インターネット計測とデータ解析 第 7 回

長 健二朗

2014 年 5 月 26 日

前回のおさらい

第 6 回 相関

- オンラインお勧めシステム
- 距離と類似度
- 相関係数
- 演習: 相関

今日のテーマ

第 7 回 多変量解析

- データセンシング
- 地理的位置情報 (geo-location)
- 線形回帰
- 主成分分析
- 演習: 線形回帰

多変量データ解析

- 一変数解析 (univariate analysis)
  - 変数をひとつずつ独立して扱う
- 多変量解析 (multivariate analysis)
  - 複数の変数を同時に扱う

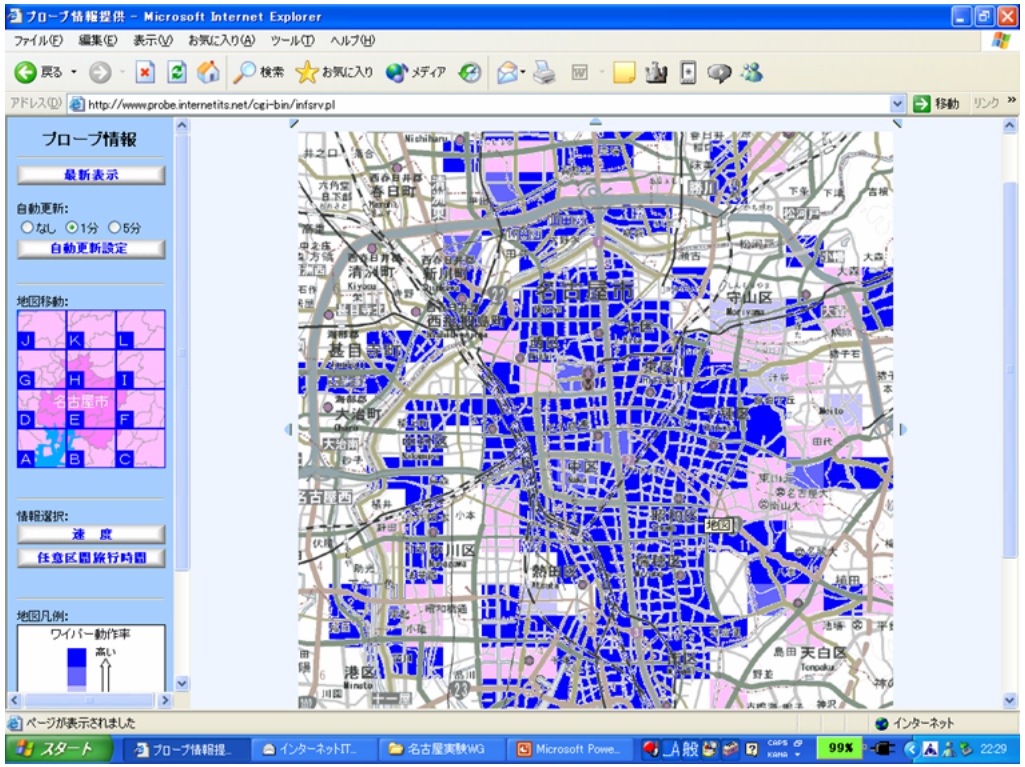
  - コンピュータの普及で発展
  - 隠れたトレンドを探る (データマイニング)

データセンシング

- データセンシング: 遠隔からデータを収集する
- インターネット経由でさまざまなセンサー情報が取得可能に
  - 気象情報、電力消費、その他さまざまな情報

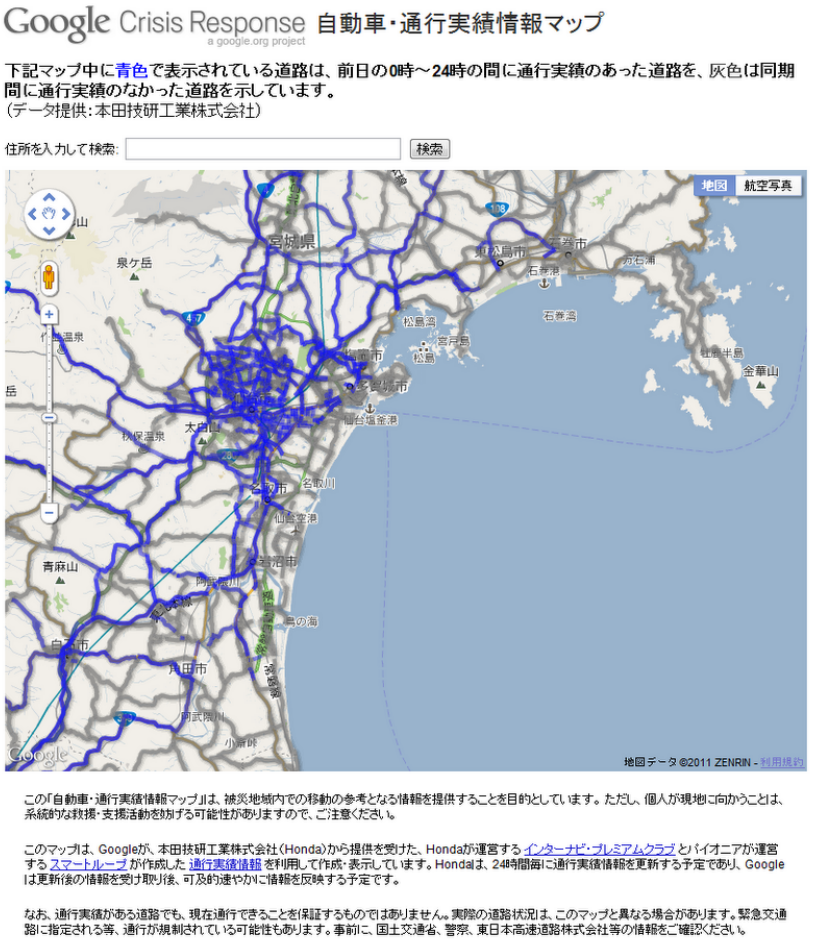
例: 自動車のワイパー情報

- WIDE プロジェクトが 2001 年に名古屋で行ったインターネット自動車実験
- 1570 台のタクシーから位置、速度、ワイパー稼働情報を収集
- 図の青い部分がワイパー動作率が高い地域で、細かな降雨状況が分かる



