インターネット計測とデータ解析 第12回

長 健二朗

2012年6月29日

前回のおさらい

ゲストトーク

- AS Core: Visualizing the Internet
- Bradley Huffaker (CAIDA)

前前回のおさらい

異常検出と機械学習

- 異常検出
- 機械学習
- スパム判定とベイズ理論
- 演習: 機械学習

今日のテーマ

検索とランキング

- 検索システム
- クローリング
- 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
  - スパム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、

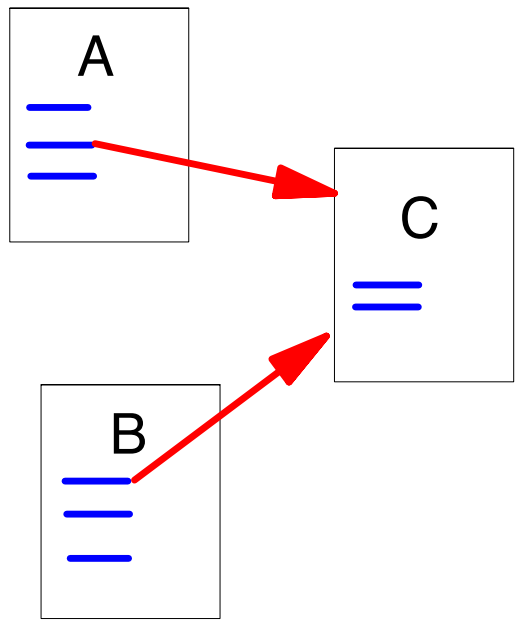
- キーワードから転置インデックスを使って、関連する Web ページのリストを得る
- リストされた Web ページをランキング順に並び変えて送信

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

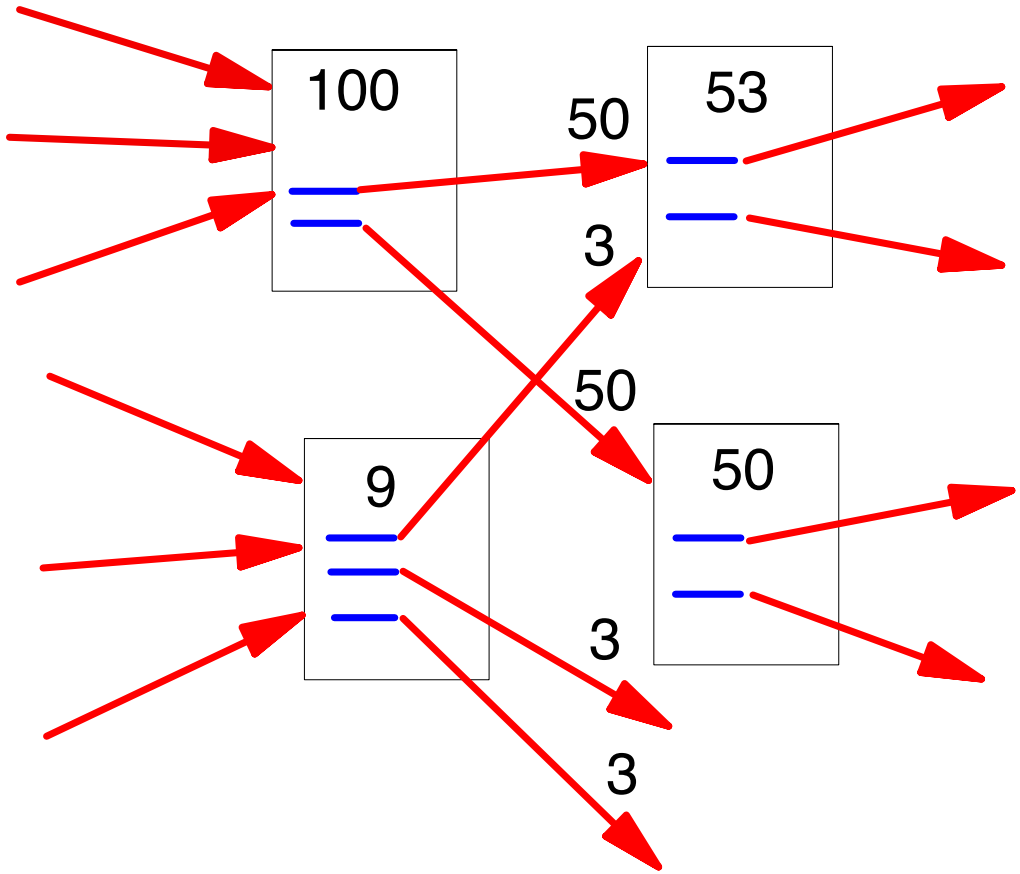
- Web ページのリンク関係だけからページをランキング
  - ページコンテンツはまったく見ない



