

インターネット計測とデータ解析 第6回

長 健二郎

2014年5月19日

前回のおさらい

第 5 回 多様性と複雑さ (5/12)

- ▶ ロングテール
- ▶ Web アクセスとコンテンツ分布
- ▶ べき乗則と複雑系
- ▶ 演習: べき乗則解析

今日のテーマ

第6回 相関

- ▶ オンラインお勧めシステム
- ▶ 距離とエントロピー
- ▶ 相関係数
- ▶ 演習: 相関

オンラインお勧めシステム

- ▶ EC サイトにおけるロングテールのユーザの潜在ニーズ
 - ▶ ユーザの嗜好に合った商品を提示して購買に繋げる
- ▶ レコメンダーパッケージによる導入コスト低下で普及



source: <http://longtail.com/>

お勧めシステムの技術

- ▶ ユーザの行動を観察して有用な情報を予測して自動的に提示
- ▶ EC サイト: 購買履歴や閲覧履歴からお勧め商品を自動的に提示
- ▶ EC サイトだけでなく検索予測、かな漢字変換などへの応用も

データベースの作り方

- ▶ アイテムベース: アイテムごとに情報をまとめる
- ▶ ユーザベース: ユーザごとに情報をまとめる
- ▶ 実際にはこれらを組み合わせて使う

お勧めシステムの予測手法

- ▶ コンテンツベース:
 - ▶ ユーザが過去に利用したアイテムから類似アイテムを推薦
 - ▶ アイテムの属性分類
 - ▶ 機械学習クラスタリングによるグループ化
 - ▶ ノウハウのルール化
 - ▶ 比較的狭い範囲での推薦になりがち、意外性が低い
- ▶ 協調フィルタリング: amazonをはじめ広く利用されている
 - ▶ 購買履歴から顧客間の類似度を計算
 - ▶ 類似したユーザの実績から共通度の高いアイテムを推薦
 - ▶ 特徴: 個別のアイテムに関する情報は使わない
 - ▶ 思いがけない発見 (serendipity) の可能性
- ▶ 単純ベイズ分類器: スпам判定と同じ原理
 - ▶ アイテムやユーザに関する個別の多様な情報から確率計算、機械学習する

最近のターゲティング広告の進化

- ▶ ターゲティング広告
 - ▶ 特定ジャンルに興味を持つユーザに絞った広告
 - ▶ 広告効果や費用対効果の向上
- ▶ アドネットワーク
 - ▶ ネット広告枠と広告主を仲介するネット広告配信サービス
 - ▶ 例: 個人が運営するサイトにバナー広告を入れる
- ▶ Real Time Bidding
 - ▶ ネット広告枠をリアルタイムでオークションする仕組み
 - ▶ 広告枠を提供するオークション主催者
 - ▶ ユーザの属性情報、行動履歴情報など (cookie による追跡)
 - ▶ 広告枠を入札するオークション参加者
 - ▶ 提供された情報を元に入札価格を決める
 - ▶ リターゲティング: 過去に自社のサイトを訪問したユーザに対する広告
 - ▶ RTB 用のプラットフォーム: ミリ秒オーダーの落札を実現

協調フィルタリング (collaborative filtering)

- ▶ 複数のアルゴリズムが存在
- ▶ シンプルなユーザ間相関分析
 - ▶ ユーザ間の相関をとり類似ユーザを抽出
 - ▶ 類似ユーザの類似度を重みに各アイテムの合計点数を計算

例: ユーザの購買履歴

user	item						
	a	b	c	d	e	f	...
A	1		1		1		...
B			1	1			...
C	1	1					...
D	1		1		1		...
...							...

A と相関高いユーザから A の持っていないアイテムのスコアを計算

user	similarity	item						
	σ	a	b	c	d	e	f	...
A	1	1		1		1		...
S	0.88		0.88		-		0.88	...
C	0.81		0.81		-		-	...
K	0.75		-		-		-	...
F	0.73		0.73		0.73		0.73	...
score			2.50		0.73		1.61	...

