

データサイエンス 第3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

- R入門
 - 統計基礎
 - 検定、推定
 - データ加工、操作
-

R言語

- 1996年, Ross Ihaka & Robert Gentleman
- 統計解析、データマイニングに適した環境
- ベクトル処理言語
- 高速な組みこみ関数群
 - 更に、C, C++, Fortranなどの外部プログラムと動的リンク可能
- 優れたグラフ機能
- オープンソース, CRANネットワーク
 - 最新の研究成果も比較的すぐ実装されたりする

SAS, SPSS, S言語等の商用解析ツールを代替

R言語：欠点

- 大規模データが処理できない
 - 基本的にオンメモリ前提
- 並列化の実装が難しい

(ただし、拡張ツールなどはいろいろある)

- アルゴリズムの中身をいじるには向いていない

基本操作

- 起動、終了

```
$ R  
> q()  #終了
```

- オブジェクトへ代入

```
> x<-1  
> x #オブジェクトの中身を表示  
[1] 1  
> x<-c(1,2,3,4,5,6)  
[1] 1 2 3 4 5 6
```

- ベクトル演算

```
> x^2  
[1] 1 4 9 16 25 36
```

- 関数の例

```
> mean(x)  
[1] 3.5  
> round(mean(x),0)  
[1] 4
```

データ型

- マトリックス (数値のみ)

```
> mat<-matrix(c(1,2,0,3), 2, 2) #2行2列の行列
> mat
  [,1] [,2]
[1,]  1  0
[2,]  2  3
> class(mat) #型
[1] "matrix"
```

- データフレーム (表みたいなもの)

```
> df1<-data.frame(cbind(LETTERS[1:4],3:0)) #アルファベット、数値をバインド
> colnames(df1)<-c("X","Y") #列名をつける
> df1
  X Y
1 A 3
2 B 2
3 C 1
4 D 0
> class(df1)
[1] "data.frame"
```

グラフプロット

```
> iris #フィッシャーのアヤメのデータ (データフレーム型)
```

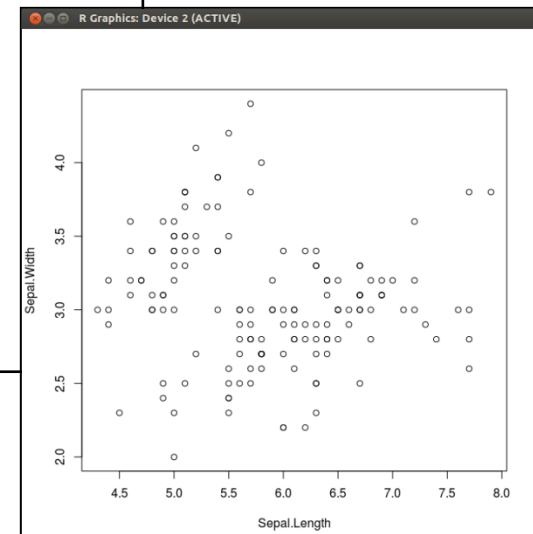
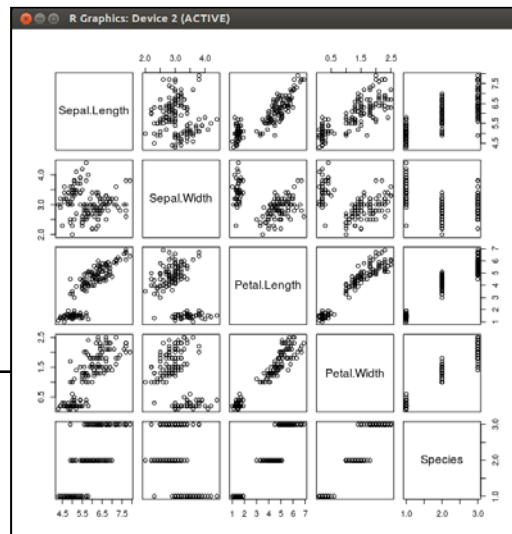
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
--	--------------	-------------	--------------	-------------	---------

1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

(省略)

```
> plot(iris)
```

```
> plot(iris[,1:2])
```



パッケージ

- さまざまなツールをまとめたもの
- 必要に応じて読み込んで使う

```
> library() #インストール済みのパッケージを表示
Packages in library '/usr/lib/R/library':
base          The R Base Package
boot          Bootstrap Functions (originally by Angelo Canty for S)
class         Functions for Classification
(省略)

> search() #読み込み済みのパッケージを表示
[1] ".GlobalEnv"      "package:stats"    "package:graphics"
[4] "package:grDevices" "package:utils"    "package:datasets"
[7] "package:methods" "Autoloads"        "package:base"
```