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

PageRank の考え方

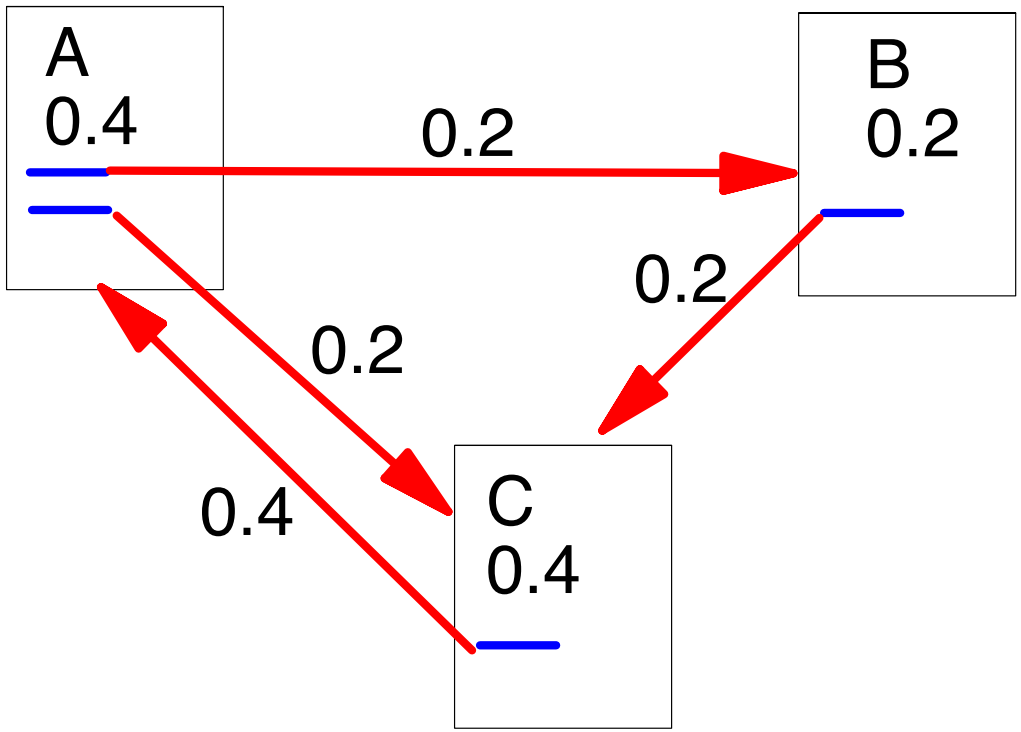
- 良質なページは、多くのページからリンクされる
- 良質なページからのリンクは価値が高い
- ページ内のリンク数が増えると、個々のリンクの価値は減る



