

インターネット計測とデータ解析 第12回

長 健二郎

2014年6月30日

前回のおさらい

第 11 回 データマイニング (6/23)

- ▶ パターン抽出
- ▶ クラス分類
- ▶ クラスタリング
- ▶ 演習: クラスタリング

今日のテーマ

第 12 回 検索とランキング

- ▶ 検索システム
- ▶ ページランク
- ▶ 演習: PageRank

検索エンジンの歴史

ほとんどのインターネットユーザが毎日利用する検索エンジン

- ▶ 1994 Yahoo! ポータル開設
 - ▶ ポータルの先駆け (ディレクトリ型)
 - ▶ 最初は自分たちのお気に入りをお勧めサイトとして公開
- ▶ 1995 Altavista
 - ▶ 検索エンジンの先駆け、ロボットによるクローリング、多言語対応
 - ▶ スпам等で精度が低下する問題
- ▶ 1998 Google サービス開始
 - ▶ Google が PageRank に基づくロボットによる検索サービス開始
 - ▶ 各ページの人気を基にスコアを算出

検索エンジンの仕組み

- ▶ ディレクトリ型
 - ▶ 人手による登録、分類
 - ▶ 高品質だがスケールしない
- ▶ ロボット型
 - ▶ 自動的に web を巡回してデータベースを作成
 - ▶ web page 数の増大に伴い主流に

ロボット型検索エンジン

- ▶ web page を収集する
 - ▶ クローリング
- ▶ 収集した情報のデータベース管理
 - ▶ インデックス生成
- ▶ 検索クエリーから web page をマッチ
 - ▶ 検索ランキング

インデックス生成

- ▶ Web page からキーワードを抽出
- ▶ キーワードから Web page への転置インデックスを作成

検索ランキング

検索時には、検索サーバは、

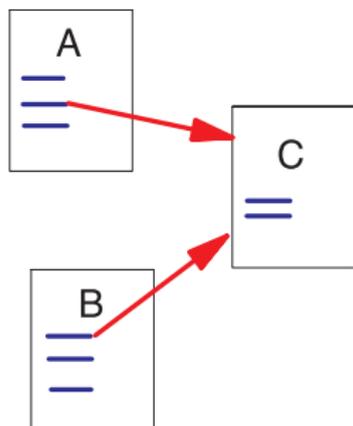
- ▶ キーワードから転置インデックスを使って、関連する Web ページのリストを得る
- ▶ リストされた Web ページをランキング順に並び変えて送信

Web ページのランキング

- ▶ Web ページの重要度を示す指標が必要
- ▶ PageRank: Google のランキング技術

PageRank: アイデア

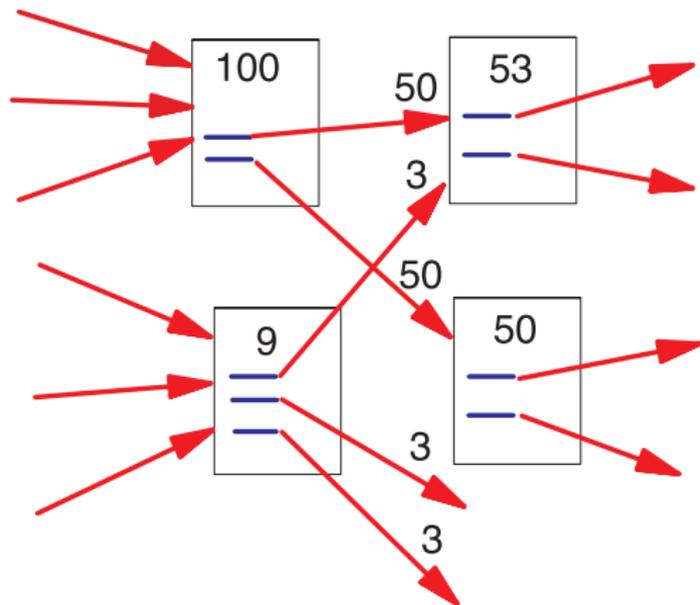
- ▶ Web ページのリンク関係だけからページをランキング
 - ▶ ページコンテンツはまったく見ない



