を使う場合はとりかたに注意）
- 特定のアプリケーションに利用することが目的であれば、往々にして費用対効果が薄い場合が多い（と思う）
- データドリブンに知識発見をしたいのであれば、データは多いにこしたことはない
  - 10/1000 と 10万/1000万は全く違う！
  - 多ければ新しいものが見つかる（有意になる）場合が増える
- データは嘘をつかないが、人間は嘘をつく
  - 重要なのは再現性
  - p値が目的化するのは本末転倒

データ加工

- いろんなフォーマット
  - CSV, XML, HTML, JSON, SQL, NoSQL, …
- 多くの場合、実データは雑多
  - 形式の不整合、欠損値
  - 読み込むだけで一苦労
  - NumPyで扱える形にするまでが大変

pandas

- NumPyベースの高度なデータ操作ライブラリ

- よいところ（作者によれば）
  - 自動的なデータアラインメント
  - 数学的操作
  - 時系列/非時系列データの統合的なデータ構造
  - 欠損値の補完
  - 他のデータベースとの連携（SQLなど）

データフレーム

- Rのものと似ている（表形式データ）

```
In [3]: import pandas as pd   #pandas をインポート

In [4]: data = {'state' : ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
   ...: 'year' : [2000, 2001, 2002, 2001, 2002],
   ...: 'pop' : [1.5, 1.7, 3.6, 2.4, 2.9]}

In [5]: frame = pd.DataFrame(data)

In [6]: frame
Out[6]:
   pop   state  year
0  1.5    Ohio  2000
1  1.7    Ohio  2001
2  3.6    Ohio  2002
3  2.4  Nevada  2001
4  2.9  Nevada  2002
```

データフレーム

- つづき

```
In [10]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
   ...: index=['one', 'two', 'three', 'four', 'five'])

In [11]: frame2
Out[11]:
       year   state  pop debt
one    2000    Ohio  1.5  NaN
two    2001    Ohio  1.7  NaN
three  2002    Ohio  3.6  NaN
four   2001  Nevada  2.4  NaN
five   2002  Nevada  2.9  NaN

In [12]: frame2['debt'] = np.arange(5.)

In [13]: frame2
Out[13]:
       year   state  pop  debt
one    2000    Ohio  1.5     0
two    2001    Ohio  1.7     1
three  2002    Ohio  3.6     2
four   2001  Nevada  2.4     3
five   2002  Nevada  2.9     4
```

読み込み、書き出し

- read_csv, read_table, to_csv など

```
In [4]: !cat ex1.csv  #!でOSコマンド呼び出し
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo

In [5]: df = pd.read_csv('ex1.csv')
           #pd.read_table('ex1.csv',sep=',')
In [6]: df
Out[6]:
   a   b   c   d  message
0  1   2   3   4    hello
1  5   6   7   8    world
2  9  10  11  12      foo
```

```
In [9]: df.to_csv(sys.stdout)  #標準出力
,a,b,c,d,message
0,1,2,3,4,hello
1,5,6,7,8,world
2,9,10,11,12,foo

In [10]: df.to_csv('out.csv',
index=False,header=False)

In [11]: !cat out.csv
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

JSON, XML, SQLなどとの連携も楽
（本講義では省略）

基本操作

- SQLライクにいろいろ操作できる
  - Webドキュメント参照

```
In [21]: frame2['year']==2001
Out[21]:
one      False
two       True
three    False
four      True
five     False
Name: year, dtype: bool

In [22]: frame2[frame2['year']==2001]
Out[22]:
      year   state  pop  debt
two   2001    Ohio  1.7     1
four  2001  Nevada  2.4     3
```

```
In [33]: frame2.drop(['two','four'])
Out[33]:
      year   state  pop  debt
one    2000    Ohio  1.5     0
three  2002    Ohio  3.6     2
five   2002  Nevada  2.9     4

In [34]: frame2.drop(['pop'],axis=1)
Out[34]:
      year   state  debt
one    2000    Ohio     0
two    2001    Ohio     1
three  2002    Ohio     2
four   2001  Nevada     3
five   2002  Nevada     4

