インターネット計測とデータ解析 第13回

長 健二朗

2012年7月6日

前回のおさらい

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

今日のテーマ

データマイニング

- 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

データマイニング

- 膨大なデータ
  - 従来の手法では把握しきれない
  - データの中に隠れた情報を抽出する必要
- Data Mining
  - 膨大なデータ、かつ、多次元、多様、分散などの特徴
  - 手法は機械学習、AI、パターン認識、統計、データベースなどからアイデア
- クラウド技術などで大量データ処理が現実的に

Data Mining 手法のいろいろ

- パターン抽出: データが内包する規則や特徴的なパターンを見つける
  - 相関
  - 時系列
- 分類: オリジナル情報にない分類を機械的に実現
  - ルールベース
  - 単純ベイズ分類器
  - ニューラルネットワーク
  - サポートベクターマシン (SVM)
  - 次元減少 (主成分分析, PCA)
- クラスタリング: 変量間の距離 (類似度) を計算しグループ化
  - 距離ベース、密度ベース、グラフベース
  - k-means、DBSCAN
- 異常検出: 統計手法を使って定常状態からのずれを検出
  - 単変数、多変数
  - 外れ値検出

距離について (復習)

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

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

標準化ユークリッド距離 (standardized Euclidean distance)

- 変数間でばらつきの大きさが異なると、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

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

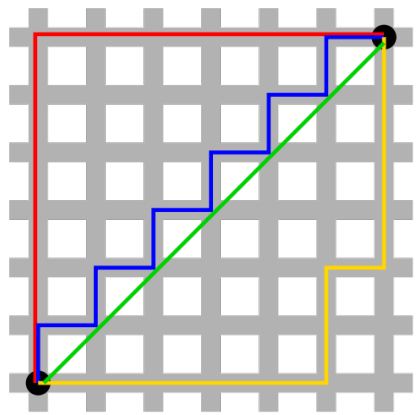
ミンコフスキー距離 (Minkowski distance)

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

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

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

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値表現

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に出現しない ⇒ 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

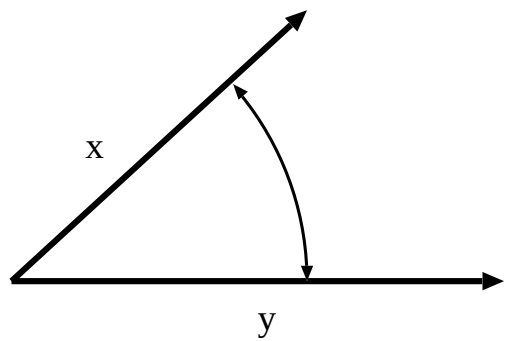
- 文書の類似度で出現頻度も考慮する場合など

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 ⇒ 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

$$x = 3 \quad 2 \quad 0 \quad 5 \quad 0 \quad 0 \quad 0 \quad 2 \quad 0 \quad 0$$
$$y = 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 2$$

$$x \cdot y = 3 * 1 + 2 * 1 = 5$$
$$\|x\| = \sqrt{3 * 3 + 2 * 2 + 5 * 5 + 2 * 2} = \sqrt{42} = 6.481$$
$$\|y\| = \sqrt{1 * 1 + 1 * 1 + 2 * 2} = \sqrt{6} = 2.449$$