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

PageRank の考え方

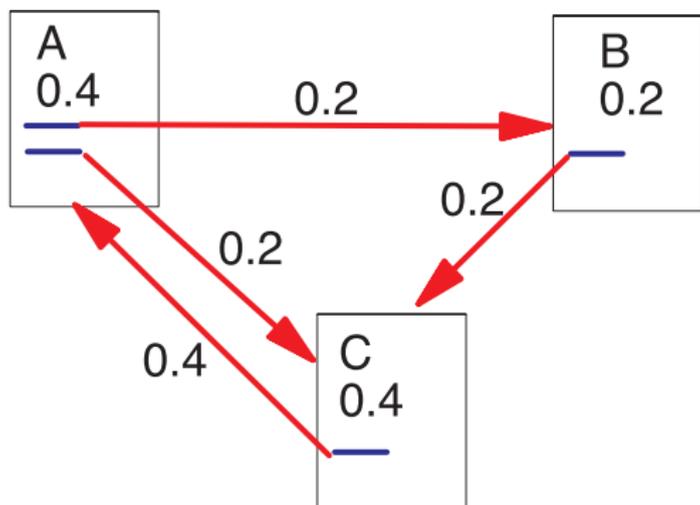
- ▶ 良質なページは、多くのページからリンクされる
- ▶ 良質なページからのリンクは価値が高い
- ▶ ページ内のリンク数が増えると、個々のリンクの価値は減る



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

PageRank の考え方

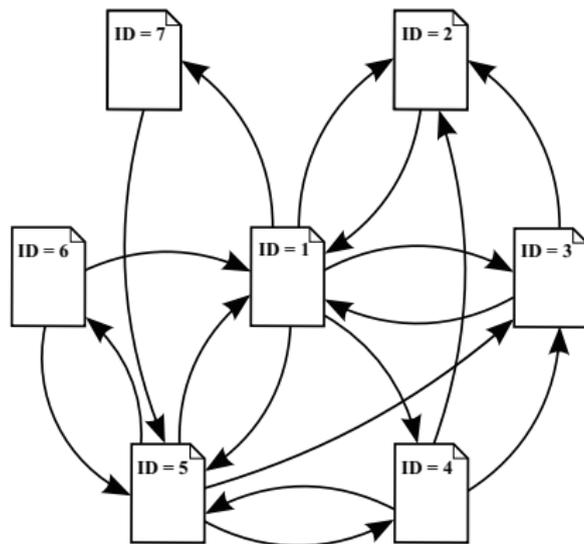
- ▶ 良質なページからリンクされるページは良質である
- ▶ ランダムサーファーモデル
 - ▶ ページ内のリンクを同じ確率でクリックして次のページへ



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

PageRank の例

Page ID	OutLinks
1	2, 3, 4, 5, 7
2	1
3	1, 2
4	2, 3, 5
5	1, 3, 4, 6
6	1, 5
7	5



行列によるモデル

Matrix Notation (src \rightarrow dst)

$$A^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Transition Matrix (dst \leftarrow src) 列の合計は 1

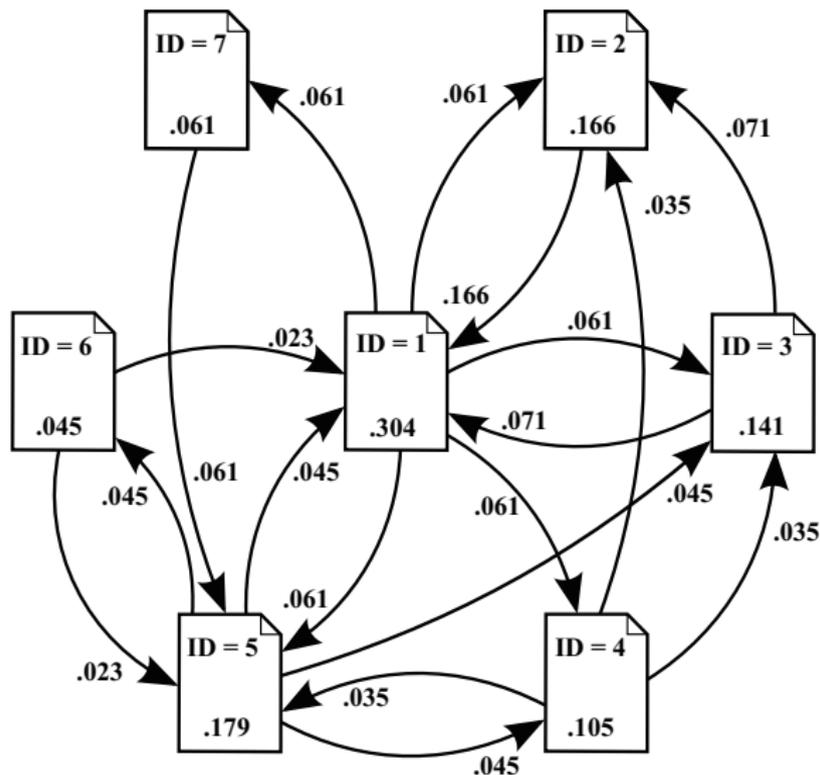
$$A = \begin{bmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R = cAR$$

