インターネット計測とデータ解析 第 11 回

長 健二朗

2014 年 6 月 23 日

前回のおさらい

第 10 回 異常検出と機械学習 (6/16)

- 異常検出
- 機械学習
- スパム判定とベイズ理論
- 演習: 単純ベイズ分類器

今日のテーマ

第 11 回 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 (分類は事前に定義)
  - 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): ベクトルのノルム
- マハラノビス距離 (Mahalanobis distance)

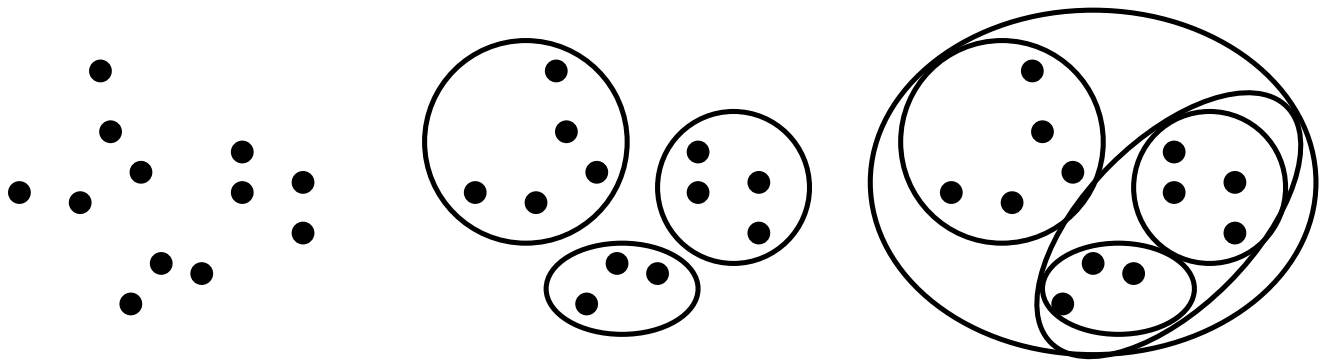
類似度

- バイナリベクトルの類似度: Jaccard 係数
- n 次元ベクトルの類似度: コサイン類似度

クラスタリング手法

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
- 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