In [43]: frame3=frame2.drop(['two','four'])

In [44]: frame3
Out[44]:
      year   state  pop  debt
one    2000    Ohio  1.5     0
three  2002    Ohio  3.6     2
five   2002  Nevada  2.9     4

In [45]: frame4 =
frame3.reindex(['one','b','three','d','five'])

In [46]: frame4
Out[46]:
      year   state  pop  debt
one    2000    Ohio  1.5     0
b       NaN     NaN  NaN   NaN
three  2002    Ohio  3.6     2
d       NaN     NaN  NaN   NaN
five   2002  Nevada  2.9     4
```

算術演算・アラインメント

```
In [55]: df1 =
pd.DataFrame(np.arange(12.).reshape((3,4)),
columns=list('abcd'))

In [56]: df2 =
pd.DataFrame(np.arange(20.).reshape((4,5)),
columns=list('abcde'))
```

```
In [57]: df1
Out[57]:
   a  b   c   d
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

```
In [58]: df2
Out[58]:
    a   b   c   d   e
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19
```

```
In [59]: df1 + df2
Out[59]:
      a     b    c    d   e
0     0     2    4    6 NaN
1     9    11   13   15 NaN
2    18    20   22   24 NaN
3 NaN   NaN  NaN  NaN NaN
```

```
In [60]: df1.add(df2, fill_value=0)
Out[60]:
    a   b   c   d   e
0   0   2   4   6   4
1   9  11  13  15   9
2  18  20  22  24  14
3  15  16  17  18  19
```

```
In [61]: df1.reindex(columns=df2.columns, fill_value=0)
Out[61]:
   a  b   c   d  e
0  0  1   2   3  0
1  4  5   6   7  0
2  8  9  10  11  0
```

表の結合

- merge, join, concat など

```
In [12]: df1 = pd.DataFrame({'key':['b','b','a','c','a','a','b'],
    ...: 'data1':range(7)})

In [13]: df2 =
pd.DataFrame({'key':['a','b','d'],'data2':range(3)})

In [14]: df1
Out[14]:
   data1 key
0      0   b
1      1   b
2      2   a
3      3   c
4      4   a
5      5   a
6      6   b
```

```
In [15]: df2
Out[15]:
   data2 key
0      0   a
1      1   b
2      2   d

In [16]: pd.merge(df1, df2, on='key')
Out[16]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0

In [17]: pd.merge(df1, df2, how='outer')
Out[17]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
6      3   c    NaN
7    NaN   d      2
```

データの前処理

重複削除

```
In [39]: data = pd.DataFrame({'k1':['one']*3 +
['two']*4,
    ...: 'k2':[1,1,2,3,3,4,4]})
```

```
In [40]: data
Out[40]:
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
```

```
In [41]: data.duplicated()
Out[41]:
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

```
In [42]: data.drop_duplicates()
Out[42]:
    k1  k2
0  one   1
2  one   2
3  two   3
5  two   4
```

値の置き換え

```
In [44]: data = pd.DataFrame([1., -999., 2., -999., -1000., 3.])

In [45]: data
Out[45]:
      0
0     1
1  -999
2     2
3  -999
4 -1000
5     3

In [46]: data.replace(-999, np.nan)
Out[46]:
      0
0     1
1   NaN
2     2
3   NaN
4 -1000
5     3

In [47]: data.replace({-999:np.nan, -1000:0})
Out[47]:                                  #辞書形式で指定
    0
0   1
1 NaN
2   2
3 NaN
4   0
5   3
```

欠損値の扱い

- 除去する

```
In [51]: from numpy import nan as NA

In [52]: data = pd.DataFrame([[1., 6.5, 3.],
[1., NA, NA], [NA, NA, NA], [NA, 6.5, 3.]])