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

クラスタリング

複雑な関係を持つデータの分類に欠かせない技術

多変量データの変量間の距離 (類似度) を計算しグループ化

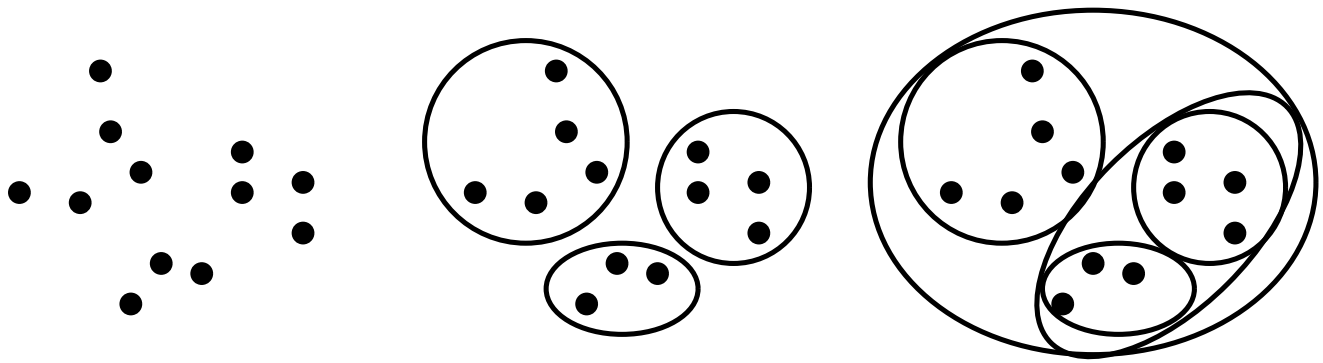
- データを分類し理解する
- データを要約する

さまざまな応用

- ビジネス: 顧客をグループ化しマーケティングに応用
- 気象情報: 複雑な気象要因からパターンを見つける
- バイオ: 遺伝子やタンパクの分類
- 医療や薬学: 症状や作用の複雑な相互関係

クラスタリング手法

- 分割型クラスタリング (patitional clustering)
  - k-means 法
- 階層型クラスタリング (hierarchical clustering)
  - MST 法
  - DBSCAN 法



original points　partitional clustering　hierarchical clustering

k-means 法

- 分割型クラスタリング
- クラスタ数 k を指定
- 基本アルゴリズムはシンプル
  - 各クラスタは重心 (centroid) を持つ (通常は平均)
  - 各データを最も近い重心を持つクラスタに割り当てる
  - データの割り当てと重心の再計算を繰り返す
- 制約
  - 事前にクラスタ数 k を指定する必要
  - 初期値によって結果が変わる
  - クラスタが異なるサイズ、密度をもつ場合や円形でない場合
  - 外れ値の影響が大きい

basic k-means algorithm:
1: select k points randomly as the initial centroids
2: **repeat**
3: form k clusters by assigning all points to the closest controid
4: recompute the centroid of each cluster
5: **until** the centroids don't change

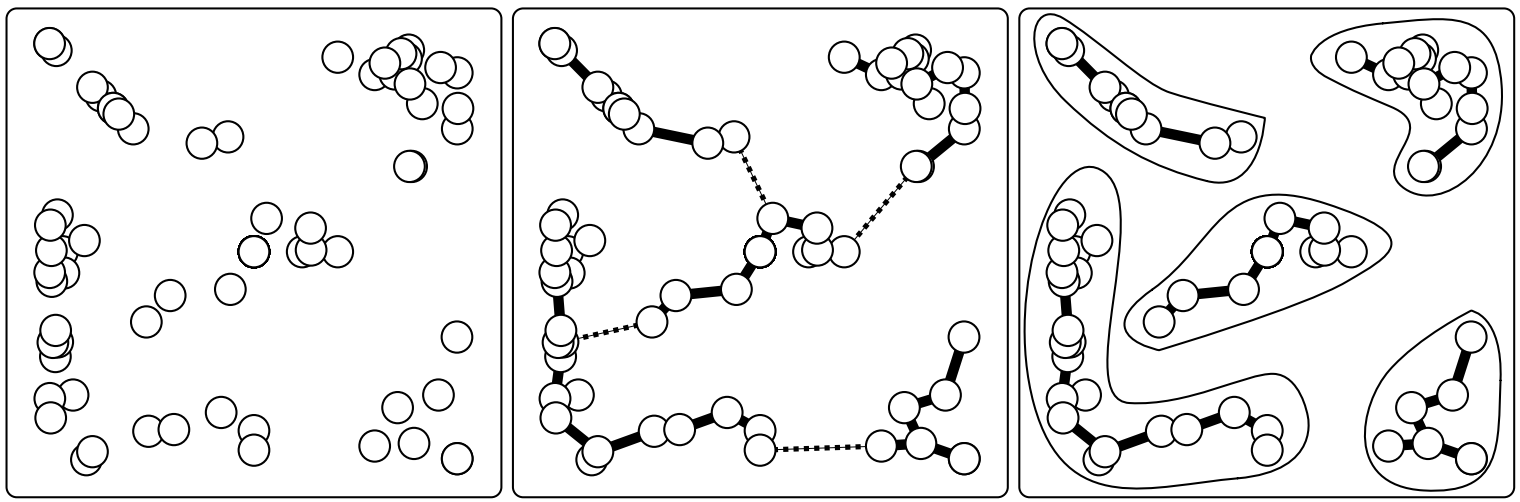
階層型クラスタリング

- ツリー構造でクラスタを生成
  - ツリー構造でクラスタ構成が説明可能
- 事前にクラスタ数を指定する必要がない
- 2 種類のアプローチ
  - 凝集型: 各データを 1 クラスタとして、統合していく
  - 分割型: 全体を 1 クラスタとして始め、分割していく