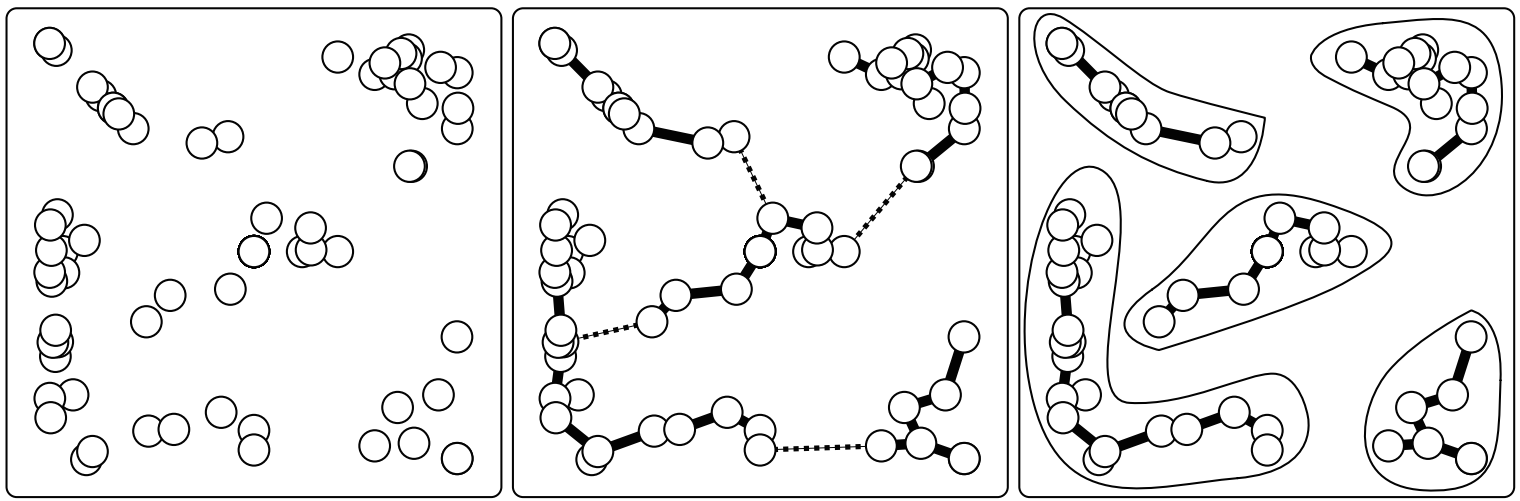
階層型クラスタリング

- 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

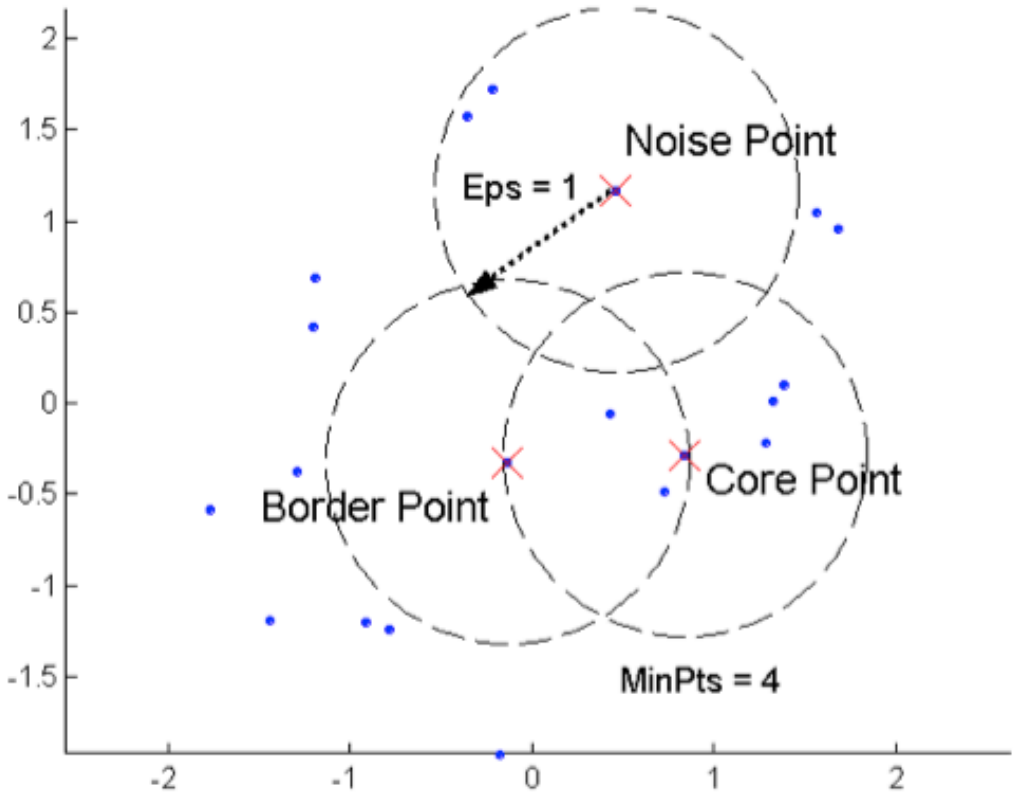
Density-Based Spatial Clustering

- 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:

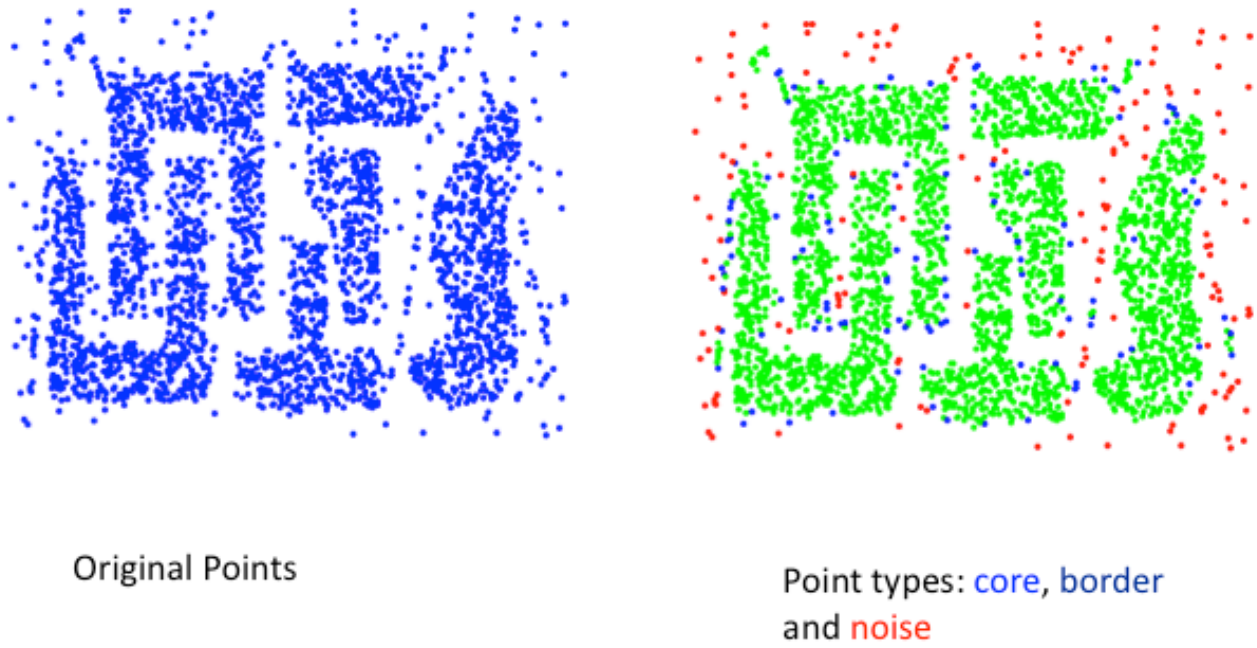
1: label all points as core, border, or noise points
2: eliminate noise points
3: put an edge between all core points that are within Eps of each other
4: make each group of connected core points into a separate cluster
5: assign each border point to one of the clusters of its associated core points