In [53]: data
Out[53]:
     0    1    2
0    1  6.5    3
1    1  NaN NaN
2 NaN  NaN NaN
3 NaN  6.5    3

In [54]: data.dropna()
Out[54]:
   0    1   2
0  1  6.5  3

In [55]: data.dropna(how='all')
Out[55]:
     0    1    2
0    1  6.5    3
1    1  NaN NaN
3 NaN  6.5    3
```

- 適当に埋める

```
In [63]: df = pd.DataFrame(np.random.randn(7,3))
In [64]: df.loc[:3, 1] = NA;
In [65]: df.loc[:1, 2] = NA;

In [66]: df
Out[66]:
          0         1         2
0 -1.251673       NaN       NaN
1 -0.853750       NaN       NaN
2  0.127473       NaN -0.116891
3 -0.770677       NaN  0.069410
4  0.016002  1.198004 -1.238413
5 -0.408303 -0.269995 -0.440169
6 -0.642193 -0.841632  0.381958

In [69]: df.fillna(0)
Out[69]:
          0         1         2
0 -1.251673  0.000000  0.000000
1 -0.853750  0.000000  0.000000
2  0.127473  0.000000 -0.116891
3 -0.770677  0.000000  0.069410
4  0.016002  1.198004 -1.238413
5 -0.408303 -0.269995 -0.440169
6 -0.642193 -0.841632  0.381958

In [70]: df.fillna({1:0, 2:0.5})
Out[70]:
          0         1         2
0 -1.251673  0.000000  0.500000
1 -0.853750  0.000000  0.500000
2  0.127473  0.000000 -0.116891
3 -0.770677  0.000000  0.069410
4  0.016002  1.198004 -1.238413
5 -0.408303 -0.269995 -0.440169
6 -0.642193 -0.841632  0.381958
```

欠損値の扱い

- 補完する

```
In [66]: df
Out[66]:
          0         1         2
0 -1.251673       NaN       NaN
1 -0.853750       NaN       NaN
2  0.127473       NaN -0.116891
3 -0.770677       NaN  0.069410
4  0.016002  1.198004 -1.238413
5 -0.408303 -0.269995 -0.440169
6 -0.642193 -0.841632  0.381958
```

```
In [77]: df.mean()
Out[77]:
0   -0.540446
1    0.028792
2   -0.268821
dtype: float64
```

```
In [78] df.fillna(df.mean())
Out[78]:
          0         1         2
0 -1.251673  0.028792 -0.268821
1 -0.853750  0.028792 -0.268821
2  0.127473  0.028792 -0.116891
3 -0.770677  0.028792  0.069410
4  0.016002  1.198004 -1.238413
5 -0.408303 -0.269995 -0.440169
6 -0.642193 -0.841632  0.381958
```

外れ値の検出・除去

```
In [79]: np.random.seed(12345)  #乱数系列の指定
In [80]: data = pd.DataFrame(np.random.randn(1000,4))
In [81]: data.describe()
Out[81]:
                0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean     -0.067684     0.067924     0.025598    -0.002298
std       0.998035     0.992106     1.006835     0.996794
min      -3.428254    -3.548824    -3.184377    -3.745356
25%      -0.774890    -0.591841    -0.641675    -0.644144
50%      -0.116401     0.101143     0.002073    -0.013611
75%       0.616366     0.780282     0.680391     0.654328
max       3.366626     2.653656     3.260383     3.927528

In [82]: col = data[3]  #第４列
In [83]: col[np.abs(col) > 3]   #絶対値が3より大を探す
Out[83]:
97     3.927528
305   -3.399312
400   -3.745356
Name: 3, dtype: float64
```

```
In [85]: data[(np.abs(data)>3).any(1)]  #いずれかの一つ以上の要素
Out[85]:
          0         1         2         3
5   -0.539741  0.476985  3.248944 -1.021228
97  -0.774363  0.552936  0.106061  3.927528
102 -0.655054 -0.565230  3.176873  0.959533
305 -2.315555  0.457246 -0.025907 -3.399312
324  0.050188  1.951312  3.260383  0.963301
400  0.146326  0.508391 -0.196713 -3.745356
499 -0.293333 -0.242459 -3.056990  1.918403
523 -3.428254 -0.296336 -0.439938 -0.867165
586  0.275144  1.179227 -3.184377  1.369891
808 -0.362528 -3.548824  1.553205 -2.186301
900  3.366626 -2.372214  0.851010  1.332846