東日本大震災での活用

- 前述のシステムは ITS の一部として利用中
- 地震の 3 日後に利用可能な道路情報が公開される
  - ホンダ (トヨタ, 日産) によるデータ提供



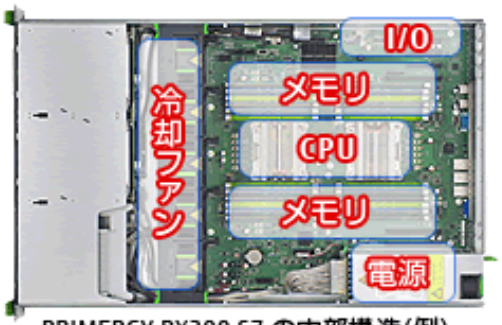
source: google crisis response

省エネルギー技術

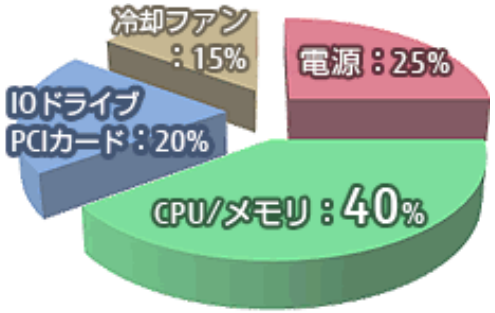
- 電力消費の削減: あらゆる技術分野で課題
  - センサー情報を用いた賢い制御で効率化
- 個別機器の効率化から全体の効率化へ
  - 例: PC サーバの効率化とデータセンタの効率化

PC サーバの効率化

- PC 内のセンサー情報を使った効率化制御
  - 温度、電圧、消費電力、ファン回転数
- PC サーバの電力消費内訳
  - CPU/メモリ: 40%
    - 高集積化・省電力化、クロック・電圧制御
  - 電源: 25%
    - ロス削減 (AC-DC, DC-DC)
  - IO: 20%
    - 省電力機能、省電力ディスク/SSD
  - 冷却ファン: 15%
    - 効率配置、エアフロー設計、制御最適化



PRIMERGY RX300 S7 の内部構造(例)

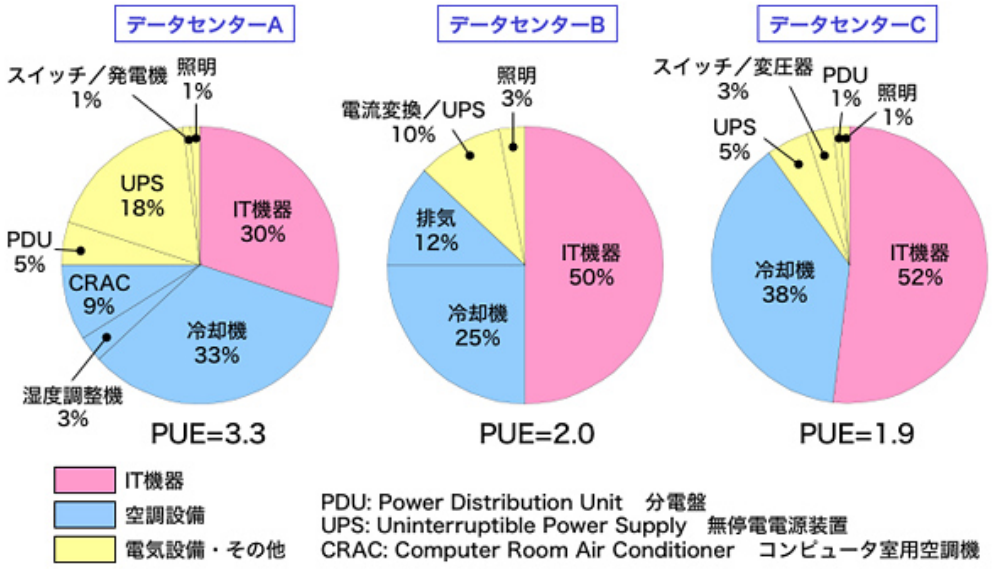


サーバ内の電力消費の割合の例

source: http://jp.fujitsu.com/platform/server/primergy/solution/power-saving/option/

データセンタの効率化

- データセンタ需要増加に伴う電力消費の急増
  - 空調電力や電力ロスが大きい
- IT 機器: 使用機器の効率化、高温仕様機器の利用
- 空調設備: 仕様見直し、エアフロー・熱負荷設計、空調器の効率化、外気冷却
- 給電設備: ロス削減、高圧給電、DC 給電、UPS 効率化、自然エネルギー利用
- 全体設計: 全体効率化、柔軟に消費電力変動に追従、入退室削減、アイドル機器の停止などの制御



source:

http://librarytaisei.jp/feature/datacenter/002/

位置情報サービス

- 場所に応じた情報の提供
- 地図サービス、ナビゲーション、時刻表
- 近隣のレストランや店舗検索 (広告への利用)
- その他、さまざまなサービスの可能性

例: 駅.Locky

- 名古屋大学 河口研が開発した時刻表サービス
  - WiFi 情報収集プロジェクトから派生した人気アプリ
- iPhone/Android 用 App
- 位置情報から最寄りの駅の時刻表を検索
  - GPS/WiFi による位置情報取得
  - 同時に、端末から見える WiFi 基地局情報を収集
- 次の出発までの時間をカウントダウン
  - 時刻表の閲覧も可能
- ユーザ提供型の時刻表データベース

GPS (Global Positioning System)