ページランクベクトル R は、遷移確率行列 A の固有ベクトル、 c は固有値の逆数

PageRank の例の計算結果

固有値計算で求まる



シンプル PageRank モデルの問題

- ▶ 現実には
 - ▶ 外向きリンクがないノードが存在 (dangling node)
 - ▶ 内向きリンクがないノードが存在
 - ▶ 閉ループが存在
- ▶ 推移確率モデルはマルコフ連鎖の遷移確率行列
 - ▶ 十分時間が経過すると平衡状態に収束する
- ▶ 収束条件: 行列は再帰かつ既約
 - ▶ 有向グラフは強連結 (任意のノードから任意のノードに到達可能)
 - ▶ ひとつの優固有ベクトルが存在

解決案: 一定の確率でランダムなページに飛ぶ挙動を追加

PageRank アルゴリズム

任意の初期値から始めて、各ページのランクが収束するまで遷移を繰り返す

- ▶ case: node with outlinks (> 0)
 - ▶ d の確率で、ノード内のリンクをランダムに選択
 - ▶ $(1 - d)$ の確率で、ランダムなノードにジャンプ
- ▶ case: dangling node (no outlink)
 - ▶ ランダムなノードにジャンプ

$$A' = dA + (1 - d)[1/N]$$

d: damping factor (= 0.85)

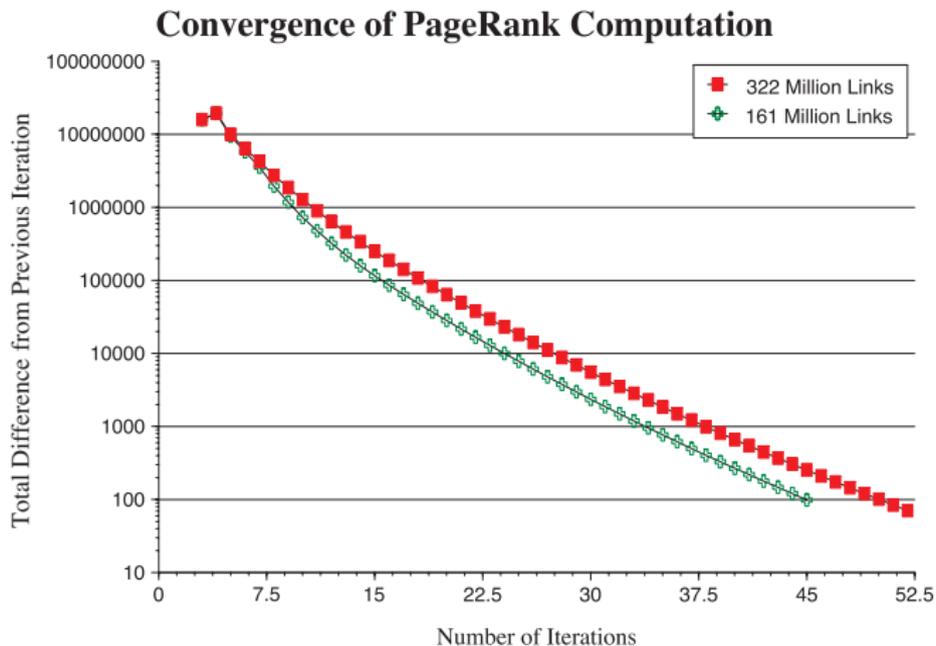
べき乗法による計算

- ▶ 固有値計算は行列が大きくなるとメモリ消費と計算量が膨大
- ▶ べき乗法による逐次繰返しによる近似

```
parameters:
  d: dampig_factor = 0.85
  thresh: convergence_threshold = 0.000001
initialize:
  for i
    r[i] = 1/N
loop:
  e = 0
  for i
    new_r[i] = d * (sum_inlink(r[j])/degree[j]) + sum_dangling(r[j])/N
              + (1 - d)/N
    e += |new_r[i] - r[i]|
  r = new_r
while e > thresh
```