DBSCAN: Core, Border, and Noise Points



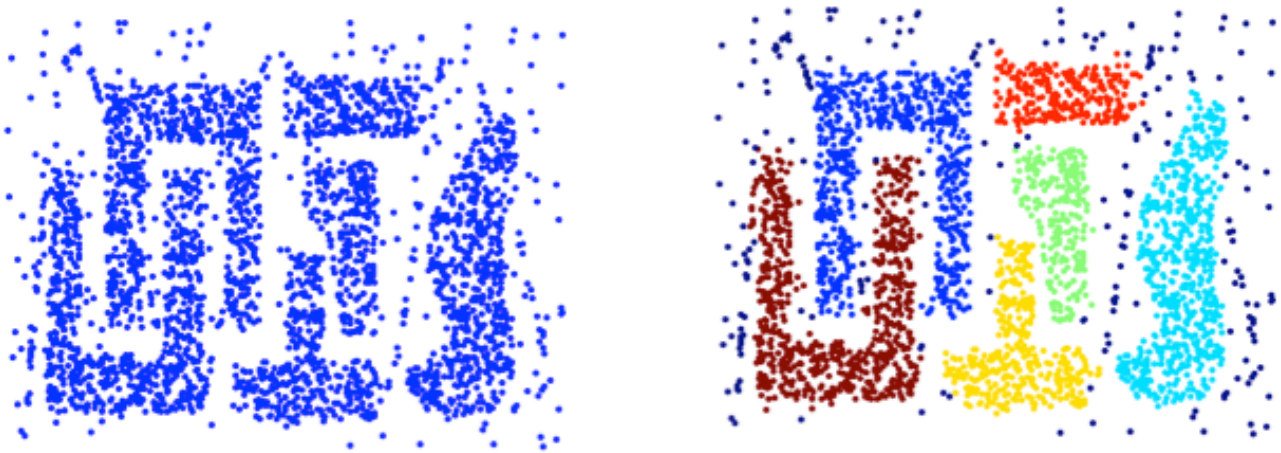
source: Tan, Steinbach, Kumer. Introduction to Data Mining