- 例

```
> library(MASS) #MASSライブラリを読み込む
> help(package="MASS") #MASS
> truehist(c(1:10))
```

パッケージ

- ないものはインストール
 - rootユーザでないと、インストール先はユーザのホームディレクトリ以下になる

```
> install.packages("randomForest")
> library("randomForest")
> randomForest(formula=Species~., data=iris)
```

Call:

```
randomForest(formula = Species ~ ., data = iris)
```

 Type of random forest: classification

 Number of trees: 500

No. of variables tried at each split: 2

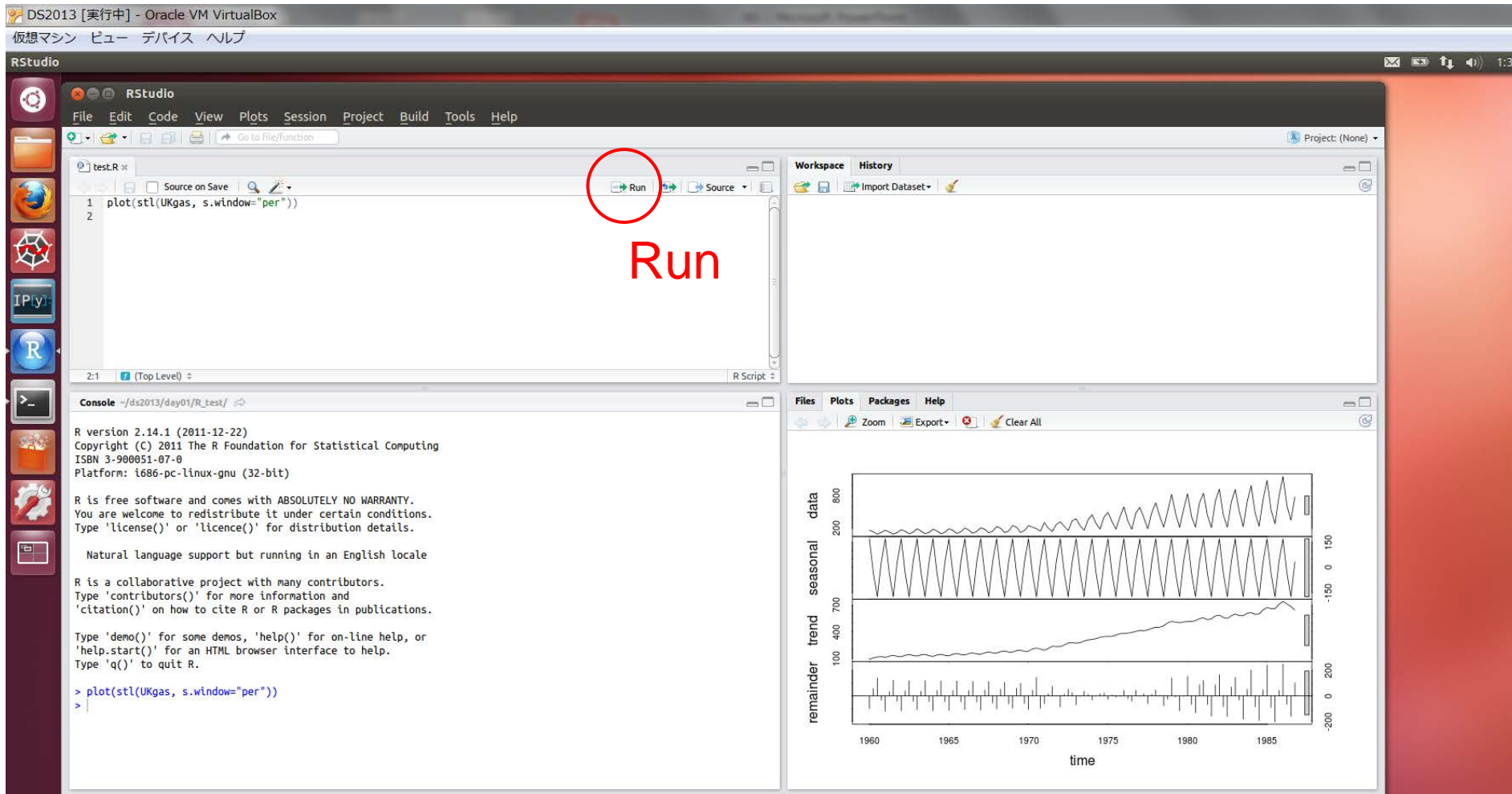
 OOB estimate of error rate: 4.67%

Confusion matrix:

	setosa	versicolor	virginica	class.error
setosa	50	0	0	0.00
versicolor	0	47	3	0.06
virginica	0	4	46	0.08