PageRank の収束

- ▶ 大量のページがあっても対数的に収束する実験結果



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

PageRank のまとめ

- ▶ シンプルなアイデア
 - ▶ 良質なページからリンクされるページは良質である
- ▶ アイデアをマルコフ遷移確率行列で定式化、収束を保証
- ▶ スケールする実装を行い、実データで有効性を実証
- ▶ ビジネスに繋げ、トップ企業に

- ▶ 注: 紹介したのは最初の論文のアルゴリズムで、現在 Google が使っているアルゴリズムは大幅に改良されているはず

google servers



google system in 1998 and a current data center

今回の演習: PageRank

```
% cat sample-links.txt
# PageID: OutLinks
1:      2      3      4      5      7
2:      1
3:      1      2
4:      2      3      5
5:      1      3      4      6
6:      1      5
7:      5

% ruby pagerank.rb -f 1.0 sample-links.txt
reading input...
initializing... 7 pages dampingfactor:1.00 thresh:0.000001
iteration:1 diff_sum:0.661905 rank_sum: 1.000000
iteration:2 diff_sum:0.383333 rank_sum: 1.000000
...
iteration:20 diff_sum:0.000002 rank_sum: 1.000000
iteration:21 diff_sum:0.000001 rank_sum: 1.000000
[1] 1 0.303514
[2] 5 0.178914
[3] 2 0.166134
[4] 3 0.140575
[5] 4 0.105431
[6] 7 0.060703
[7] 6 0.044728
```

今回の演習: PageRank code (1/4)

```
require 'optparse'

d = 0.85 # damping factor (recommended value: 0.85)
thresh = 0.000001 # convergence threshold

OptionParser.new {|opt|
  opt.on('-f VAL', Float) {|v| d = v}
  opt.on('-t VAL', Float) {|v| thresh = v}
  opt.parse!(ARGV)
}

outdegree = Hash.new # outdegree[id]: outdegree of each page
inlinks = Hash.new # inlinks[id][src0, src1, ...]: inlinks of each page
rank = Hash.new # rank[id]: pagerank of each page
last_rank = Hash.new # last_rank[id]: pagerank at the last stage
dangling_nodes = Array.new # dangling pages: pages without outgoing link

# read a page-link file: each line is "src_id dst_id_1 dst_id_2 ..."
ARGF.each_line do |line|
  pages = line.split(/\D+/) # extract list of numbers
  next if line[0] == '?' || pages.empty?

  src = pages.shift.to_i # the first column is the src
  outdegree[src] = pages.length
  if outdegree[src] == 0
    dangling_nodes.push src
  end
  pages.each do |pg|
    dst = pg.to_i
    inlinks[dst] ||= []
    inlinks[dst].push src
  end
end
```

今回の演習: PageRank code (2/4)

```
# initialize
# sanity check: if dst node isn't defined as src, create one as a dangling node
inlinks.each_key do |j|
  if !outdegree.has_key?(j)
    # create the corresponding src as a dangling node
    outdegree[j] = 0
    dangling_nodes.push j
  end
end

n = outdegree.length # total number of nodes
# initialize the pagerank of each page with 1/n
outdegree.each_key do |i| # loop through all pages
  rank[i] = 1.0 / n
end
$stderr.printf " %d pages dampingfactor:%.2f thresh:%f\n", n, d, thresh
```