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

PageRank の考え方

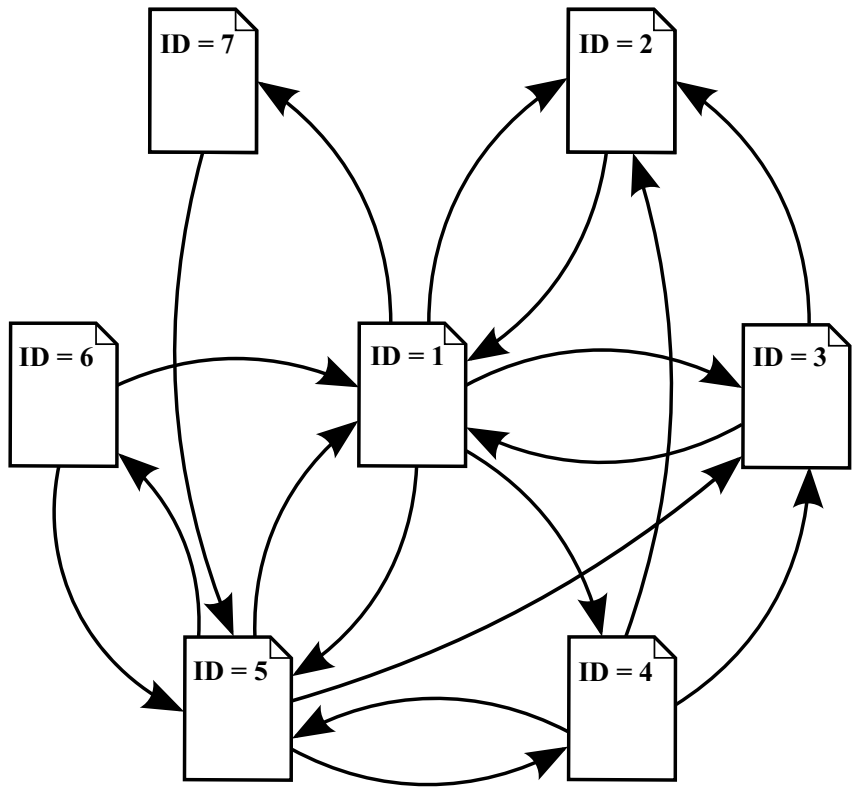
- 良質なページからリンクされるページは良質である
- ランダムサーファーモデル
  - ページ内のリンクを同じ確率でクリックして次のページへ



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 → dst)

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

Transition Matrix (dst ← src) 列の合計は 1

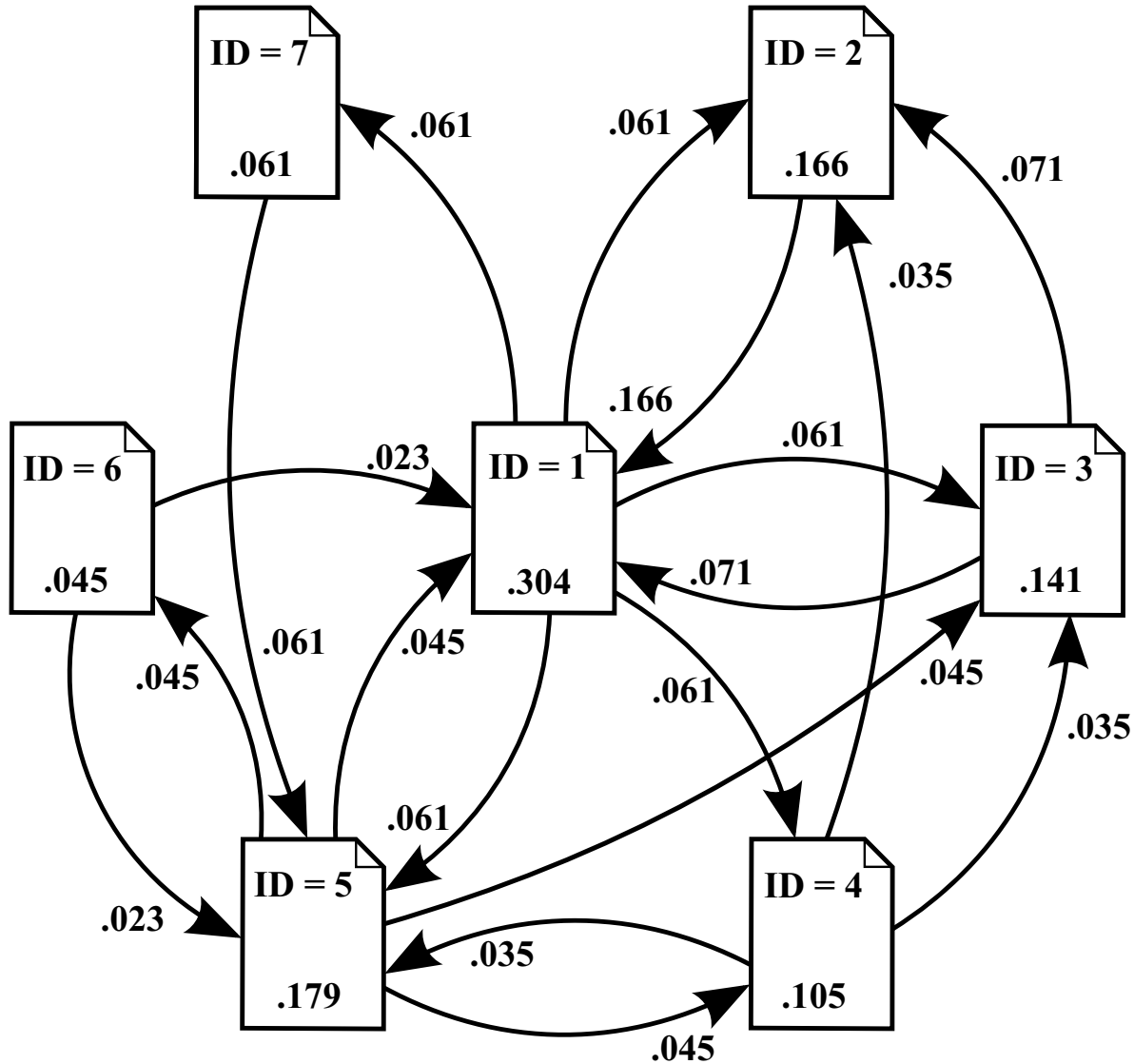
$$A = \begin{bmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{R} = c\mathrm{AR}$$