例: Netflix Prize

- ▶ 米国のオンライン DVD レンタルサービス Netflix のアルゴリズムコンテスト
- ▶ 同社のオンラインお薦めシステムの性能を 10%向上すれば 100 万ドルの賞金
- ▶ コンテスト用データセット:
 - < *user_id, movie_id, date_of_grade, grade* >
 - ▶ トレーニング用データセット: 1 億件の評価スコア
 - ▶ 評価用データセット: 280 万件の評価スコア
 - ▶ 答え合わせ用データセット: 140 万件
 - ▶ 採点用データセット: 140 万件
 - ▶ 採点スコアは結果の誤差の平均二乗偏差 (10%改善目標)
- ▶ コンテストは 2006 年に始まり、2009 年に終了
 - ▶ プライバシー問題で批判
 - ▶ 匿名化されたユーザを他の映画評価サイトのユーザとマッチング可能

距離について

いろいろな距離

- ▶ ユークリッド距離 (Euclidean distance)
- ▶ 標準化ユークリッド距離 (standardized Euclidean distance)
- ▶ ミンコフスキー距離 (Minkowski distance)
- ▶ マハラノビス距離 (Mahalanobis distance)

類似度

- ▶ バイナリベクトルの類似度
- ▶ n 次元ベクトルの類似度

距離の性質

空間上の2点 (x, y) 間の距離 $d(x, y)$:

非負性 (positivity)

$$d(x, y) \geq 0$$

$$d(x, y) = 0 \Leftrightarrow x = y$$

対称性 (symmetry)

$$d(x, y) = d(y, x)$$

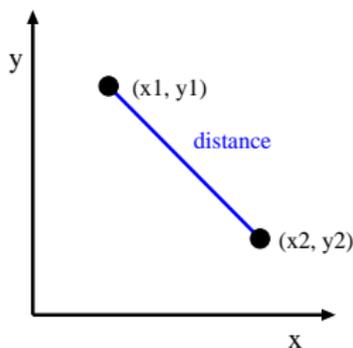
三角不等式 (triangle inequality)

$$d(x, z) \leq d(x, y) + d(y, z)$$

ユークリッド距離 (Euclidean distance)

普通に距離といえばユークリッド距離を指す
n次元空間での2点 (x, y) の距離

$$d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}$$



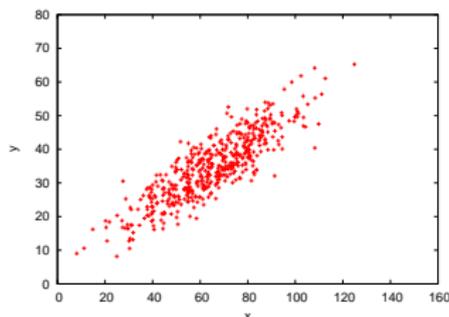
euclidean distance in 2-dimensional space

標準ユークリッド距離

(standardized Euclidean distance)

- ▶ 変数間でばらつきが大きさが異なると、距離が影響を受ける
- ▶ そこで、ユークリッド距離を各変数の分散で割って正規化

$$d(x, y) = \sqrt{\sum_{k=1}^n \left(\frac{x_k}{s_k} - \frac{y_k}{s_k} \right)^2} = \sqrt{\sum_{k=1}^n \frac{(x_k - y_k)^2}{s_k^2}}$$



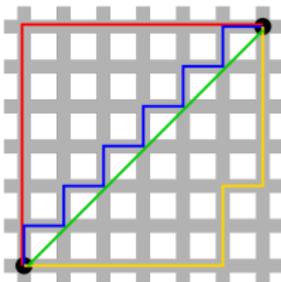
ミンコフスキー距離 (Minkowski distance)

ユークリッド距離を一般化

- ▶ パラメータ r が大きいほど、次元軸にとらわれない移動 (斜め方向のショートカット) を重視する距離

$$d(x, y) = \left(\sum_{k=1}^n |x_k - y_k|^r \right)^{\frac{1}{r}}$$

- ▶ $r = 1$: マンハッタン距離
 - ▶ ハミング距離: 2つの文字列間の同じ位置の文字の不一致数
 - ▶ 例えば、111111 と 101010 のハミング距離は 3
- ▶ $r = 2$: ユークリッド距離



Manhattan distance vs. Euclidean distance

ベクトルのノルム (vector norm) (1/2)