今回の演習: PageRank code (3/4)

```
# compute pagerank by power method
k = 0 # iteration number
begin
  rank_sum = 0.0 # sum of pagerank of all pages: should be 1.0
  diff_sum = 0.0 # sum of differences from the last round
  last_rank = rank.clone # copy the entire hash of pagerank

  # compute dangling ranks
  danglingranks = 0.0
  dangling_nodes.each do |i| # loop through dangling pages
    danglingranks += last_rank[i]
  end

  # compute page rank
  outdegree.each_key do |i| # loop through all pages
    inranks = 0.0
    # for all incoming links for i, compute
    # inranks = sum (rank[j]/outdegree[j])
    if inlinks[i] != nil
      inlinks[i].each do |j|
        inranks += last_rank[j] / outdegree[j]
      end
    end
  end

  rank[i] = d * (inranks + danglingranks / n) + (1.0 - d) / n
  rank_sum += rank[i]

  diff = last_rank[i] - rank[i]
  diff_sum += diff.abs
end

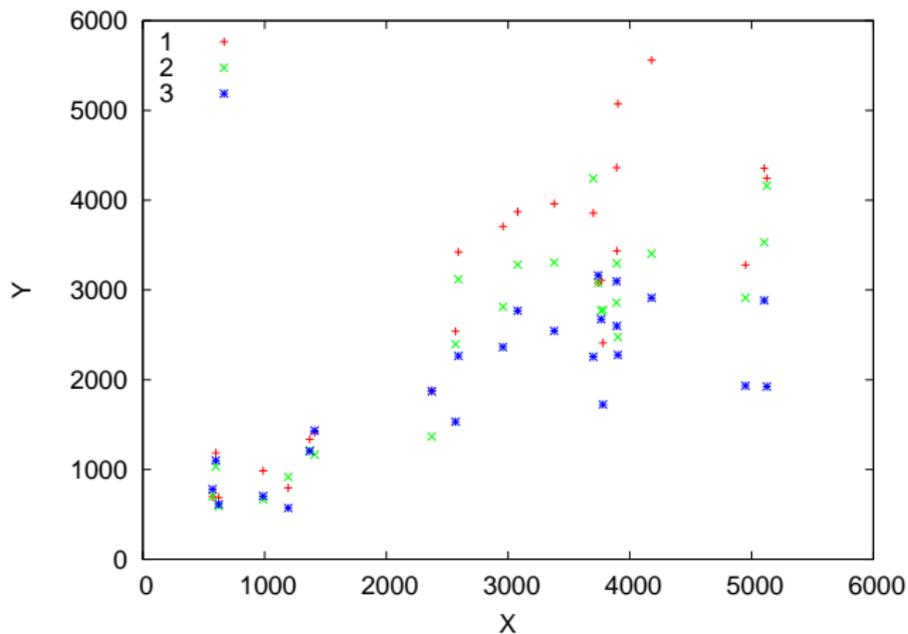
k += 1
$stderr.printf "iteration:%d diff_sum:%f rank_sum: %f\n", k, diff_sum, rank_sum
end while diff_sum > thresh
```

今回の演習: PageRank code (4/4)

```
# print pagerank in the decreasing order of the rank
# format: [position] id pagerank
i = 0
rank.sort_by{|k, v| -v}.each do |k, v|
  i += 1
  printf "[%d] %d %f\n", i, k, v
end
```

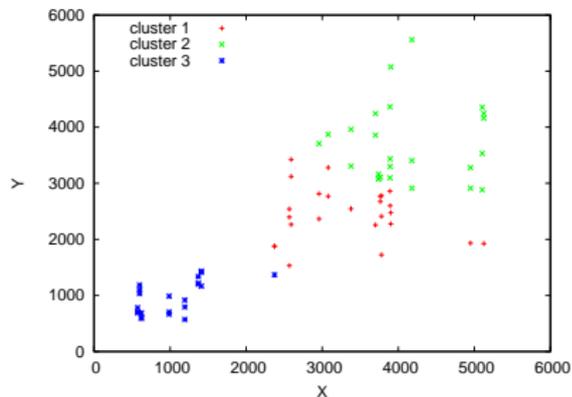
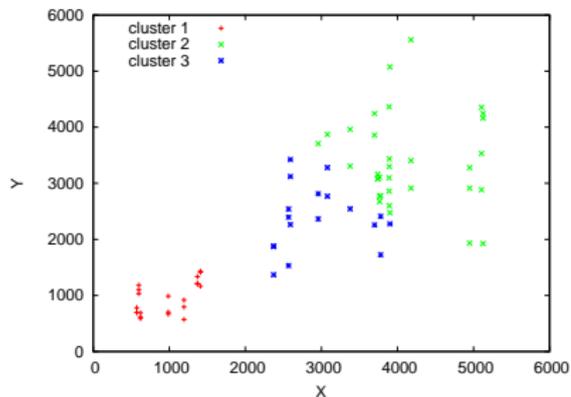
前回の演習: k-means clustering

```
% ruby k-means.rb km-data.txt > km-results.txt
```