DBSCAN: example of Core, Border, and Noise Points



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

DBSCAN: example clusters

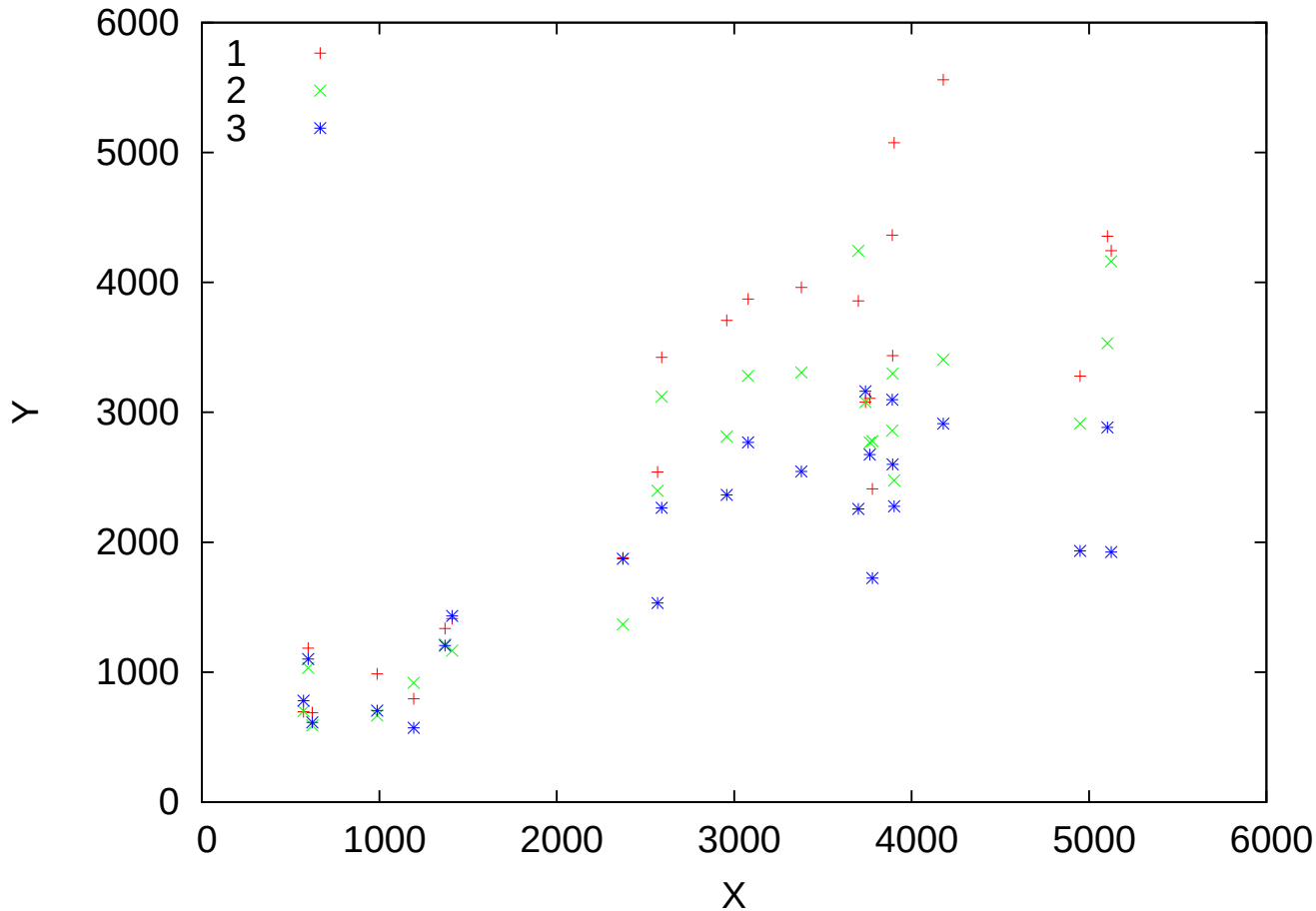


Clusters

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

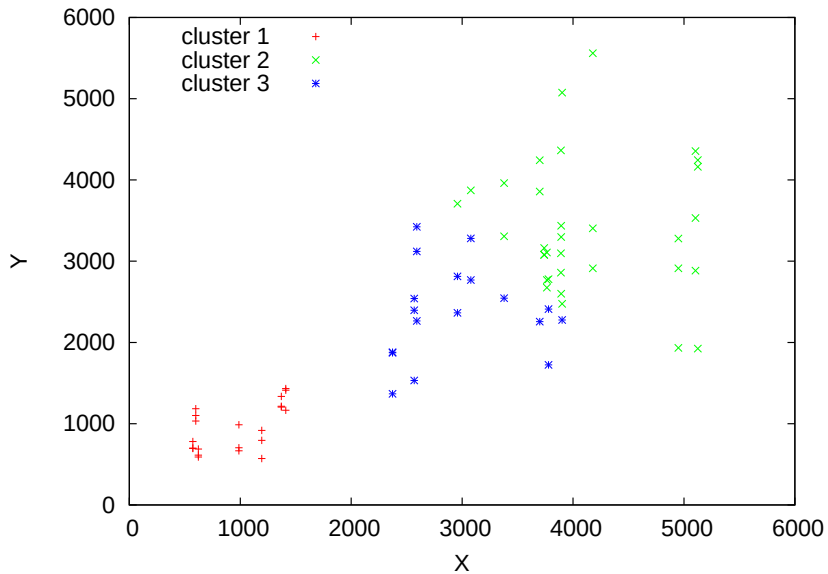
演習: k-means clustering

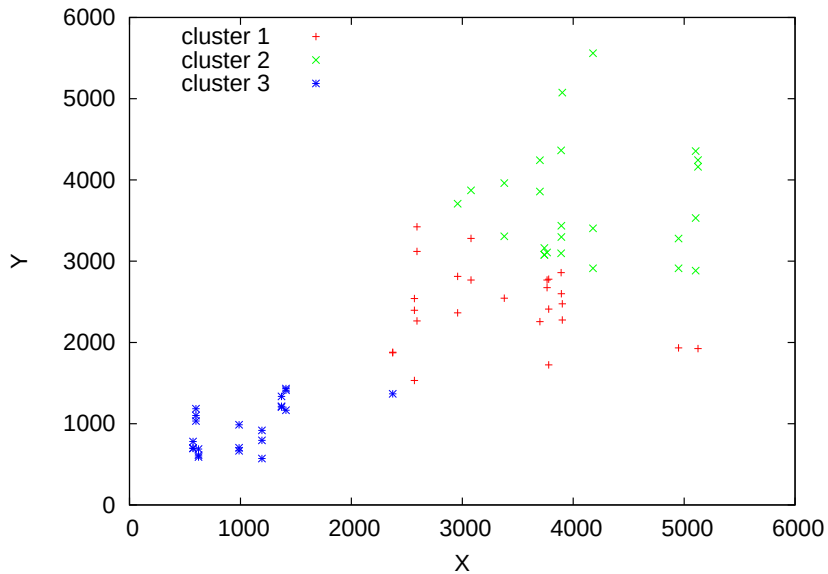
```
% ruby k-means.rb km-data.txt > km-results.txt
```



k-means clustering 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 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 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
```

課題 2: twitter データ解析

- ねらい: 大規模実データ処理の実践
- 課題用データ:
  - Kwak らによる 2009 年 7 月の twitter data、約 4000 万ユーザ分
    - 元データ: http://an.kaist.ac.kr/traces/WWW2010.html
  - twitter_degrees-10000.txt (135KB)
    - 10,000 人分のサンプルデータ
  - twitter_degrees.zip (164MB, 解凍後 550MB)
    - 約 4000 万人分のフルデータ
  - numeric2screen.zip (365MB, 解凍後 756MB)
    - ユーザ ID とスクリーン名のマッピング
- 提出項目
  1. twitter ユーザの following/follower 数散布図プロット
     - 10,000 人分のデータを使った散布図
  2. フルデータによる following/follower 数分布の CCDF プロット
     - X 軸に following/follower 数を取り log-log プロット
  3. フォローワ数の多いトップ 50 ユーザの表
     - ランク、ユーザ ID、スクリーン名、フォロー数、フォローワ数
  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: PDF 形式のレポートを SFC-SFS から提出
- 提出〆切: 2014 年 6 月 18 日

課題データについて

twitter_degrees.zip (164MB, 解凍後 550MB)