ベクトルのノルム: ベクトルの長さ

$\|x\|$ where x is a vector

ベクトル x の l_n -ノルムはミンコフスキー距離で定義される

$$\|x\|_n = \sqrt[n]{\sum_i |x_i|^n}$$

l_0 -norm: ベクトルの 0 でない要素の数

$$\|x\|_0 = \#(i|x_i \neq 0)$$

l_1 -norm: 差分の和

$$\|x\|_1 = \sum_i |x_i|$$

l_2 -norm: ユークリッド距離

$$\|x\|_2 = \sqrt{\sum_i |x_i|^2}$$

l_∞ -norm: ベクトルの最大要素

$$\|x\|_\infty = \max(|x_i|)$$

ベクトルのノルム (vector norm) (2/2)

例: ベクトル $x = (1, 2, 3)$ に対して

$$\|x\|_0 = 3 = 3.000$$

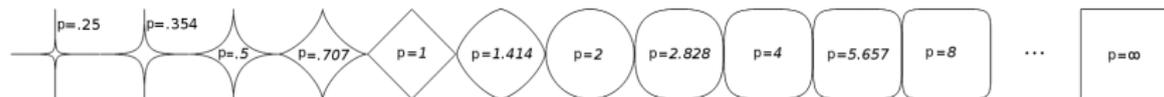
$$\|x\|_1 = 6 = 6.000$$

$$\|x\|_2 = \sqrt{14} = 3.742$$

$$\|x\|_3 = 6^{2/3} = 3.302$$

$$\|x\|_4 = 2^{1/4}\sqrt{7} = 3.146$$

$$\|x\|_\infty = 3 = 3.000$$



unit circles of l_p -norm with various values of p

マハラノビス距離 (Mahalanobis distance)

変数間に相関がある場合に、相関を考慮した距離

$$\text{mahalanobis}(x, y) = (x - y)\Sigma^{-1}(x - y)^T$$

ここで Σ^{-1} は共分散行列の逆行列

類似度

類似度

- ▶ ふたつのデータの似ている度合の数值表現

類似度の性質

非負性 (positivity)

$$0 \leq s(x, y) \leq 1$$

$$s(x, y) = 1 \Leftrightarrow x = y$$

対称性 (symmetry)

$$s(x, y) = s(y, x)$$

三角不等式 (triangle inequality) は一般に類似度には当てはまらない

バイナリベクトルの類似度

Jaccard 係数

- ▶ 1 の出現が少ないバイナリベクトル同士の類似度に使われる
- ▶ 文書中に出現する単語から文書の類似度を示す場合など
- ▶ 多くの単語は両方とも出現しない \Rightarrow これらは考慮しない
- ▶ 2 つのベクトルの各要素の対応関係を表のように集計

		vector y	
		1	0
vector x	1	n_{11}	n_{10}
	0	n_{01}	n_{00}

Jaccard 係数は以下で表される

$$J = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$$

n次元ベクトルの類似度

一般のベクトルの類似度

- ▶ 文書の類似度で出現頻度も考慮する場合など

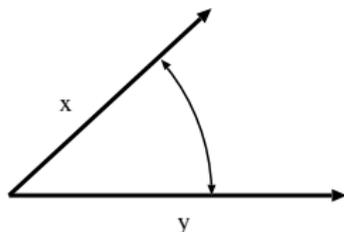
コサイン類似度

- ▶ ベクトルの x, y の cosine を取る、向きが一致:1、直交:0、向きが逆:-1
- ▶ ベクトルの長さで正規化 \Rightarrow 大きさは考慮しない

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

$x \cdot y = \sum_{k=1}^n x_k y_k$: ベクトルの積

$\|x\| = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{x \cdot x}$: ベクトルの長さ