MST クラスタリング

Minimum Spanning Tree クラスタリング

- 分割型の階層型クラスタリング
- 任意の点からスタートしスパニングツリーを作る
- 距離の長いエッジから削除してクラスタを分割していく



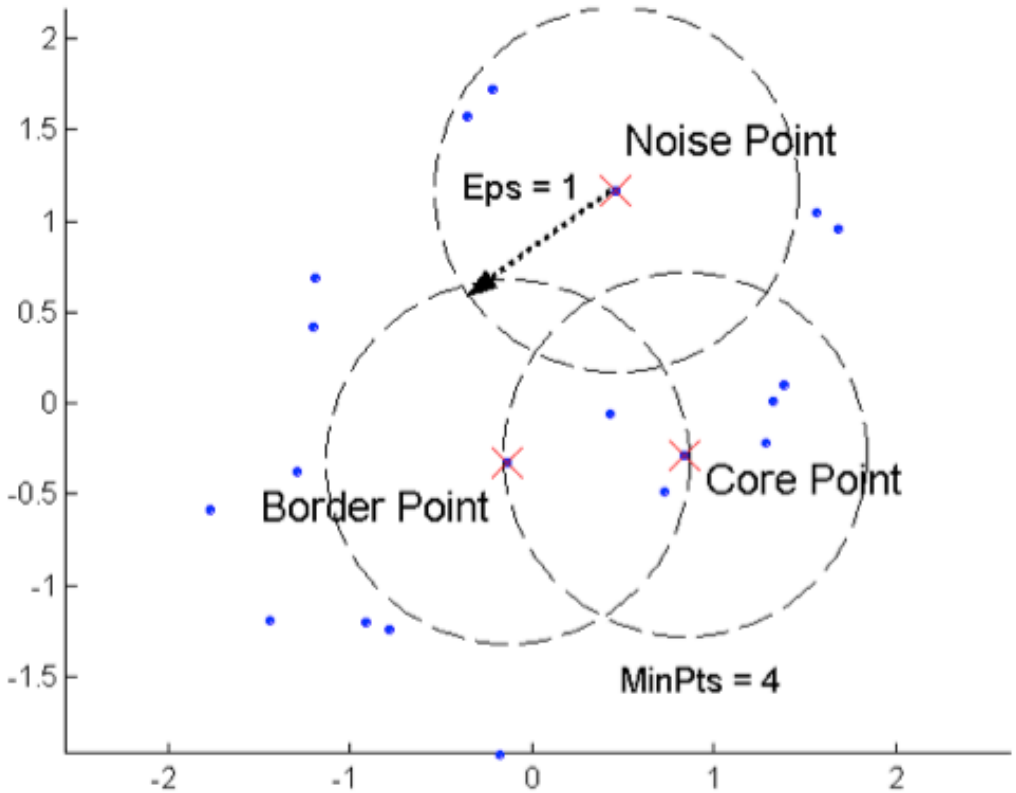
DBSCAN

Density-Based Spatial Clustering

- 密度: 指定した距離内のデータ数
- (球状でない) 任意形状のクラスタの抽出が可能
- ノイズに強い
- 距離の閾値 *Eps* と数の閾値 *MinPts*
  - Core points: 距離 *Eps* 内に *MinPts* 以上の近傍点がある
  - Border points: Core ではないが、距離 *Eps* 内に Core が存在
  - Noise points: 距離 *Eps* 内に Core が存在しない
- 弱点: 密度が異なるクラスタや次数の多いデータ