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

$$\mathrm{A'} = d\mathrm{A} + (1 - d)[1/N]$$

d: damping factor ($= 0.85$)

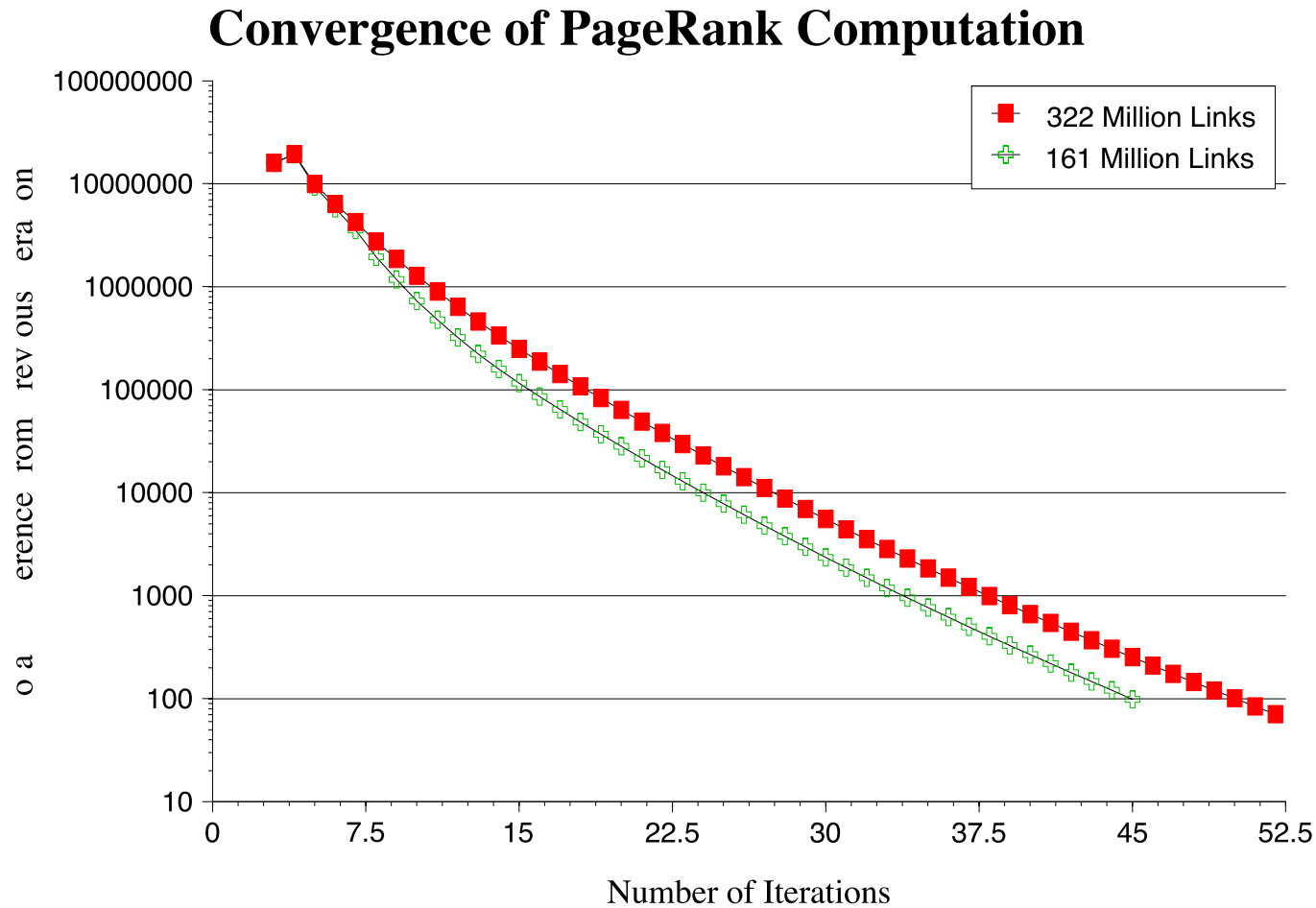
べき乗法による計算

- 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])
                   + (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

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

    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
```

前前回の演習: スパム判定

- 単純ベイズ分類器を使ったスパム判定
  - 「集合知プログラミング 6 章」のコードから作成
  - 日本語を扱うには単語に分割する形態素解析が必要

```
% ruby naivebayes.rb
classifying "quick rabbit" => good
classifying "quick money" => bad
```

前前回の演習: 演習に使う単純ベイズ分類器

出現単語により文書が特定のカテゴリに分類される確率を求める

$$P(C)\prod_{i=1}^{n} P(x_i|C)$$

- $P(C)$: カテゴリの出現確率
- $\prod_{i=1}^{n} P(x_i|C)$: カテゴリにおける各単語の条件付き確率の積

もっとも確率の高いカテゴリを選ぶ

- 閾値： 2 番目のカテゴリより *thresh* 倍高い必要

前前回の演習: スパム判定スクリプト

- トレーニングと判定

```
# create a classifier instance
cl = NaiveBayes.new

# training
cl.train('Nobody owns the water.','good')
cl.train('the quick rabbit jumps fences','good')
cl.train('buy pharmaceuticals now','bad')
cl.train('make quick money at the online casino','bad')
cl.train('the quick brown fox jumps','good')

# classify
sample_data = [ "quick rabbit", "quick money" ]

sample_data.each do |s|
  print "classifying \"#{s}\" => "
  puts cl.classify(s, default="unknown")
end
```

前前回の演習: Classifier Class (1/2)

```
# feature extraction
def getwords(doc)
  words = doc.split(/\W+/)
  words.map!{|w| w.downcase}
  words.select{|w| w.length < 20 && w.length > 2 }.uniq
end

# base class for classifier
class Classifier
  def initialize
    # initialize arrays for feature counts, category counts
    @fc, @cc = {}, {}
  end

  def getfeatures(doc)
    getwords(doc)
  end

  # increment feature/category count
  def incf(f, cat)
    @fc[f] ||= {}
    @fc[f][cat] ||= 0
    @fc[f][cat] += 1
  end

  # increment category count
  def incc(cat)
    @cc[cat] ||= 0
    @cc[cat] += 1
  end

  ...
```

前前回の演習: Classifier Class (2/2)

```
  def fprob(f,cat)
    if catcount(cat) == 0
      return 0.0
    end

    # the total number of times this feature appeared in this
    # category divided by the total number of items in this category
    Float(fcount(f, cat)) / catcount(cat)
  end

  def weightedprob(f, cat, weight=1.0, ap=0.5)
    # calculate current probability
    basicprob = fprob(f, cat)
    # count the number of times this feature has appeared in all categories
    totals = 0
    categories.each do |c|
      totals += fcount(f,c)
    end
    # calculate the weighted average
    ((weight * ap) + (totals * basicprob)) / (weight + totals)
  end

  def train(item, cat)
    features = getfeatures(item)
    features.each do |f|
      incf(f, cat)
    end
    incc(cat)
  end
end
```

前前回の演習: NaiveBayes Class