- 約 30 個の GPS 衛星
- 元来はアメリカ合州国の軍事用
  - 当初は意図的に誤差データを加え 100m 程度の精度にしていた
  - 2000 年に誤差混入が廃止され、10m 程度の精度になる
- さまざまな民生用途
  - カーナビ、携帯端末、デジカメ
- 測位: 3 個の GPS 衛星からの距離から位置を特定
  - GPS 信号には衛星の位置、時刻情報が含まれる
  - 距離は GPS 衛星からの時刻データのずれから計算
  - 受信機の時刻補正のため 4 個の衛星を捕捉する必要
  - より多くの衛星を捕捉すれば精度が向上
- 欠点
  - 衛星が見えないと使えない
  - 初期位置取得時間
- 高精度化: 加速度センサーや振動型ジャイロスコープと組合せ

基地局を利用した位置情報

- 端末は接続している基地局が分かる
  - 基地局側からも接続している端末が分かる
  - 接続していなくても電波を受信できる基地局が分かる
- 基地局がその緯度経度を発信するサービスも存在
- 屋内でも利用可能
  - 他にも、音波、可視光などによるアプローチも存在
- GPS との組合せによる精度向上

インターネットの特徴量

通信レベルの特徴量

- 回線容量、スループット
- 遅延
- ジッタ
- パケットロス

測定手法

- アクティブ計測: ping 等、計測パケットを注入
- パッシブ計測: 計測用パケットを使わない
  - 2 点で観測して比較
  - TCP の挙動等から推測
  - トランスポート機能内部で情報収集

遅延

- 遅延成分
  - 遅延 = 伝搬遅延 + キュー待ち遅延 + その他
  - パケット毎に一定の遅延成分とパケット長に比例する成分
  - 輻輳がなければ、遅延は伝搬遅延 + α
- 遅延計測
  - RTT(round trip time) 計測: パケットの往復時間
  - 一方向遅延計測: 両端の時刻同期が必要

  - 遅延の平均
  - 最大遅延: 例えば、一般に音声会話は 400ms 以下が必要
  - ジッタ: 遅延値のばらつき
    - リアルタイム通信でのバッファサイズの決定
    - 下位層の影響: 無線での再送、イーサネットのコリジョン等

代表的な遅延値

- パケット伝送時間 (ワイヤースピード)
  - 1500 bytes at 10Mbps: 1.2 msec
  - 1500 bytes at 100Mbps: 120 usec
  - 1500 bytes at 1Gbps: 12 usec
- ファイバー中の伝搬速度: 約 200,000 km/s
  - 100km round-trip: 1 msec
  - 20,000km round-trip: 200 msec
- 衛星の RTT
  - LEO (Low-Earth Orbit): 200 msec
  - GEO (Geostationary Orbit): 600 msec

パケットロス

パケットロス率

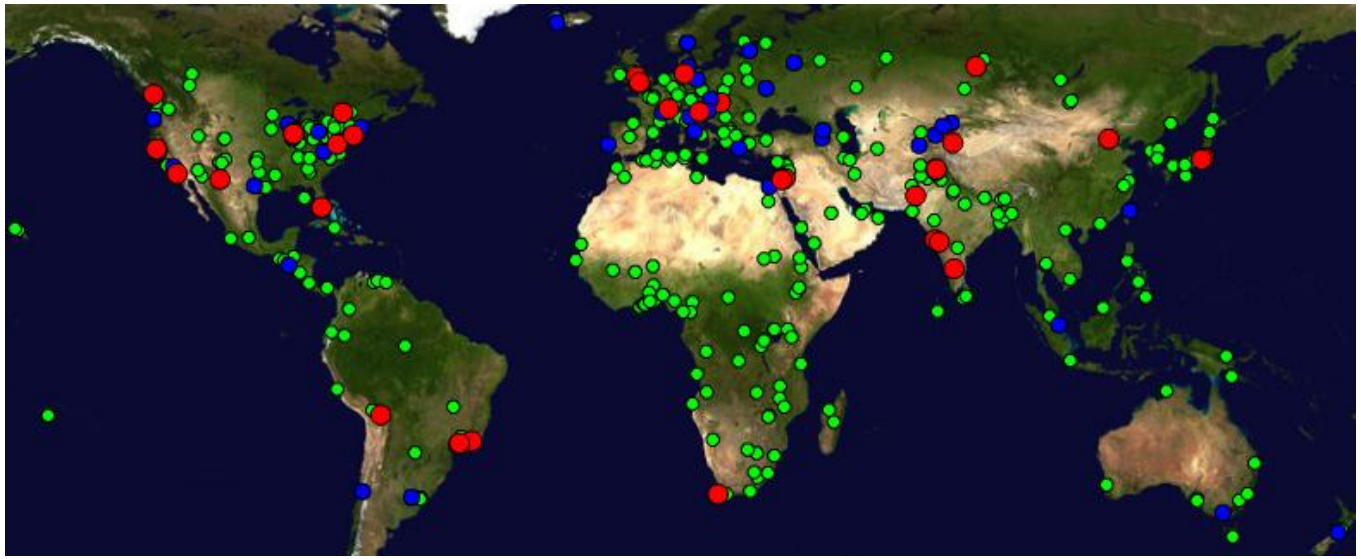
- パケットロスがランダムに発生すると見なせればロス率だけでいいが
- 一定間隔のプローブでは分からない傾向
  - バースト的なロス: バッファ溢れ等
  - パケット長による違い: 無線でのビット誤り等

pingER project

- the Internet End-to-end Performance Measurement (IEPM) project by SLAC
- using ping to measure rtt and packet loss around the world
  - http://www-iepm.slac.stanford.edu/pinger/
  - started in 1995
  - over 600 sites in over 125 countries

pingER project monitoring sites

- monitoring (red), beacon (blue), remote (green) sites
  - beacon sites are monitored by all monitors