k-means clustering results

▶ 初期値によって結果が異なる



サンプルコード (1/2)

```
k = 3 # k clusters
re = /^(d+)\s+(d+)/
INFINITY = 0x7fffffff

# read data
nodes = Array.new # array of array for data points: [x, y, cluster_index]
centroids = Array.new # array of array for centroids: [x, y]
ARGF.each_line do |line|
  if re.match(line)
    c = rand(k) # randomly assign initial cluster
    nodes.push [$1.to_i, $2.to_i, c]
  end
end

round = 0
begin
  updated = false

  # assignment step: assign each node to the closest centroid
  if round != 0 # skip assignment for the 1st round
    nodes.each do |node|
      dist2 = INFINITY # square of distance to the closest centroid
      cluster = 0 # closest cluster index
      for i in (0 .. k - 1)
        d2 = (node[0] - centroids[i][0])**2 + (node[1] - centroids[i][1])**2
        if d2 < dist2
          dist2 = d2
          cluster = i
        end
      end
      node[2] = cluster
    end
  end
end
```

サンプルコード (2/2)

```
# update step: compute new centroids
sums = Array.new(k)
clsize = Array.new(k)
for i in (0 .. k - 1)
  sums[i] = [0, 0]
  clsize[i] = 0
end
nodes.each do |node|
  i = node[2]
  sums[i][0] += node[0]
  sums[i][1] += node[1]
  clsize[i] += 1
end

for i in (0 .. k - 1)
  newcenter = [Float(sums[i][0]) / clsize[i], Float(sums[i][1]) / clsize[i]]
  if round == 0 || newcenter[0] != centroids[i][0] || newcenter[1] != centroids[i][1]
    centroids[i] = newcenter
    updated = true
  end
end

round += 1

end while updated == true

# print the results
nodes.each do |node|
  puts "#{node[0]}\t#{node[1]}\t#{node[2]}"
end
```

gnuplot script

```
set key left
set xrange [0:6000]
set yrange [0:6000]
set xlabel "X"
set ylabel "Y"

plot "km-results.txt" using 1:($3==0?$2:1/0) title "cluster 1" with points, \
"km-results.txt" using 1:($3==1?$2:1/0) title "cluster 2" with points, \
"km-results.txt" using 1:($3==2?$2:1/0) title "cluster 3" with points
```

最終レポートについて

- ▶ A, B からひとつ選択
 - ▶ A. ウィキペディア日本語版の Pageview ランキング
 - ▶ B. 自由課題
- ▶ 8 ページ以内
- ▶ pdf ファイルで提出
- ▶ 提出〆切: 2014 年 7 月 28 日 (月) 23:59

最終レポート 選択テーマ

A. ウィキペディア日本語版の Pageview ランキング

- ▶ ねらい: 実データから人気キーワードを抽出し時間変化を観測
- ▶ データ: ウィキペディア日本語版の Pageview データ
- ▶ 提出項目
 - ▶ A-1 Pageview カウント分布調査
 - ▶ 各ページの 1 週間分のリクエスト総数を集計し、分布を CCDF でプロット
 - ▶ A-2 各日および 1 週間合計からリクエスト数トップ 10 を抽出
 - ▶ トップ 10 の結果を表にする
 - ▶ A-3 週間トップ 10 についてランキングの推移をプロット
 - ▶ ランキング変化が分かり易いよう時間粒度を考え図を工夫する
 - ▶ A-4 オプション解析: その他の自由解析
 - ▶ A-5 考察: データから読みとれることを考察