```
# naive baysian classifier
class NaiveBayes < Classifier
  def initialize
    super
    @thresholds = {}
  end

  def docprob(item, cat)
    features = getfeatures(item)
    # multiply the probabilities of all the features together
    p = 1.0
    features.each do |f|
      p *= weightedprob(f, cat)
    end
    return p
  end

  def prob(item, cat)
    catprob = Float(catcount(cat)) / totalcount
    docprob = docprob(item, cat)
    return docprob * catprob
  end

  def classify(item, default=nil)
    # find the category with the highest probability
    probs, max, best = {}, 0.0, nil
    categories.each do |cat|
      probs[cat] = prob(item, cat)
      if probs[cat] > max
        max = probs[cat]
        best = cat
      end
    end
    # make sure the probability exceeds threshold*next best
```

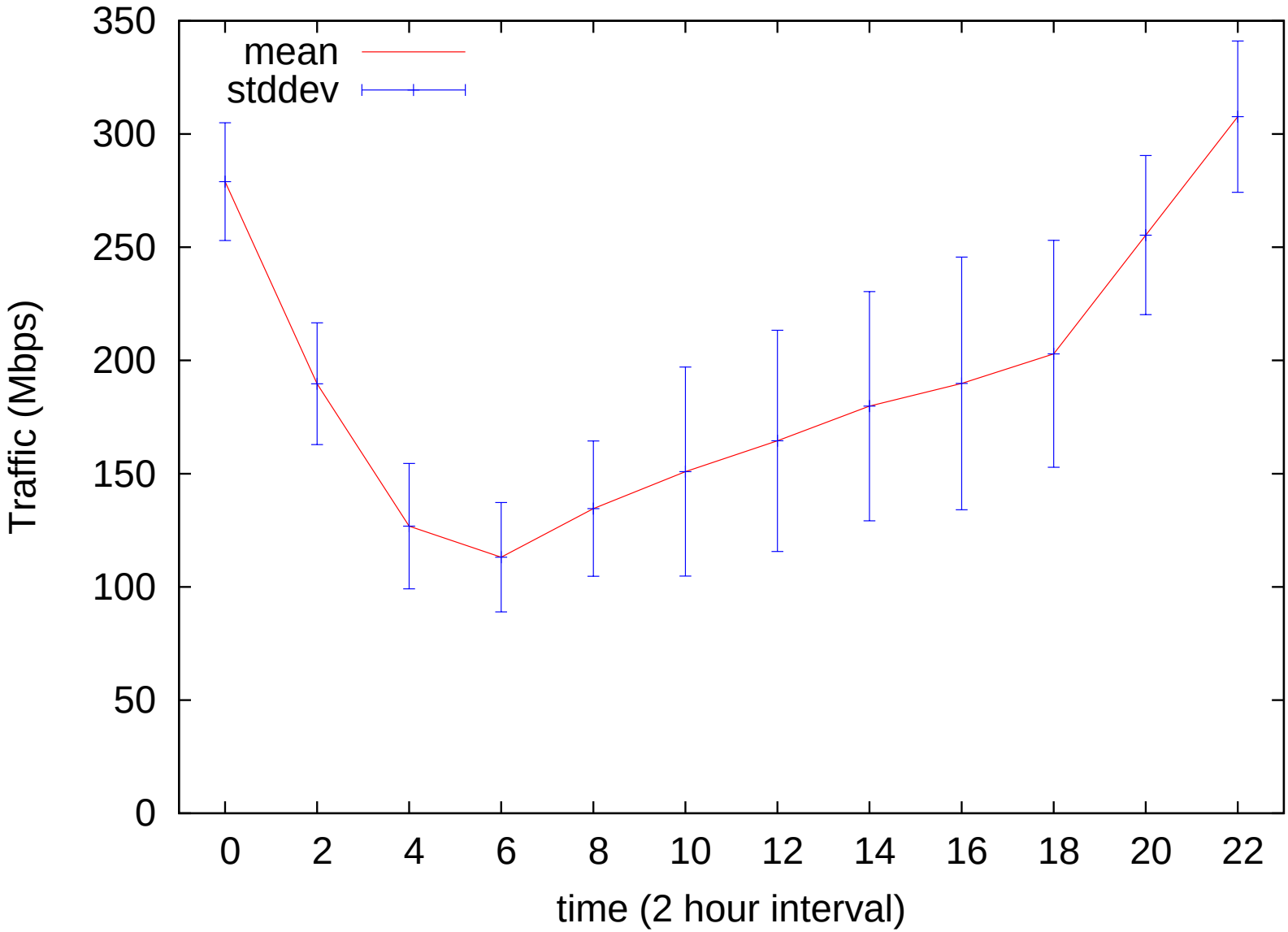
課題2解答: トラフィック解析

- ねらい: 実時系列データから時間帯別や曜日別の情報を抽出
- 課題用データ: ifbps.txt (今回の演習 2 と同じ)