from pingER web site

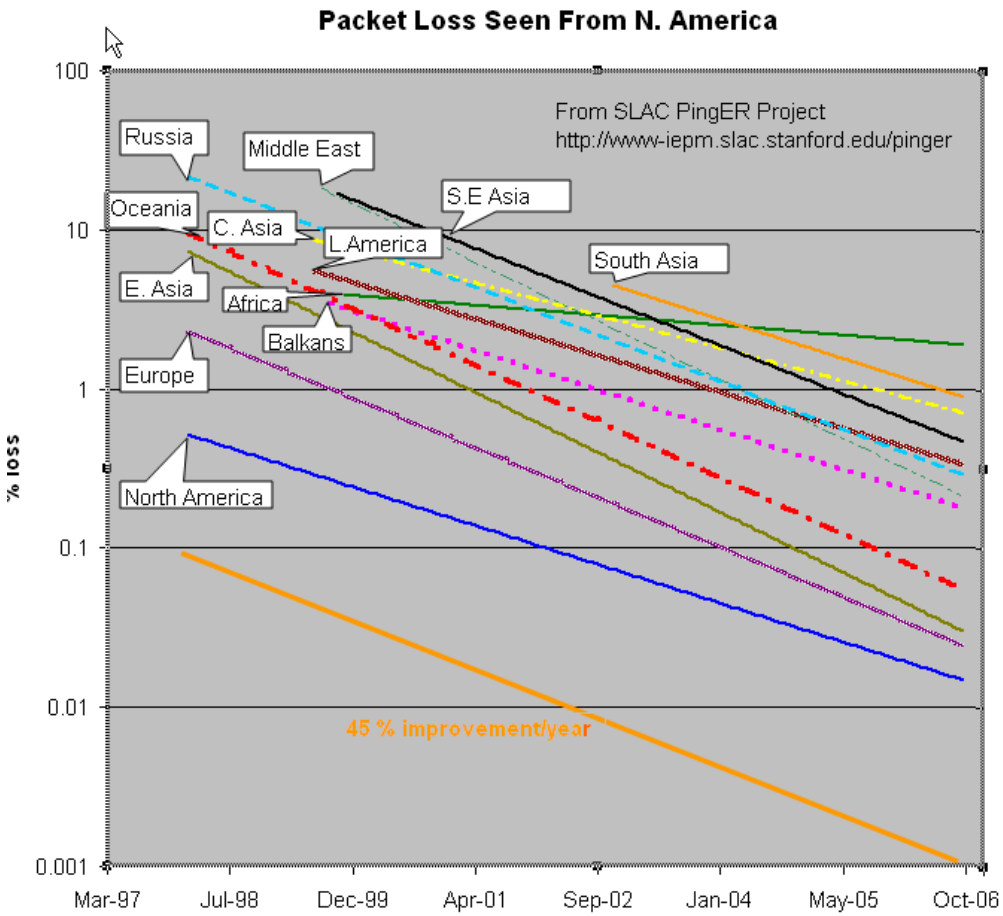
pingER project monitoring sites in east asia

- monitoring (red) and remote (green) sites

from pingER web site

pingER packet loss

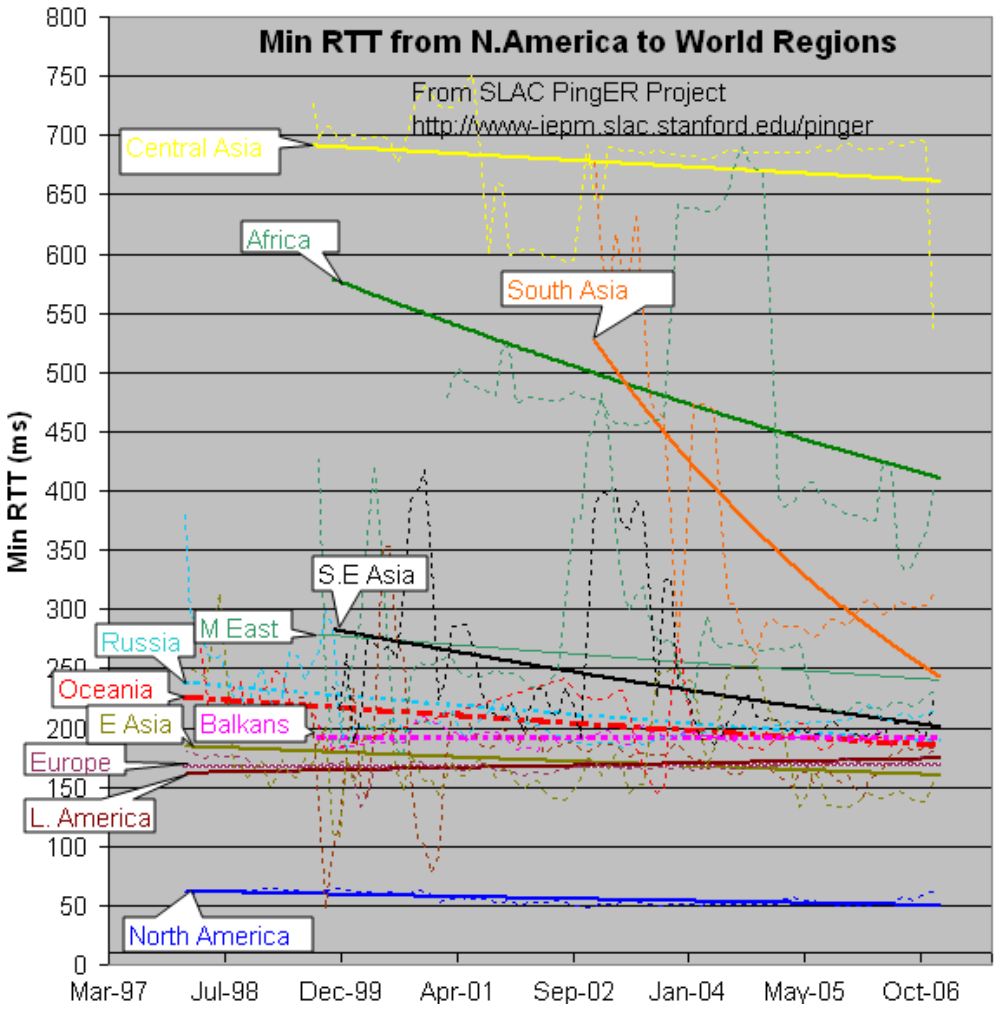
- packet loss observed from N. Ameria
- exponential improvement in 10 years