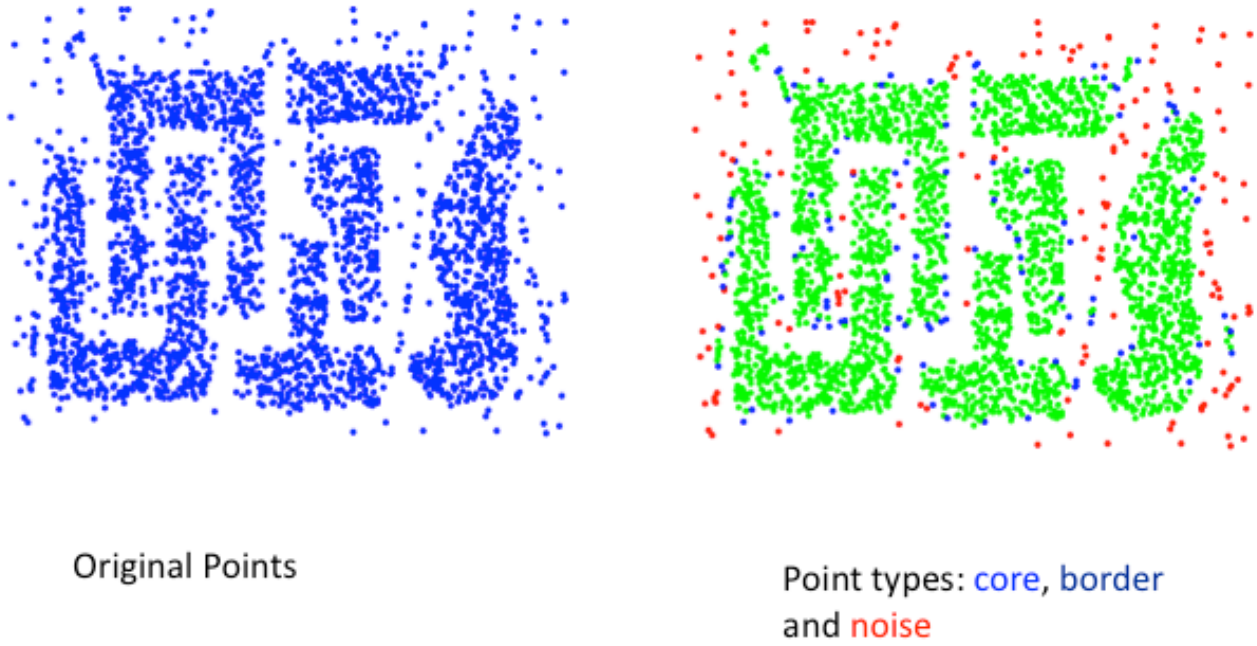
DBSCAN algorithm:
1: label all points as core, border, or noise points
2: eliminate noise points
3: put an edge between all core points that are within *Eps* of each other
4: make each group of connected core points into a separate cluster
5: assign each border point to one of the clusters of its associated core points

DBSCAN: Core, Border, and Noise Points



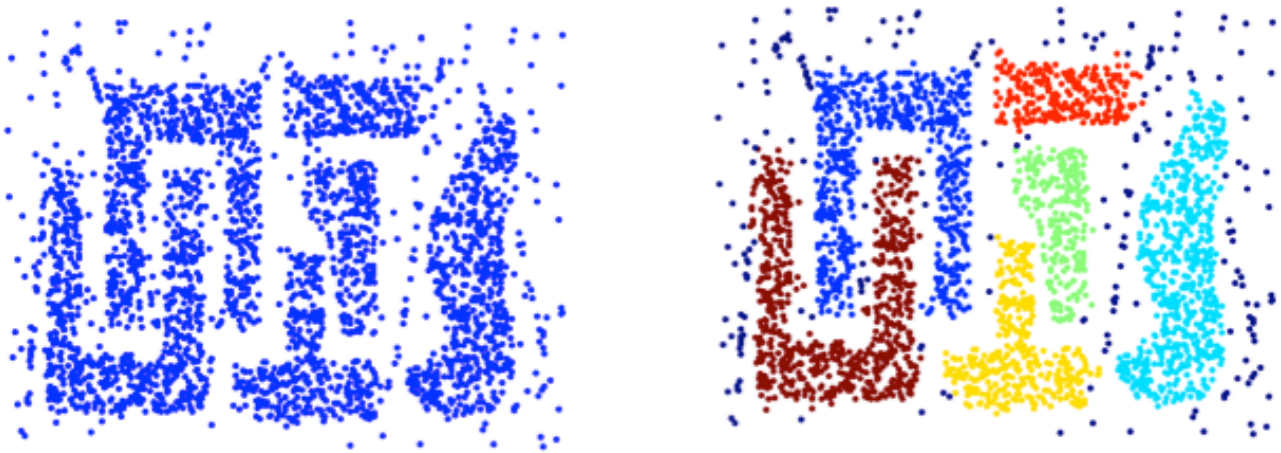
source: Tan, Steinbach, Kumer. Introduction to Data Mining