  - あるブロードバンド収容ルータのインターフェイスカウンタ値
  - 2011 年 5 月の 1ヶ月分、2 時間粒度
  - format: time IN(bits/sec) OUT(bits/sec)
- 提出項目
  1. OUT の時間帯別トラフィックプロット
     - 時間毎の平均と標準偏差をプロット
  2. OUT の曜日別時間帯別トラフィックプロット
     - 曜日毎のトラフィックをプロット
  3. OUT の曜日間の相関係数行列テーブル
     - 曜日間の各時間帯平均値を使い相関係数行列を計算
  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: 2012 年 6 月 18 日

いずれの課題も演習スクリプトの正規表現の部分を IN ではなく OUT の値を読むように変更するだけ

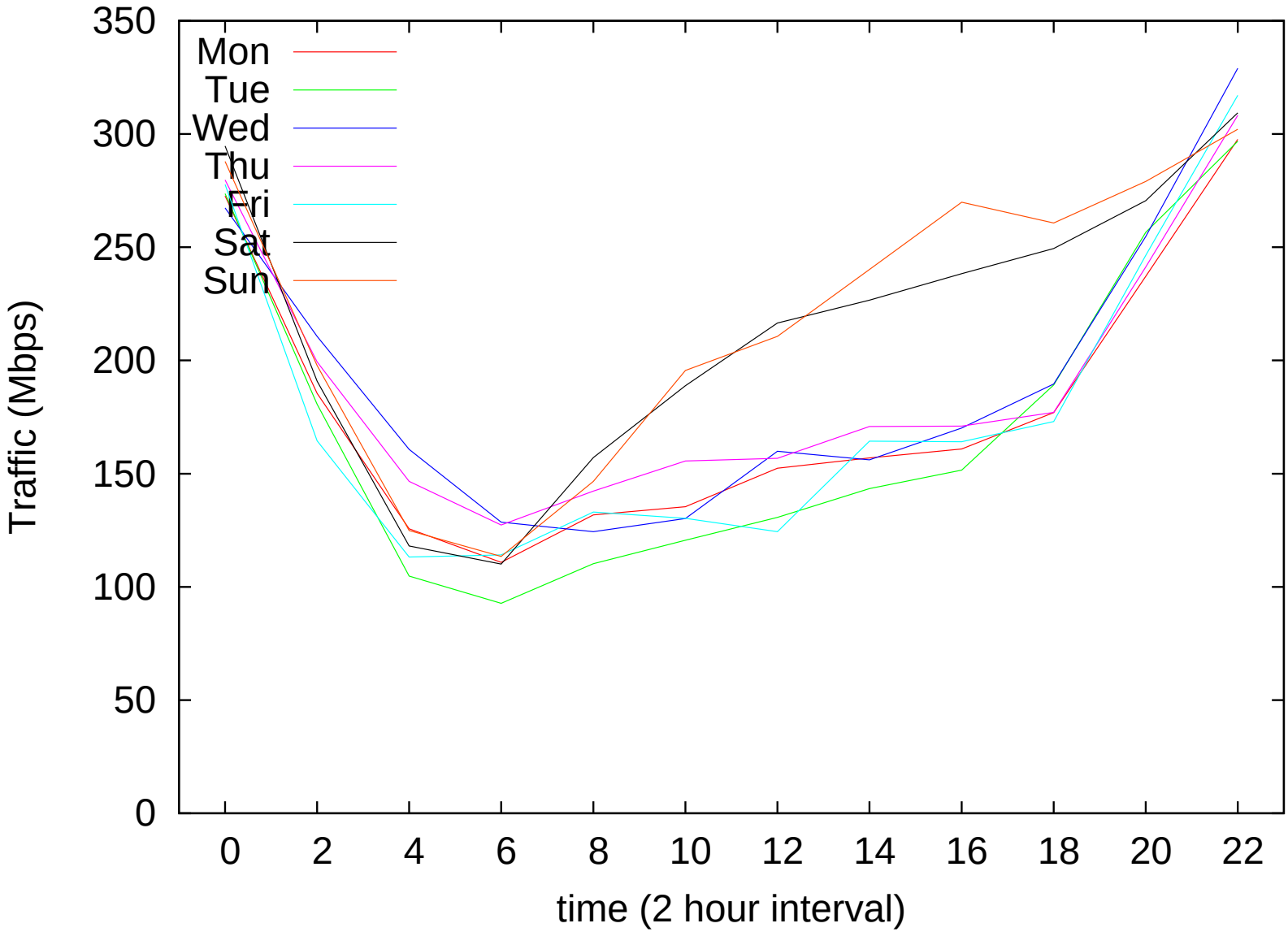
課題 2: 時間帯別トラフィック

- 時間毎の平均と標準偏差をプロット



課題 2: 曜日別時間帯別トラフィック

- 曜日毎のトラフィックをプロット



課題 2: 曜日間の相関係数行列

- 曜日間の相関係数行列を計算
  - 曜日間の各時間帯平均値を使う

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Mon	1.000	0.990	0.975	0.995	0.985	0.880	0.820
Tue	0.990	1.000	0.967	0.978	0.977	0.890	0.844
Wed	0.975	0.967	1.000	0.976	0.954	0.802	0.751
Thu	0.995	0.978	0.976	1.000	0.984	0.848	0.788
Fri	0.985	0.977	0.954	0.984	1.000	0.868	0.816
Sat	0.880	0.890	0.802	0.848	0.868	1.000	0.984
Sun	0.820	0.844	0.751	0.788	0.816	0.984	1.000