from pingER web site

pinger minimum rtt

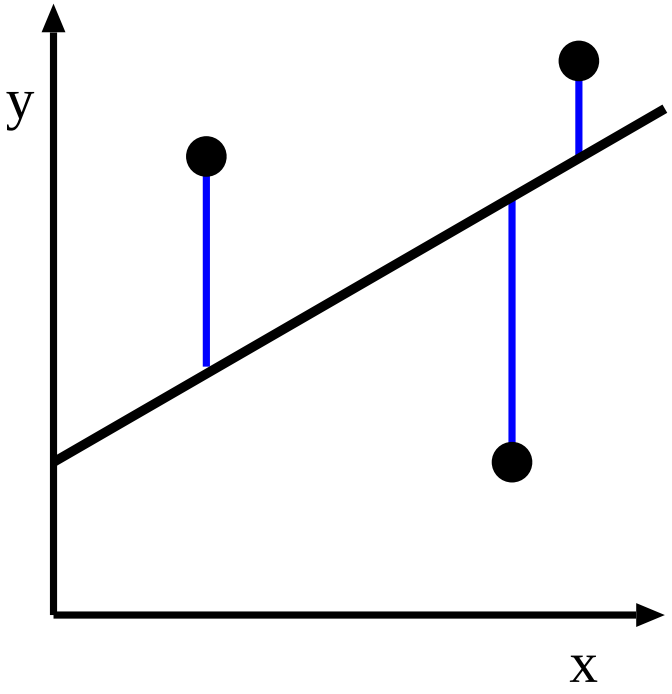
- minimum rtts observed from N. America
- gradual shift from satellite to fiber in S. Asia and Africa

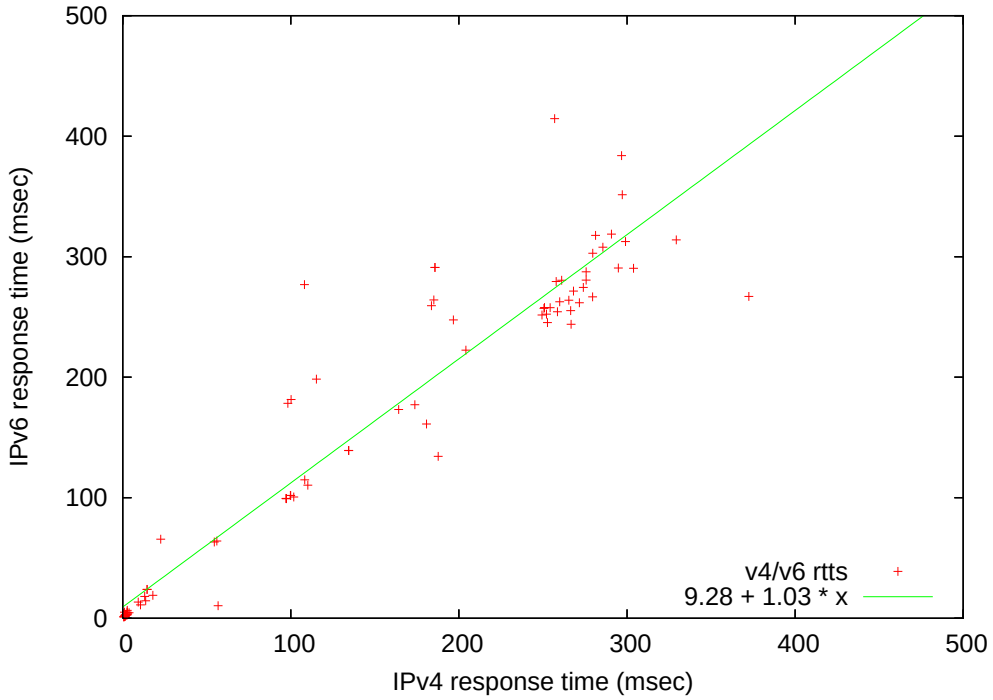


from pingER web site

線形回帰 (linear regression)

- データに一次関数を当てはめる
  - 最小二乗法 (least square method): 誤差の二乗和を最小にする





最小二乗法 (least square method)

誤差の二乗和を最小にする一次関数を求める

$$f(x) = b_0 + b_1 x$$

切片と傾きの求め方

$$b_1 = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

ここで

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\sum xy = \sum_{i=1}^{n} x_i y_i \qquad \sum x^2 = \sum_{i=1}^{n} (x_i)^2$$

最小二乗法の導出

i 番目の変数の誤差 $e_i = y_i - (b_0 + b_1 x_i)$、$n$ 回の観測における誤差の平均は

$$\bar{e} = \frac{1}{n}\sum_i e_i = \frac{1}{n}\sum_i (y_i - (b_0 + b_1 x_i)) = \bar{y} - b_0 - b_1\bar{x}$$

誤差平均が 0 になるようにすると $b_0 = \bar{y} - b_1\bar{x}$
b_0 を b_1 で表現すると $e_i = y_i - \bar{y} + b_1\bar{x} - b_1 x_i = (y_i - \bar{y}) - b_1(x_i - \bar{x})$
誤差の二乗和 SSE は

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}[(y_i - \bar{y})^2 - 2b_1(y_i - \bar{y})(x_i - \bar{x}) + b_1^2(x_i - \bar{x})^2]$$

分散に書き直す

$$\begin{aligned}\frac{SSE}{n} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 - 2b_1\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x}) + b_1^2\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\ &= \sigma_y^2 - 2b_1\sigma_{xy}^2 + b_1^2\sigma_x^2\end{aligned}$$