In [86]: data[np.abs(data)>3] = np.sign(data)*3  #符号の配列

In [87]: data.describe()
Out[87]:
                0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean     -0.067623     0.068473     0.025153    -0.002081
std       0.995485     0.990253     1.003977     0.989736
min      -3.000000    -3.000000    -3.000000    -3.000000
25%      -0.774890    -0.591841    -0.641675    -0.644144
50%      -0.116401     0.101143     0.002073    -0.013611
75%       0.616366     0.780282     0.680391     0.654328
max       3.000000     2.653656     3.000000     3.000000
```

ダミー変数

- カテゴリ情報を数値的に扱うための変数（0 or 1）

```
In [113]: df = pd.DataFrame({'class':['b','b','a','c','a','b'],
     ...: 'data1':range(6)})

In [114]: df
Out[114]:
  class  data1
0     b      0
1     b      1
2     a      2
3     c      3
4     a      4
5     b      5

In [115]: pd.get_dummies(df['class'])
Out[115]:
   a  b  c
0  0  1  0
1  0  1  0
2  1  0  0
3  0  0  1
4  1  0  0
5  0  1  0

In [116]: dummies = pd.get_dummies(df['class'])

In [117]: df_with_dummy = df[['data1']].join(dummies)

In [118]: df_with_dummy
Out[118]:
   data1  a  b  c
0      0  0  1  0
1      1  0  1  0
2      2  1  0  0
3      3  0  0  1
4      4  1  0  0
5      5  0  1  0
```

例)

- タイタニックの生存者予測
  - 入門者向けの定番データの一つ

```
In [14] data = pd.read_csv('titanic.csv')

In [15]: data.describe()
Out[15]:
       PassengerId    Survived      Pclass         Age       SibSp       Parch        Fare
count   891.000000  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std     257.353842    0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min       1.000000    0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%     446.000000    0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%     668.500000    1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000    6.000000  512.329200

In [22]: data2 = data[np.isnan(data['Age'])]