最終レポートについて

- A, B からひとつ選択
  - A. Wikipedia の PageRank 計算
  - B. 自由課題
- 8 ページ以内
- pdf ファイルで提出
- 提出〆切: 2012 年 7 月 31 日 (火) 23:59

最終レポート 選択テーマ

A. Wikipedia の PageRank 計算

- データ: wikipedia 英語版のリンクデータ (570 万ページ分)
- A-1 ページの次数分布調査
  - A-1-1 各ページの出次数 (outdegree) の分布を CDF と CCDF でプロットする
  - A-1-2 Wikipedia の出次数分布に関する考察
- A-2 PageRank の計算
  - A-2-1 PageRank を計算し、トップ 30 の結果を表にする
  - A-2-2 その他の解析と結果の考察

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

注意: プログラミングはクラスメートと相談してやっても良いが、協力してやった場合は相手を明記すること。 その場合も考察は自分で考えること。

課題 A. Wikipedia の PageRank 計算

データ: wikipedia 英語版のリンクデータ (570 万ページ分)

- created by Henry Haselgrove (http://haselgrove.id.au/wikipedia.htm)
  - 授業ページにローカルコピーへのリンク
  - テスト用に、10 万ページ分のサブセットデータも用意
- links-simple-sorted.zip: リンクデータ (323MB 展開後 1GB)
  - 各ページは、整数で表現
  - format: $from : to_1, to_2, ...to_n$
- titles-sorted.zip: タイトルデータ (28MB 展開後 106MB)
  - n 行目がページ番号 n のタイトル (1 origin)