SSE を最小にする b_1 は、この式を b_1 の 2 次式とみて b_1 について微分して 0 と置く

$$\frac{1}{n}\frac{d(SSE)}{db_1} = -2\sigma_{xy}^2 + 2b_1\sigma_x^2 = 0$$

すなわち $b_1 = \frac{\sigma_{xy}^2}{\sigma_x^2} = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$

主成分分析 (principal component analysis; PCA)

主成分分析の目的

- 複数の変数間の関係を、少数の互いに独立な合成変数 (成分) で近似

共分散行列の固有値問題として解ける

主成分分析の応用

- 次元減少
  - 寄与率の大きい順に主成分を取る、寄与率の小さい成分は無視できる
- 主成分のラベル付け
  - 主成分の構成要素から、その意味を読みとる

注意点

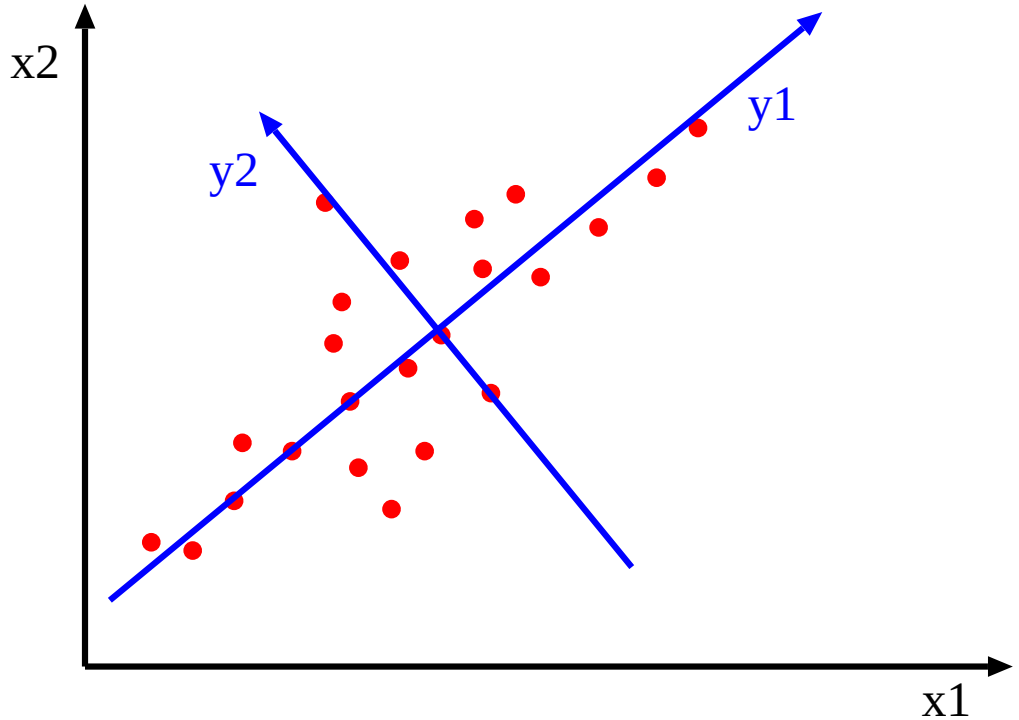
- あくまで、ばらつきの大きい成分を抜き出すだけ
  - とくに各軸の単位が違う場合は注意
- 機械的に複雑な関係を分析できる便利な手法であるが、それで複雑な関係が説明できる訳ではない

主成分分析の直観的な説明

座標変換の観点から 2 次元の図で説明すると

- データのばらつきが最も大きい方向に重心を通る直線 (第 1 主成分軸) を引く
- 第 1 主成分軸に直交し、次にばらつきが大きい方向に第 2 主成分軸を引く
- 同様に第 3 主成分軸以降を引く

例えば、「身長」と「体重」を「体の大きさ」と「太り具合」に変換。「座高」や「胸囲」など変数が増えても同様



主成分分析 (おまけ)

主成分の単位ベクトルは、共分散行列の固有ベクトルとして求まる
X を d 次の変数、これを主成分 Y に変換する $d \times d$ の直交行列 P を求める

$$Y = P^\top X$$

これを $cov(Y)$ は対角行列 (各変数が独立)、かつ P は直交行列 ($P^{-1} = P^\top$) という制約のもとで解く
Y の共分散行列は

$$\begin{aligned} cov(Y) &= E[YY^\top] = E[(P^\top X)(P^\top X)^\top] = E[(P^\top X)(X^\top P)] \\ &= P^\top E[XX^\top]P = P^\top cov(X)P \end{aligned}$$

したがって

$$Pcov(Y) = PP^\top cov(X)P = cov(X)P$$

P を $d \times 1$ 行列でかくと、

$$P = [P_1, P_2, \ldots, P_d]$$

また、$cov(Y)$ は対角行列 (各変数が独立) なので

$$cov(Y) = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_d \end{bmatrix}$$