n [25]: data2[:3]
Out[25]:
    PassengerId  Survived  Pclass                            Name     Sex  Age    (略)
5             6         0       3                Moran, Mr. James    male  NaN
17           18         1       2    Williams, Mr. Charles Eugene    male  NaN
19           20         1       3         Masselmani, Mrs. Fatima  female  NaN
```

本日の内容

- 統計基礎
  - 検定、推定
- データ加工、操作
- **多変量解析**

本題に入る前に

- データの種類にはいろいろあり、尺度を意識することが重要

データの種類	尺度の種類	尺度の意味	可能な計算	例
量的データ	比尺度	原点（0という値）と比率に意味がある	＋、－、×、÷	身長、体重、金額
	間隔尺度	値の間隔に意味がある	＋、－	知能指数
質的データ	順序尺度	順序に意味がある	度数，最頻値，中央値	マラソンの順位
	名義尺度	区別するだけ	度数，最頻値	性別、血液型

例えば…

- ５段階評価のアンケート

  (1)悪い (2)やや悪い (3)ふつう (4)良い (5)とても良い

  - 順序尺度。平均に意味はあるか？
    - 正しくデータを表す代表値となるかは不明

  - カテゴリをつけず“5点満点”なら比尺度？

多変量解析とは

- 大規模、高次元なデータから本質的な情報（できれば低次元）を抽出するための統計的手法群の総称
  - 目的変数がない場合

説明変数	手法
量的データ(比尺度)	主成分分析、因子分析
量的データ(間隔尺度)	クラスター分析、多次元尺度構成法、数量化Ⅳ類
質的データ	数量化Ⅲ類、対応分析

  - 目的変数がある場合

目的変数	説明変数	手法
量的データ	量的データ	回帰分析、正準相関分析
	質的データ	数量化Ⅰ類
質的データ	量的データ	判別分析
	質的データ	数量化Ⅱ類

ダミー変数

ダミー変数

多変量解析による次元圧縮

- 生データは一般に極めて高次元
  - 例） 文書、画像　数十万~数百万次元
  - 次元の呪い：適切な学習が難しくなる
  - 人間にとっても意味が掴みにくい（可視化できない）

- 実際のデータは冗長であり、本質的に重要な構造は低次元で表現できる（場合が多い）

主成分分析：Principal Component Analysis（PCA）

- p次元の特徴ベクトル $\mathbf{x} = (x_1, x_2, \ldots, x_p)^T$ を、元のデータの構造をできるだけ保ったまま低次元へ圧縮したい

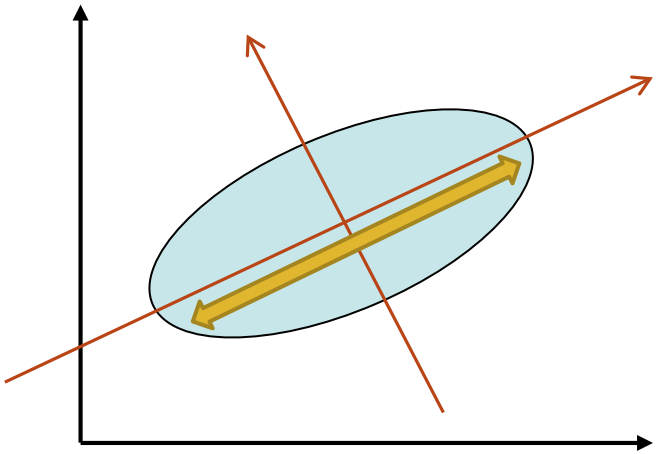
  線形射影： $z = a_1 x_1 + a_2 x_2 + \cdots + a_p x_p = \mathbf{a}^T \mathbf{x}$ （ただし $\mathbf{a}^T \mathbf{a} = 1$ ）

- データの分布を最もよく記述する軸は？

  ⇒ 分散最大基準

$$\mathrm{var}(z) = \frac{1}{n} \sum_{i=1}^{n} (z_i - \bar{z})^2$$

を最大化する $\mathbf{a}$ を求めたい



PCA： 分散最大基準による導出

$$\mathrm{var}(z) = \frac{1}{n}\sum_{i=1}^{n}(z_i - \bar{z})^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{a}^T\mathbf{x}_i - \mathbf{a}^T\bar{\mathbf{x}}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{a}^T\mathbf{x}_i - \mathbf{a}^T\bar{\mathbf{x}}\right)\left(\mathbf{a}^T\mathbf{x}_i - \mathbf{a}^T\bar{\mathbf{x}}\right)^T$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{a}^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\mathbf{a}$$