コサイン類似度の例題

$$x = 3 \ 2 \ 0 \ 5 \ 0 \ 0 \ 0 \ 2 \ 0 \ 0$$

$$y = 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 2$$

$$x \cdot y = 3 * 1 + 2 * 1 = 5$$

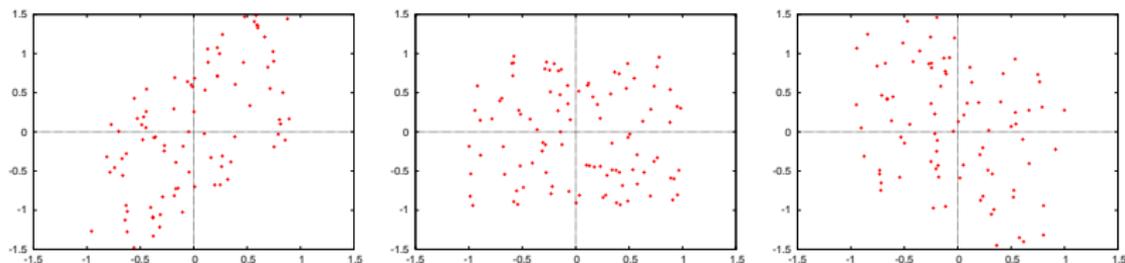
$$\|x\| = \sqrt{3 * 3 + 2 * 2 + 5 * 5 + 2 * 2} = \sqrt{42} = 6.481$$

$$\|y\| = \sqrt{1 * 1 + 1 * 1 + 2 * 2} = \sqrt{6} = 2.449$$

$$\cos(x, y) = \frac{5}{6.481 * 2.449} = 0.315$$

散布図と相関係数

- ▶ 散布図は 2 つの変数の関係を見るのに有効
 - ▶ X 軸: 変数 X
 - ▶ Y 軸: それに対応する変数 Y の値
- ▶ 散布図で分かる事
 - ▶ X と Y に関連があるか
 - ▶ 無相関、正の相関、負の相関
 - ▶ 外れ値の存在があるか
- ▶ 相関係数: 相関の方向 (正負) と強さを表す量



例: (左) 正の相関 0.7 (中) 無相関 0.0 (右) 負の相関 -0.5

相関 (correlation)

- ▶ 共分散 (covariance):

$$\sigma_{xy}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

- ▶ 相関係数 (correlation coefficient):

$$\rho_{xy} = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

- ▶ 相関係数は共分散を正規化したもの。 -1 から 1 の値を取る。
- ▶ 相関係数は外れ値の影響を大きく受ける。 散布図と併用し、外れ値を確認する必要。
- ▶ 相関関係と因果関係
 - ▶ 相関関係が因果関係を示すとは限らない。
 - ▶ 未知の第 3 の共通の要因が存在する場合
 - ▶ 単なる偶然

相関係数の計算 (1)

偏差平方和 (sum of squares)

$$\begin{aligned}\sum_{i=1}^n (x_i - \bar{x})^2 &= \sum_{i=1}^n (x_i^2 - 2x_i\bar{x} + \bar{x}^2) \\ &= \sum_{i=1}^n x_i^2 - 2\bar{x} \sum_{i=1}^n x_i + n\bar{x}^2 \\ &= \sum_{i=1}^n x_i^2 - 2\bar{x} \cdot n\bar{x} + n\bar{x}^2 \\ &= \sum_{i=1}^n x_i^2 - n\bar{x}^2 = \sum_{i=1}^n x_i^2 - \frac{(\sum_{i=1}^n x_i)^2}{n}\end{aligned}$$

偏差積和 (sum of products)

$$\begin{aligned}\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) &= \sum_{i=1}^n (x_i y_i - x_i \bar{y} - \bar{x} y_i + \bar{x} \bar{y}) \\ &= \sum_{i=1}^n x_i y_i - \bar{x} \sum_{i=1}^n y_i - \bar{y} \sum_{i=1}^n x_i + n\bar{x} \bar{y} \\ &= \sum_{i=1}^n x_i y_i - \bar{x} \cdot n\bar{y} - \bar{y} \cdot n\bar{x} + n\bar{x} \bar{y} \\ &= \sum_{i=1}^n x_i y_i - n\bar{x} \bar{y} = \sum_{i=1}^n x_i y_i - \frac{(\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n}\end{aligned}$$

相関係数の計算 (2)

相関係数 (correlation coefficient)

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

他の相関係数

- ▶ ピアソンの積率相関係数 (Pearson's product-moment correlation coefficient)
 - ▶ 単に相関係数といえはこれを指す (この授業でも同様)
- ▶ 順位相関係数 (rank correlation coefficient):
 - ▶ 同じ項目を持つデータセットに対する順位付けの違いとその関係
 - ▶ スピアマンの順位相関係数
 - ▶ ケンドールの順位相関係数
- ▶ その他の相関係数