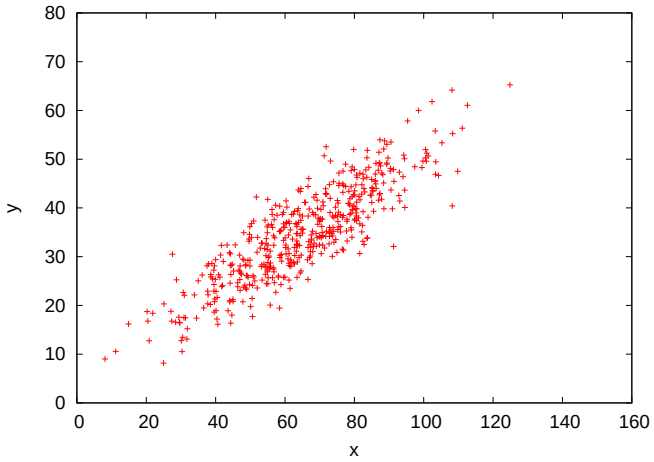
書き直すと

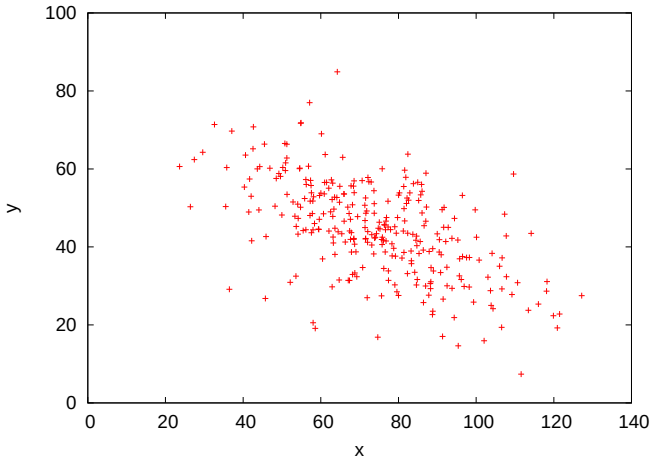
$$[\lambda_1 P_1, \lambda_2 P_2, \ldots, \lambda_d P_d] = [cov(X)P_1, cov(X)P_2, \ldots, cov(X)P_d]$$

$\lambda_i P_i = cov(X)P_i$ において、P_i は X の共分散行列の固有ベクトルであることが分かる
したがって、固有ベクトルを見つければ求めていた変換行列 P が得られる

前回の演習: 相関係数の計算

- データの相関係数を計算する
  - correlation-data-1.txt, correlation-data-2.txt





data-1:r=0.87 (left), data-2:r=-0.60 (right)

前回の演習: 相関係数の計算スクリプト

```
#!/usr/bin/env ruby

# regular expression for matching 2 floating numbers
re = /([-+]?\d+(?:\.\d+)?)\s+([-+]?\d+(?:\.\d+)?)/

sum_x = 0.0 # sum of x
sum_y = 0.0 # sum of y
sum_xx = 0.0 # sum of x^2
sum_yy = 0.0 # sum of y^2
sum_xy = 0.0 # sum of xy
cc = 0.0 # correlation coefficient
n = 0 # the number of data

ARGF.each_line do |line|
    if re.match(line)
      x = $1.to_f
      y = $2.to_f
      sum_x += x
      sum_y += y
      sum_xx += x**2
      sum_yy += y**2
      sum_xy += x * y
      n += 1
    end
end

denom = (sum_xx - sum_x**2 / n) * (sum_yy - sum_y**2 / n)
if denom != 0.0
  cc = (sum_xy - sum_x * sum_y / n) / Math.sqrt(denom)
end

printf "n:%d r:%.3f\n", n, cc
```

前回の演習 2: 類似度計算

- データの類似度を計算する
  - 「集合知プログラミング」2 章の参考データ
  - 7 人のユーザの映画評価スコア: scores.txt

```
% cat scores.txt
# A dictionary of movie critics and their ratings of a small set of movies
'Lisa Rose': 'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5, 'Just My Luck': 3.0, 'Superman Returns':
'Gene Seymour': 'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5, 'Just My Luck': 1.5, 'Superman Returns
'Michael Phillips': 'Lady in the Water': 2.5, 'Snakes on a Plane': 3.0, 'Superman Returns': 3.5, 'The Nigh
'Claudia Puig': 'Snakes on a Plane': 3.5, 'Just My Luck': 3.0, 'The Night Listener': 4.5, 'Superman Return
'Mick LaSalle': 'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0, 'Just My Luck': 2.0, 'Superman Returns
'Jack Matthews': 'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0, 'The Night Listener': 3.0, 'Superman
'Toby': 'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman Returns':4.0
```

スコアデータ

- 簡単な例題: データが少なすぎる
- 一覧にすると以下のようになる

```
#name: 'Lady in the Water' 'Snakes on a Plane' 'Just My Luck' 'Superman Returns
Lisa Rose:          2.5 3.5 3.0 3.5 3.0
Gene Seymour:       3.0 3.5 1.5 5.0 3.0
Michael Phillips:   2.5 3.0  -  3.5 4.0
Claudia Puig:        -  3.5 3.0 4.0 4.5
Mick LaSalle:       3.0 4.0 2.0 3.0 3.0
Jack Matthews:      3.0 4.0  -  5.0 3.0
Toby:                -  4.5  -  4.0  -
```

類似度計算の実行

- コサイン類似度を使ってユーザ間の類似度行列を作る

```
% ruby similarity.rb scores.txt
Lisa Rose:        1.000 0.959 0.890 0.921 0.982 0.895 0.708
Gene Seymour:     0.959 1.000 0.950 0.874 0.962 0.979 0.783
Michael Phillips: 0.890 0.950 1.000 0.850 0.929 0.967 0.693
Claudia Puig:     0.921 0.874 0.850 1.000 0.875 0.816 0.695
Mick LaSalle:     0.982 0.962 0.929 0.875 1.000 0.931 0.727
Jack Matthews:    0.895 0.979 0.967 0.816 0.931 1.000 0.822
Toby:             0.708 0.783 0.693 0.695 0.727 0.822 1.000
```

類似度計算スクリプト (1/2)

```
# regular expression to read data
# 'name': 'title0': score0, 'title1': score1, ...
re = /'(.+?)':\s+(\S.*)/
name2uid = Hash.new # keeps track of name to uid mapping
title2tid = Hash.new # keeps track of title to tid mapping
scores = Hash.new # scores[uid][tid]: score of title_id by user_id

# read data into scores[uid][tid]
ARGF.each_line do |line|
  if re.match(line)
    name = $1
    ratings = $2.split(",")

    if name2uid.has_key?(name)
      uid = name2uid[name]
    else
      uid = name2uid.length
      name2uid[name] = uid
      scores[uid] = {} # create empty hash for title and score pairs
    end
    ratings.each do |rating|
      if rating.match(/'(.+?)':\s*(\d\.\d)/)
        title = $1
        score = $2.to_f
        if title2tid.has_key?(title)
          tid = title2tid[title]
        else
          tid = title2tid.length
          title2tid[title] = tid
        end
        scores[uid][tid] = score
      end
    end
  end
end
```