Rstudio

- 統合開発環境
「Run」で一行ずつ実行



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概念が重要

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〇〇と△△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強力な証拠になる
=帰無仮説を棄却する。

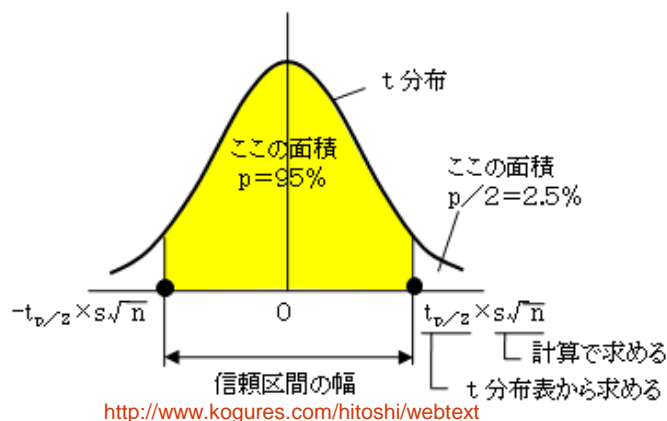
平均値の検定 (t検定)

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



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

μ_0, s_0 標本の平均、標準偏差
 n 標本サイズ

- 標本の大きさ n が小さい場合 ($n < 30$)、母集団が正規分布に従っている必要がある

平均値の検定 (t検定)

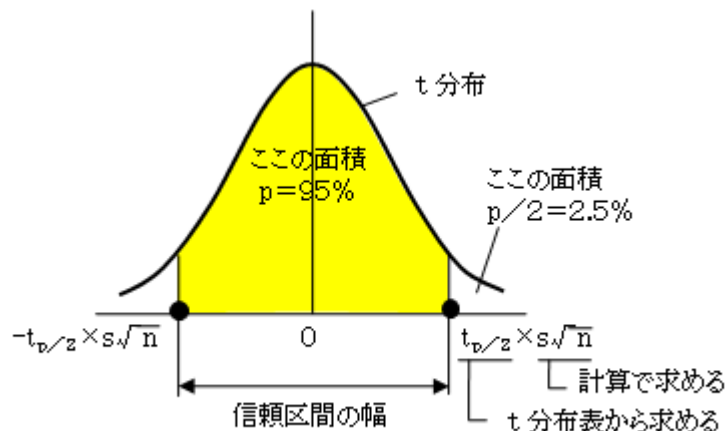
- t.test 関数 (R)

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 であることに不利な証拠を示していない (有意水準5%)

> 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
```

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： 世の中の人たち？（と主張するには無理があるが…）

標本： ラボのみんな

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

相関の有意性の検定

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)
```

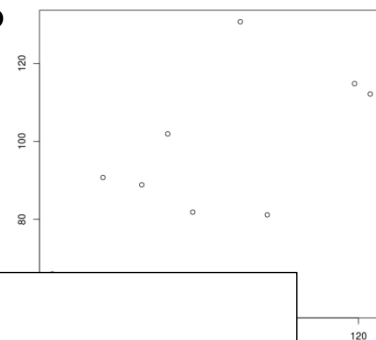
```
> y<-x+rnorm(50, sd=15) # xに正規分布によるノイズを加える
```

```
> 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
```