課題 1: 東京マラソン完走時間のプロット

- ▶ ねらい: 実データから分布を調べる
- ▶ データ: 2014 年の東京マラソンの記録
 - ▶ <http://www.tokyo42195.org/history/>
 - ▶ フルマラソン参加者のネットタイム (公式タイムではない) 完走者 34,058 人
- ▶ 提出項目
 1. 全完走者、男性完走者、女性完走者それぞれの、完走時間の平均、標準偏差、中間値
 2. それぞれの完走時間のヒストグラム
 - ▶ 3つのヒストグラムを別々の図に書く
 - ▶ ビン幅は 10 分にする
 - ▶ 3つのプロットは比較できるように目盛を合わせる
 3. それぞれの CDF プロット
 - ▶ ひとつの図に 3つの CDF プロットを書く
 4. オプション: その他の解析
 5. 考察
 - ▶ データから読みとれることを記述
- ▶ 提出形式: レポートをひとつの PDF ファイルにして SFC-SFS から提出
- ▶ 提出〆切: 2014 年 5 月 16 日 (extended)

東京マラソンデータ

データフォーマット

#	bib#	Name	Category	5km	10km	15km	20km	25km	30km	35km	40km	FinishTime
1	"	キルイ アベル"	M	0:14:50	0:29:37	0:44:33	0:59:42	1:14:48	1:30:01	1:45:32	2:01:37	2:09:02
2	"	トラ タデセ"	M	0:14:51	0:29:38	0:44:34	0:59:43	1:14:50	1:30:01	1:44:57	1:59:19	2:05:56
3	"	キビエゴ マイケル"	M	0:14:51	0:29:38	0:44:33	0:59:42	1:14:48	1:30:00	1:44:56	1:59:54	2:06:56
4	"	キトワラ サミー"	M	0:14:50	0:29:38	0:44:33	0:59:42	1:14:48	1:30:00	1:44:56	1:59:43	2:06:28
5	"	ソメ ピーター"	M	0:14:50	0:29:38	0:44:33	0:59:42	1:14:49	1:30:00	1:44:56	2:00:20	2:07:03
6	"	チムサ デレサ"	M	0:14:50	0:29:38	0:44:33	0:59:43	1:14:49	1:30:01	1:45:03	2:00:27	2:07:38
7	"	チュンバ ディクソン"	M	0:14:51	0:29:38	0:44:34	0:59:43	1:14:50	1:30:01	1:44:57	1:59:18	2:05:41
8	"	キプサング ジョフリー"	M	0:14:52	0:29:39	0:44:34	0:59:43	1:14:50	1:30:01	1:44:57	2:00:00	2:07:36
9	"	ロスリン ビクトル"	?									
10	"	アスメロン ヤレド"	M	0:14:54	0:30:12							
11	"	ブーラムダン アブデラヒム"	M	0:14:54	0:30:03	0:45:16	1:00:50	1:16:31	1:32:27	1:48:33	2:05:00	2:12:07
21	"	藤原 新"	M	0:14:51	0:29:38	0:44:32	0:59:42	1:14:50	1:31:56	1:54:16	2:20:15	2:30:56
22	"	中本 健太郎"	?									
23	"	ジュイ サイラス"	M	0:14:51	0:29:38	0:44:33	0:59:42	1:14:49	1:30:02	1:45:42	2:01:52	2:09:33
24	"	石川 末廣"	M	0:14:51	0:29:38	0:44:33	0:59:42	1:14:49	1:30:13	1:46:01	2:02:16	2:09:27
...												