類似度計算スクリプト (2/2)

```
# compute cosine similarity between 2 users
def comp_similarity(h1, h2)
  sum_xx = 0.0 # sum of x^2
  sum_yy = 0.0 # sum of y^2
  sum_xy = 0.0 # sum of xy
  score = 0.0 # similarity score

  h1.each do |tid, score|
    sum_xx += score**2
    if h2.has_key?(tid)
      sum_xy += score * h2[tid]
    end
  end
  h2.each_value do |score|
    sum_yy += score**2
  end
  denom = Math.sqrt(sum_xx) * Math.sqrt(sum_yy)
  if denom != 0.0
    score = sum_xy / denom
  end
  return score
end

# create n x n matrix of similarities between users
n = name2uid.length
similarities = Array.new(n) { Array.new(n) }
for i in 0 .. n - 1
  printf "%-18s", name2uid.key(i) + ':'
  for j in 0 .. n - 1
    similarities[i][j] = comp_similarity(scores[i], scores[j])
    printf "%.3f ", similarities[i][j]
  end
  print "\n"
end
```

より本格的なデータセット

- MovieLens: http://grouplens.org/datasets/movielens/
  - ミネソタ大学が公開している協調フィルタリング用データセット
  - ユーザの映画評価: 100K, 1M, 10M のスコアデータ
    - u.data: スコアデータ
    - 他にも各ユーザの属性情報や各映画の属性情報も含まれている

```
% head u.data
#user_id item_id rating timestamp
196      242     3       881250949
186      302     3       891717742
22       377     1       878887116
244      51      2       880606923
166      346     1       886397596
298      474     4       884182806
115      265     2       881171488
253      465     5       891628467
305      451     3       886324817
6        86      3       883603013
...
```

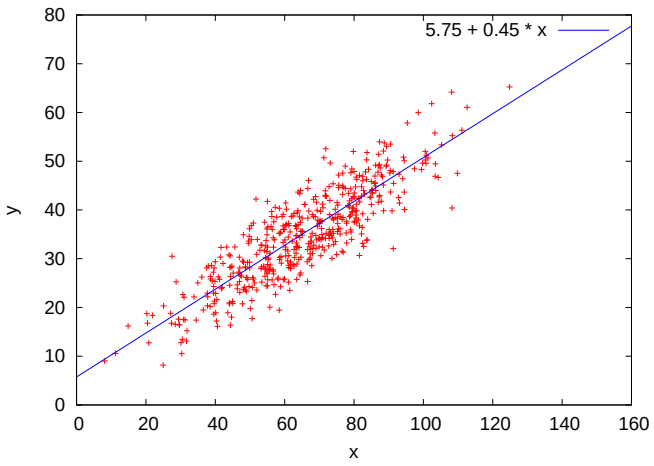
今回の演習: 線形回帰の計算

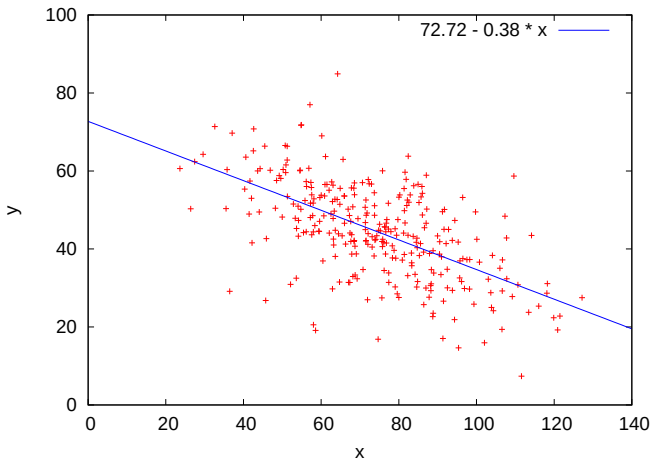
- 前回のデータを使い回帰直線を計算する
  - correlation-data-1.txt, correlation-data-2.txt

$$f(x) = b_0 + b_1 x$$

$$b_1 = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$





data-1:r=0.87 (left), data-2:r=-0.60 (right)

演習: 回帰直線の計算スクリプト

```
#!/usr/bin/env ruby

# regular expression for matching 2 floating numbers
re = /([-+]?\d+(?:\.\d+)?)\s+([-+]?\d+(?:\.\d+)?)/

sum_x = sum_y = sum_xx = sum_xy = 0.0
n = 0
ARGF.each_line do |line|
    if re.match(line)
      x = $1.to_f
      y = $2.to_f

      sum_x += x
      sum_y += y
      sum_xx += x**2
      sum_xy += x * y
      n += 1
    end
end

mean_x = Float(sum_x) / n
mean_y = Float(sum_y) / n
b1 = (sum_xy - n * mean_x * mean_y) / (sum_xx - n * mean_x**2)
b0 = mean_y - b1 * mean_x

printf "b0:%.3f b1:%.3f\n", b0, b1
```

演習: 散布図に回帰直線を加える

```
set xrange [0:160]
set yrange [0:80]

set xlabel "x"
set ylabel "y"

plot "correlation-data-1.txt" notitle with points, \
5.75 + 0.45 * x lt 3
```

まとめ

第 7 回 多変量解析

- データセンシング
- 地理的位置情報 (geo-location)
- 線形回帰
- 主成分分析
- 演習: 線形回帰

次回予定

第 8 回 時系列データ (6/2)

- インターネットと時刻
- ネットワークタイムプロトコル
- 時系列解析
- 演習: 時系列解析
- 課題 **2**