B. 自由課題

- ▶ 授業内容と関連するテーマを自分で選んでレポート
- ▶ 必ずしもネットワーク計測でなくてもよいが、何らかのデータ解析を行い、考察すること

最終レポートは考察を重視する

課題 A. ウィキペディア日本語版の Pageview ランキング

データ: ウィキペディア日本語版のデータ Pageview データ

- ▶ wikimedia が提供するデータからウィキペディア日本語版だけを抜き出したもの。
- ▶ 元データ情報:
<http://dumps.wikimedia.org/other/pagecounts-raw/>
- ▶ 課題用 Pageview データ: 20140616-22.zip (609MB 解凍後 3GB)
 - ▶ 1 時間毎の Pageview データ 1 週間分 (2014 年 6 月 16 日-22 日)
- ▶ オプションデータセット: 20140601-15.zip (1.3GB 解凍後 6.3GB)
 - ▶ オプション解析で利用可能な追加データ (2014 年 6 月 1 日-15 日)

データフォーマット

- ▶ project encoded_pagetitle requests size
 - ▶ project: wikimedia のプロジェクト名 (課題用データでは全て"ja")
 - ▶ encoded_pagetitle: URI エンコードされたページタイトル
 - ▶ requests: ページのリクエスト回数
 - ▶ size: ページのバイト数

```
% head -n 10 20140616-22/pagecounts-20140616-00
ja $ 1 0
ja $10 1 8922
ja %22B%22ORDERLESS 1 13777
ja %22BLUE%22_A_TRIBUTE_TO_YUTAKA_OZAKI 1 21159
ja %22HAPPY%22_Coming_Century,_20th_Century_Forever 1 21326
ja %22LUCKY%22_20th_Century,_Coming_Century_to_be_continued... 1 0
ja %22X%22_plosion_GUNDAM_SEED 2 50386
ja %26C 1 16485
ja %26_(%E4%B8%80%E9%9D%92%E7%AA%88%E3%81%AE%E3%82%A2%E3%83%AB%E3%83%90%E3%83%A0) 1 12635
ja %26_(%E6%BC%AB%E7%94%BB) 1 0
```

```
% head -n 10 20140616-22/pagecounts-20140616-00 | ./urldecode.rb
ja "$" 1 0
ja "$10" 1 8922
ja ""B"ORDERLESS" 1 13777
ja ""BLUE"_A_TRIBUTE_TO_YUTAKA_OZAKI" 1 21159
ja ""HAPPY"_Coming_Century,_20th_Century_Forever" 1 21326
ja ""LUCKY"_20th_Century,_Coming_Century_to_be_continued..." 1 0
ja ""X"_plosion_GUNDAM_SEED" 2 50386
ja "&C" 1 16485
ja "&_(一青窈のアルバム)" 1 12635
ja "&_(漫画)" 1 0
```

タイトルのデコードスクリプト

- ▶ タイトルはパーセントエンコードされている
 - ▶ ruby の CGI.unescape() で UTF-8 に変換できる

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

require 'cgi'

re = /^(([\w\.]+)\s+(\S+)\s+(\d+)\s+(\d+)/

ARGF.each_line do |line|
  if re.match(line)
    project, title, requests, bytes = $~.captures
    decoded_title = CGI.unescape(title)
    print "#{project} \"#{decoded_title}\" #{requests} #{bytes}\n"
  end
end
```

課題 A Pageview ランキング補足

- ▶ A-1 Pageview カウント分布調査
 - ▶ 各ページの 1 週間分のリクエスト総数を集計し、分布を CCDF でプロット
 - ▶ X 軸はリクエスト数、Y 軸は CCDF、log-log でプロット
- ▶ A-2 各日および 1 週間合計からリクエスト数トップ 10 を抽出
 - ▶ トップ 10 の結果を表にする

```
rank  6/16 6/17 6/18 6/19 6/20 6/21 6/22 total
     1  "a"  "b"  "c"  "d"  "e"  "f"  "g"  "x"
     2  "h"  "i"  "j"  "k"  "l"  "m"  "n"  "y"
     ...
```

- ▶ A-3 週間トップ 10 についてランキングの推移をプロット
 - ▶ X 軸に時間、Y 軸にランキングをとる
 - ▶ ランキング変化が分かり易いよう時間粒度やランキングの見せ方を考え図を工夫する

まとめ

第 12 回 検索とランキング

- ▶ 検索システム
- ▶ ページランク
- ▶ 演習: PageRank

次回予定

第 13 回 スケールする計測と解析 (7/7)

- ▶ 大規模計測
- ▶ クラウド技術
- ▶ MapReduce
- ▶ 演習: MapReduce