- ▶ bib#: ゼッケン番号
 - ▶ 1-43:招待 101-235:エリート 10000 台:陸連登録選手 20000-50000 台:一般参加 60001:ゲスト 70000 台:チャリティランナー
- ▶ Name: " "で囲まれている (UTF-8)
- ▶ Category: M(Men)/W(Women)
 - ▶ 棄権だと"?"となっている
- ▶ ネットタイム: 機械的に読みとったスタートからの時間 (5km ごとのスプリットと完走時間)
- ▶ 完走者を抽出したら、総数が合っているかチェックすること

課題 1 問 1 平均、標準偏差、中間値の計算

- ▶ 分単位での計算 (秒まで含めた値とは少し異なる)

	n	mean	stddev	median
all	34,058	282.7	56.8	281
men	27,141	277.2	56.7	275
women	6,917	304.0	51.8	306

データ抽出用スクリプト例

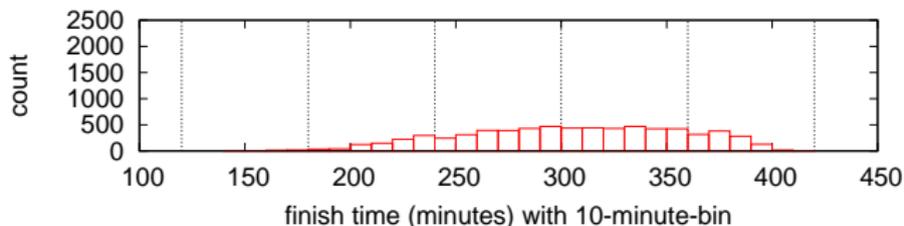
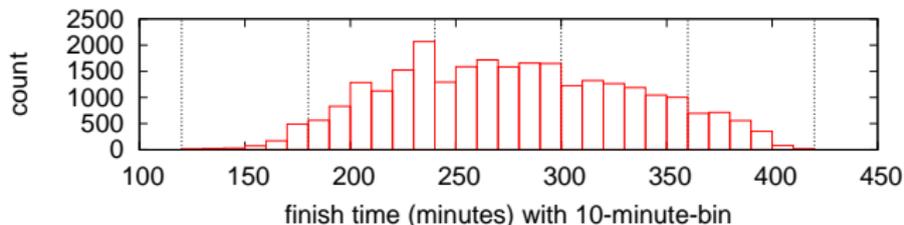
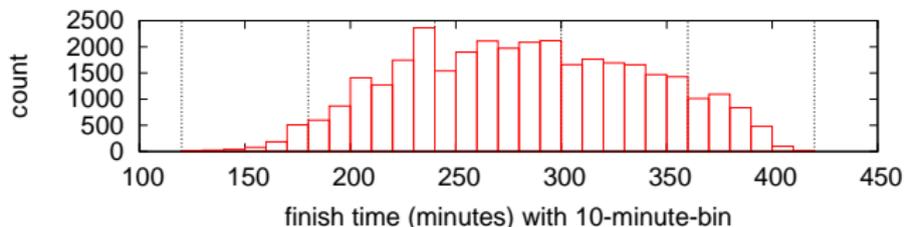
```
# regular expression to read finishtime from tokyomarathon2014.txt
re = /(\d+)\s+".+?"\s+(M|W)(?:\s+(\d:\d{2}:\d{2}))?{9}/

# regular expression to read hour:minute:second
re_time = /(\d):(\d{2}):(\d{2})/

ARGF.each_line do |line|
  if re.match(line)
    bib, category, ftime = $~.captures
    if re_time.match(ftime)
      min = $1.to_i * 60 + $2.to_i
      print "#{min}\t#{category}\n"
    end
  end
end
```