DBSCAN: example of Core, Border, and Noise Points



source: Tan, Steinbach, Kumer. Introduction to Data Mining

DBSCAN: example clusters

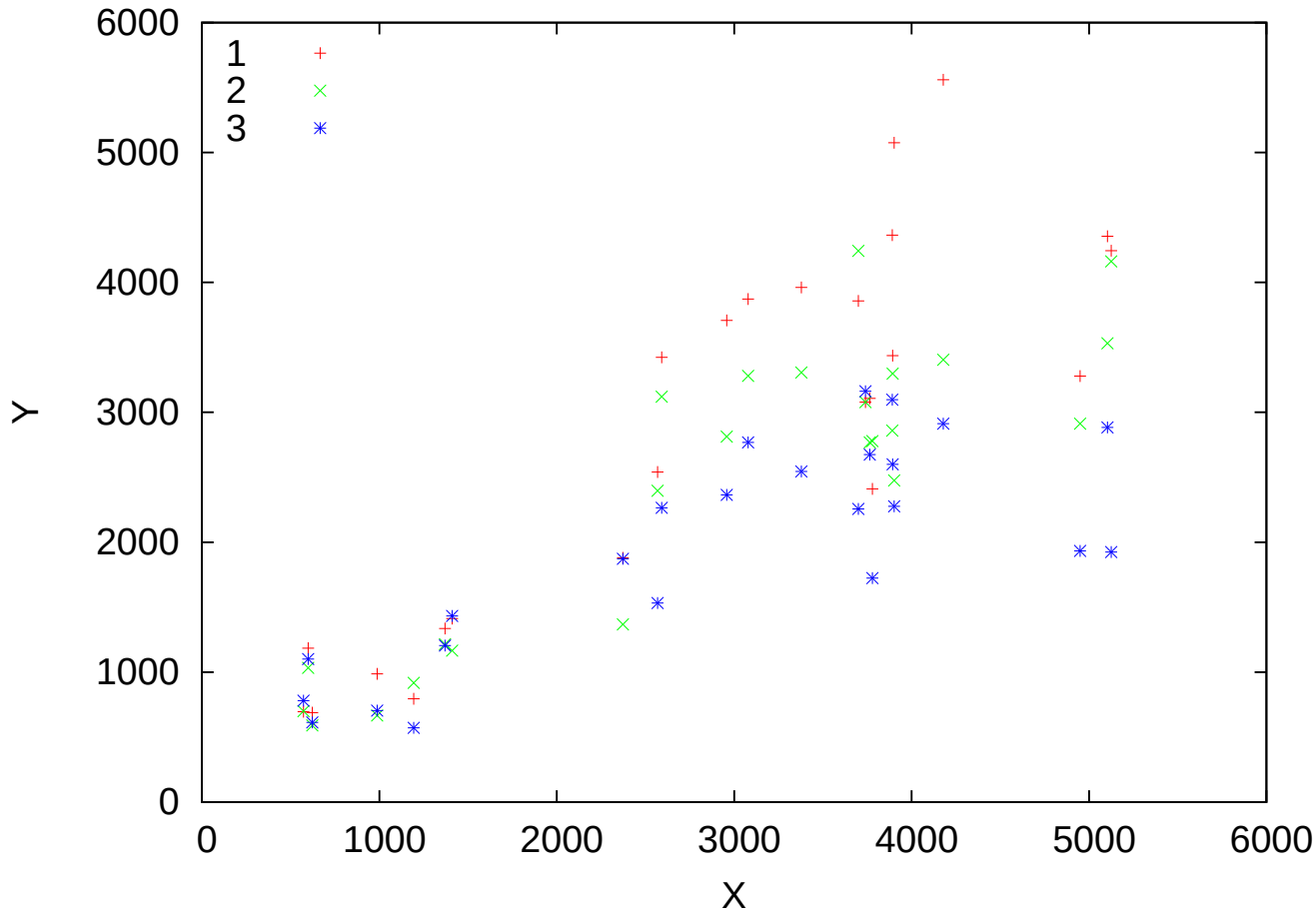


Clusters

source: Tan, Steinbach, Kumer. Introduction to Data Mining

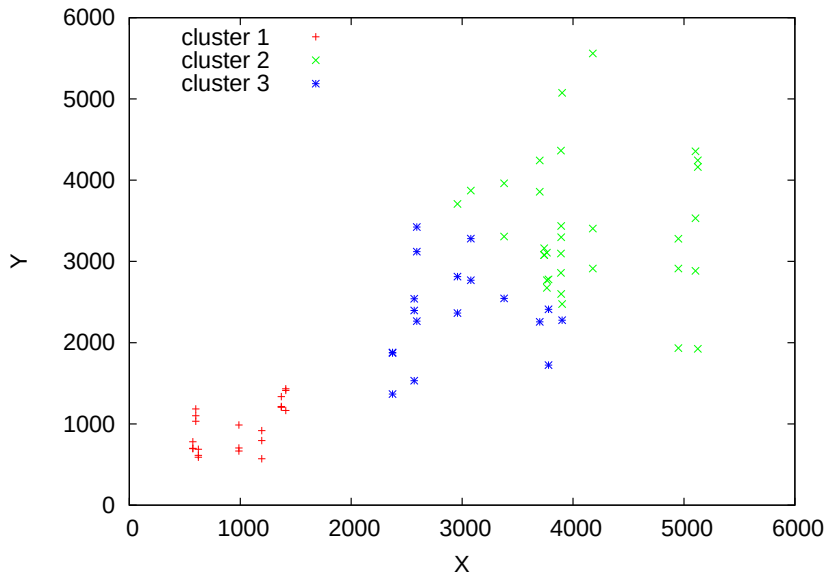
演習: k-means clustering

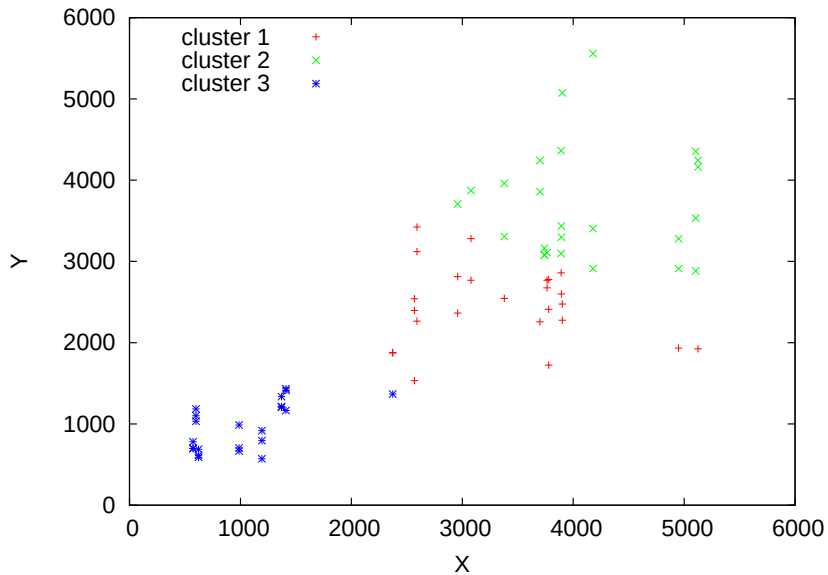
```
% ruby k-means.rb km-data.txt > km-results.txt
```



k-means clustering 結果

- 初期値によって結果に違いがでる





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 dsistance 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
```

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 スクリプト

```
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
```

前回の演習: 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
```

前回の演習: 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
```

前回の演習: 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
```

前回の演習: 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
```

前回の演習: 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
```

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

データマイニング

- 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

次回予定

第 14 回 スケールする計測と解析 (7/13)

- 大規模計測
- MapReduce
- 分散並列処理
- クラウド技術
- インターネット計測とプライバシー
- 演習: 並列処理