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。



```
> 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

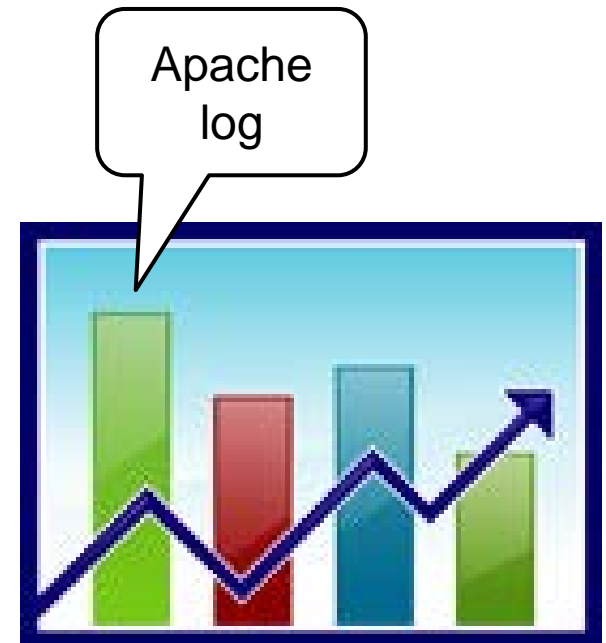
sample estimates:

cor

0.5710318

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

割合の予測

- ログ中で、ある属性を持ったユーザの割合が重要
 - 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、・・・
-

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+〇〇県〇〇町+…”
小さい場合は大丈夫だろうか？

比率の信頼区間を求める

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)` #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)
0.3333 ~ 0.3334
```

```
> prop.test(1e+05, 3e+05)
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

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

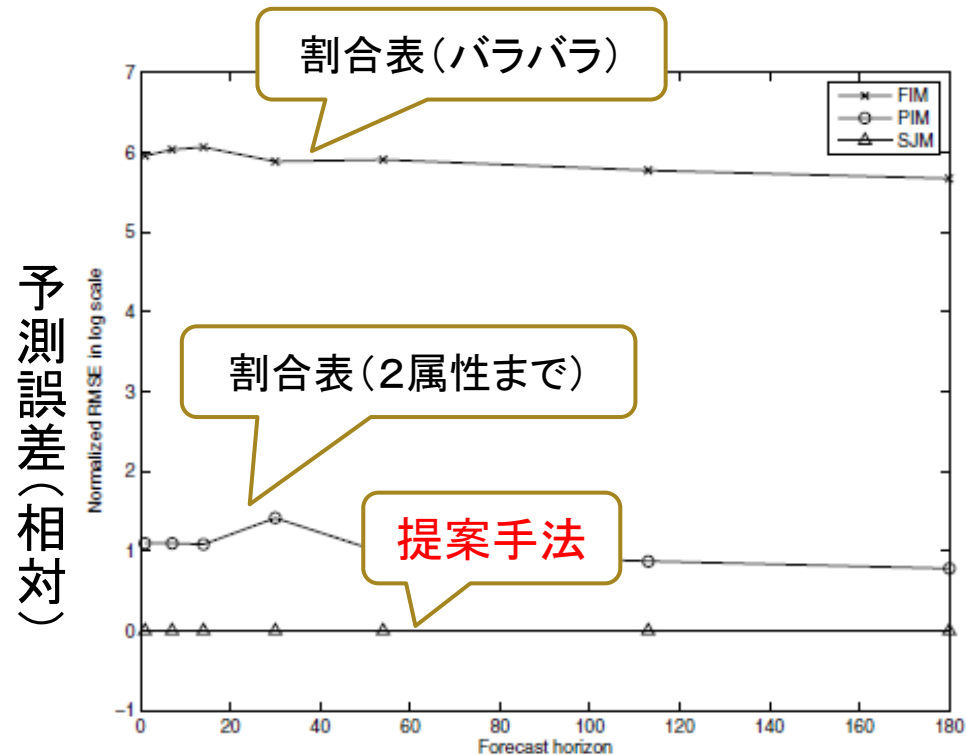


Figure 2: Count Forecast Accuracy: Varying Forecast Horizon (Y-axis shown in logscale, base e)

Rの検定いろいろ

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

- 他多数

ちなみに…

- 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
- CS分野では甘くても許される場合が多い
(個人的な印象。UI系などはしっかりやっているはず)
 - ベンチマークのスコア (e.g. 3回のランダム試行における識別正解率) が1%あがりました！とか。。
 - 多分突っ込んではいけない… (たまに偉い人が突っ込む)
 - 有意水準数10%の業界？ (スコアはあくまで飾り)
- 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

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) 可能性が増える

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 [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

```
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

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
```

データの事前処理

● 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	1
1	-999
2	2
3	-999
4	-1000
5	3

```
In [46]: data.replace(-999, np.nan)
```

```
Out[46]:
```

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	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.ix[:3, 1] = NA;
```

```
In [65]: df.ix[: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	