課題 1 問 2 完走時間のヒストグラム

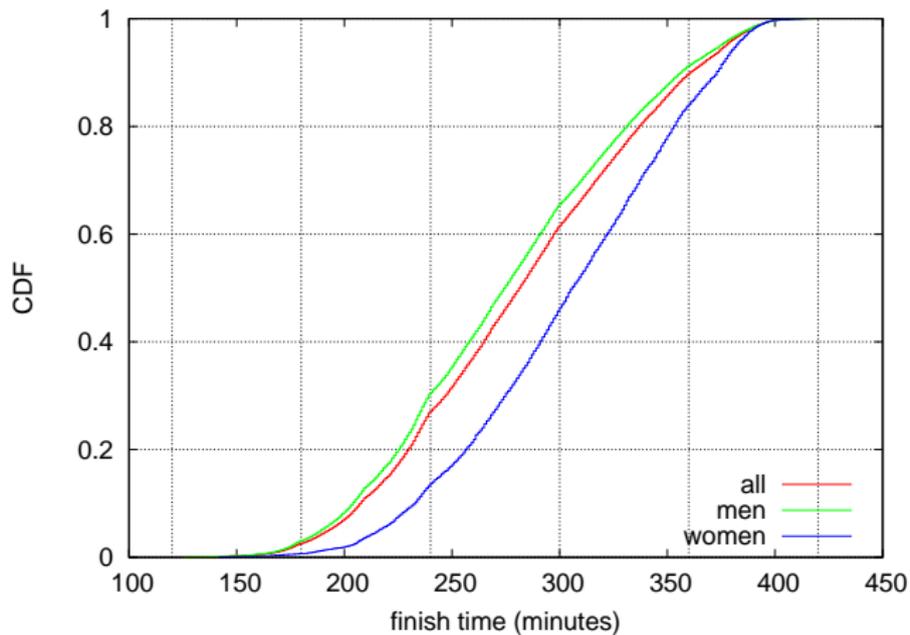
- ▶ 3つのヒストグラムを別々の図に書く
- ▶ ビン幅は10分にする
- ▶ 3つのプロットは比較できるように目盛を合わせること



完走時間ヒストグラム 全体 (上) 男子 (中) 女子 (下)

課題 1 問 3 CDF プロット

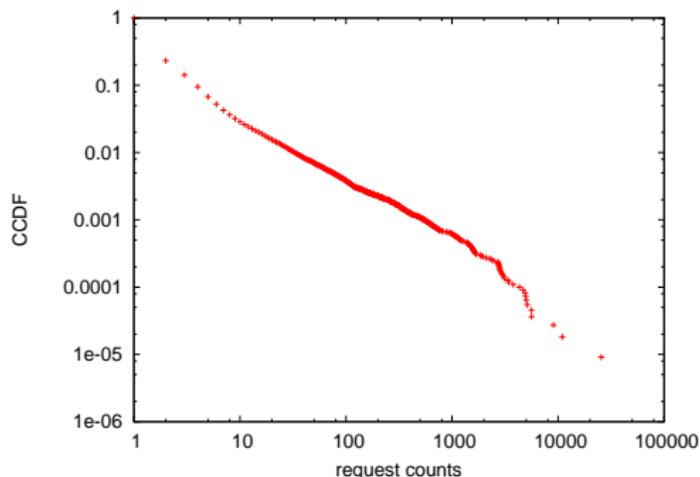
- ▶ ひとつの図に 3 つのプロットを書く



前回の演習: CCDF のプロット

第3回演習で使った JAIST サーバのアクセスログのコンテンツ毎のアクセス数分布を CCDF プロットにする

```
% ./count_contents.rb sample_access_log > contents.txt  
% ./make_ccdf.rb contents.txt > ccdf.txt
```



コンテンツ毎のアクセス数抽出スクリプト

```
# output: URL req_count byte_count
# regular expression for apache combined log format
# host ident user time request status bytes referer agent
re = /^(S+) (\S+) (\S+) \[(.*?)\] "(.*?)" (\d+) (\d+|-) "(.*?)" "(.*?)" /
# regular expression for request: method url proto
req_re = /(\w+) (\S+) (\S+)/
contents = Hash.new([0, 0])
count = parsed = 0
ARGF.each_line do |line|
  count += 1
  if re.match(line)
    host, ident, user, time, request, status, bytes, referer, agent = $~.captures
    # ignore if the status is not success (2xx)
    next unless /2\d{2}/.match(status)
    if req_re.match(request)
      method, url, proto = $~.captures
      # ignore if the method is not GET
      next unless /GET/.match(method)
      parsed += 1
      # count contents by request and bytes
      contents[url] = [contents[url][0] + 1, contents[url][1] + bytes.to_i]
    else
      # match failed. print a warning msg
      $stderr.puts("request match failed at line #{count}: #{line.dump}")
    end
  else
    $stderr.puts("match failed at line #{count}: #{line.dump}") # match failed.
  end
end
contents.sort_by{|key, value| -value[0]}.each do |key, value|
  puts "#{key} #{value[0]} #{value[1]}"
end
$stderr.puts "# #{contents.size} unique contents in #{parsed} successful GET requests"
$stderr.puts "# parsed:#{parsed} ignored:#{count - parsed}"
```