```
% head -3 links-simple-sorted.txt
1: 1664968
2: 3 747213 1664968 1691047 4095634 5535664
3: 9 77935 79583 84707 564578 594898 681805 681886 835470 ...
%
% sed -n '2713439p' titles-sorted.txt
Keio-Gijuku_University
```

課題 A-1 ページの次数分布調査

A-1 ページの次数分布調査

- A-1-1 各ページの出次数 (outdegree) の分布を CDF と CCDF でプロットする
  - 次数 0 もカウントすること
- A-1-2 Wikipedia の出次数分布に関する考察
  - オプションでその他の解析など
  - ヒント: 分布は次数が低いページと高いページで傾向が異なる

課題 A-2 PageRank の計算

A-2 PageRank の計算

- A-2-1 PageRank を計算し、トップ 30 の結果を表にする
  - フォーマット: 順位 PageRank 値 ページ ID ページタイトル
  - 演習用スクリプトを利用すればいい
    - damping factor:0.85 thresh:0.000001 を使用すること
  - メモリ 8GB の iMac で約 5 時間 (メモリ 2GB 以下のマシンでは難しい)
- A-2-2 その他の解析と結果の考察
  - オプションでその他の解析など
    - 処理の高速化の工夫
    - PageRank の改良案を実装してみる
  - 結果の考察

まとめ

検索とランキング

- 検索システム
- クローリング
- ページランク
- 演習: PageRank

次回予定

第 13 回 データマイニング (7/6)

- 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