$$= \mathbf{a}^T\left(\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\right)\mathbf{a}$$

$$= \mathbf{a}^T C_X \mathbf{a}$$

x の共分散行列

$J_{PCA} = \mathbf{a}^T C_X \mathbf{a}$ を

$\mathbf{a}^T\mathbf{a} = 1$ のもとで最大化

$$J'_{PCA} = \mathbf{a}^T C_X \mathbf{a} - \lambda(\mathbf{a}^T\mathbf{a} - 1)$$ を最大化

（ラグランジュの未定乗数法）

$$\frac{\partial J'_{PCA}}{\partial \mathbf{a}} = 2C_X\mathbf{a} - 2\lambda\mathbf{a} = 0$$ （停留点）

$$\Rightarrow \boxed{C_X\mathbf{a} = \lambda\mathbf{a}}$$

※行列の微分についてはmatrix cookbook等を参照
http://orion.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

PCA： 平均二乗誤差最小基準による導出

主成分空間に射影した点の元の空間における座標は

$$\hat{\mathbf{x}}_i = z_1\mathbf{a}_1 + z_2\mathbf{a}_2 + \cdots + z_m\mathbf{a}_m = \sum_{j=1}^{m} \mathbf{a}_j^T\mathbf{x}_i\mathbf{a}_j$$

$$\varepsilon^2(\mathbf{a}) = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i - \hat{\mathbf{x}}_i\right)^2$$

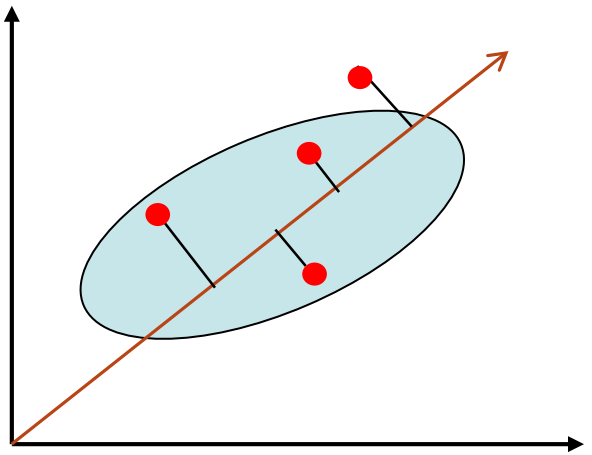
$$= \frac{1}{n}\sum_{i=1}^{n}\left\|\mathbf{x}_i - \sum_{j=1}^{m}\mathbf{a}_j^T\mathbf{x}_i\mathbf{a}_j\right\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i\|^2 - \sum_{j=1}^{m}\mathbf{a}_j^T\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{a}_j$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i\|^2 - \sum_{j=1}^{m}\mathbf{a}_j^T R_X\mathbf{a}_j$$

定数

結局こちらを最大化



自己相関行列 $R_X = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T$

の固有値問題に帰着

$$R_X\mathbf{a} = \lambda\mathbf{a}$$

PCA：つづき

- 複数の無相関な軸が、固有値に対応する固有ベクトルとして得られる
  - 固有値の大きさがその軸（固有ベクトル）におけるデータの分散の大きさに対応

- 累積寄与率を参考に主成分（固有ベクトル）の数を決める
  - i番目の主成分の寄与率： $\lambda_i \Big/ \sum_{j=1}^{p} \lambda_j$
  - m番目の主成分までの累積寄与率： $\sum_{j=1}^{m} \lambda_j \Big/ \sum_{j=1}^{p} \lambda_j$

（固有値は降順にならんでいるものとする）

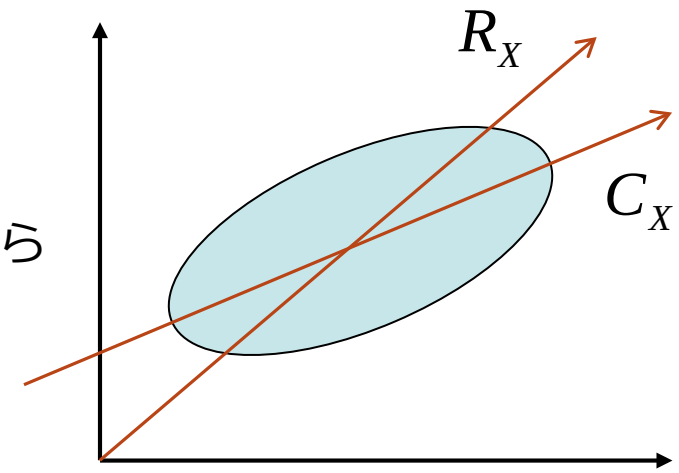
注意

- 共分散行列、自己相関行列、相関係数行列と、それぞれの固有値問題で張られる部分空間の違いに注意

- 自己相関行列（相関係数行列ではない！）

$$R_X = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T = C_X + \overline{\mathbf{x}}\overline{\mathbf{x}}^T$$

  - 二乗誤差基準で導出した場合は、一般にはこちら
  - 座標原点を中心に分散を見た場合に相当
  - 特徴に非負制約がある場合に有効



- 最初にデータから平均を引いておけば分かりやすい（一致する）
  - ただし、いつもそれが適切とは限らない

（参考） 相関係数行列

- 元データの各特徴を平均0、分散1に正規化したあとの共分散行列に等しい