コンテンツ毎のアクセス数

```
% cat contents.txt
```

```
/project/linuxonandroid/Ubuntu/12.04/full/ubuntu1204-v4-full.zip 25535 17829045  
/project/morefont/xiongmaozhongwen.apk 10949 13535294486  
/project/morefont/zhongguoxin.apk 9047 9549531354  
/project/honi/some_software/Windows/Office_Plus_2010_SP1_W32_xp911.com.rar 5616  
/project/morefont/fangzhengyouyijian.apk 5609 2879391721  
/pub/Linux/CentOS/5.9/extras/i386/repodata/repomd.xml 5121 12213484  
/pub/Linux/CentOS/5.9/updates/i386/repodata/repomd.xml 5006 10969621  
/pub/Linux/CentOS/5.9/os/i386/repodata/repomd.xml 4953 6832653  
/project/npppluginmgr/xml/plugins.md5.txt 4881 1369547  
/project/winpenpack/X-LenMus/releases/X-LenMus_5.3.1_rev5.zip 4689 990250462
```

```
...
```

```
/pub/Linux/openSUSE/distribution/12.3/repo/oss/suse/x86_64/gedit-3.6.2-2.1.2.x8  
/pub/sourceforge/n/nz/nzbcatcher/source/?C=D;O=A 1 1075  
/ubuntu/pool/universe/m/mmass/mmass_5.4.1.orig.tar.gz 1 3754849
```

アクセス数を CCDF にするスクリプト

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

re = /^S+s+(\d+)\s+\d+/

n = 0
counts = Hash.new(0)
ARGF.each_line do |line|
  if re.match(line)
    counts[$1.to_i] += 1
    n += 1
  end
end

cum = 0
counts.sort.each do |key, value|
  comp = 1.0 - Float(cum) / n
  puts "#{key} #{value} #{comp}"
  cum += value
end
```

アクセス数の相補累積度数

```
% cat ccdf.txt
1 84414 1.0
2 9813 0.2315731022366253
3 5199 0.14224463601358184
4 3034 0.0949177537254331
5 1636 0.06729902688137779
6 1083 0.05240639764048316
7 663 0.04254776838138241
8 495 0.03651243024769468
9 367 0.03200640856417214
10 274 0.028665580366489807

...

5616 1 3.6412296432475344e-05
9047 1 2.730922232441202e-05
10949 1 1.8206148216237672e-05
25535 1 9.103074108174347e-06
```

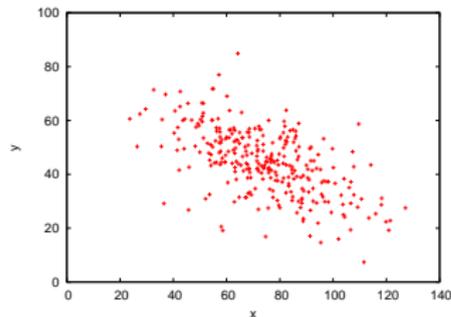
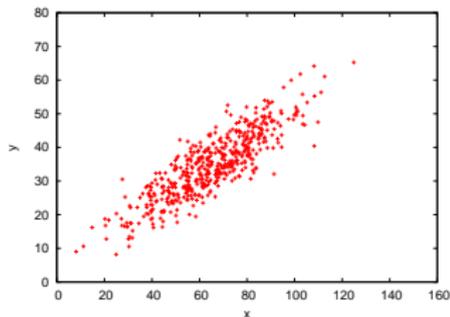
gnuplot スクリプト

```
set logscale
set xlabel "request counts"
set ylabel "CCDF"

plot "ccdf.txt" using 1:3 notitle with points
```

今回の演習: 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)

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 = /([+]?[0-9]+\.[0-9]*)\s+([+]?[0-9]+\.[0-9]*)/

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
```

今回の演習 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なになる

```
#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
...
```

まとめ

第6回 相関

- ▶ オンラインお勧めシステム
- ▶ 距離とエントロピー
- ▶ 相関係数
- ▶ 演習: 相関

次回予定

第7回 多変量解析 (5/26)

- ▶ 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