```
# id followings followers

12      586     1001061
13      243     1031830
14      106     8808
15      275     14342
16      273     218
17      192     6948
18      87      6532
20      912     1213787
21      495     9027
22      272     3791
...
```

numeric2screen.zip (365MB, 解凍後 756MB)

```
# id screenname

12 jack
13 biz
14 noah
15 crystal
16 jeremy
17 tonystubblebine
18 Adam
20 ev
21 dom
22 rabble
...
```

課題 提出物

散布図

- 10,000 人分のデータを用いて、X 軸に following、Y 軸に follower 数を取り log-log プロット

CCDF プロット

- X 軸に following/follower 数を取り log-log プロット

フォローワ数の多いトップ 50 ユーザの表

- ランク、ユーザ ID、スクリーン名、フォロー数、フォローワ数

```
# rank   id              screenname    followings followers

1       19058681        aplusk          183     2997469
2       15846407        TheEllenShow    26      2679639
3       16409683        britneyspears   406238  2674874
4       428333          cnnbrk          18      2450749
5       19397785        Oprah           15      1994926
6       783214          twitter         55      1959708
...
```

sort コマンド

sort コマンド: テキストファイルの行をソートして並び替える

```
$ sort [options] [FILE ...]
```

- options (課題で使いそうなオプション)
  - -n : フィールドを数値として評価
  - -r : 結果を逆順に並べる
  - -k POS1[,POS2] : ソートするフィールド番号 (1 オリジン) を指定する
  - -t SEP : フィールドセパレータを指定する
  - -m : 既にソートされたファイルをマージする
  - -T DIR : 一時ファイルのディレクトリを指定する

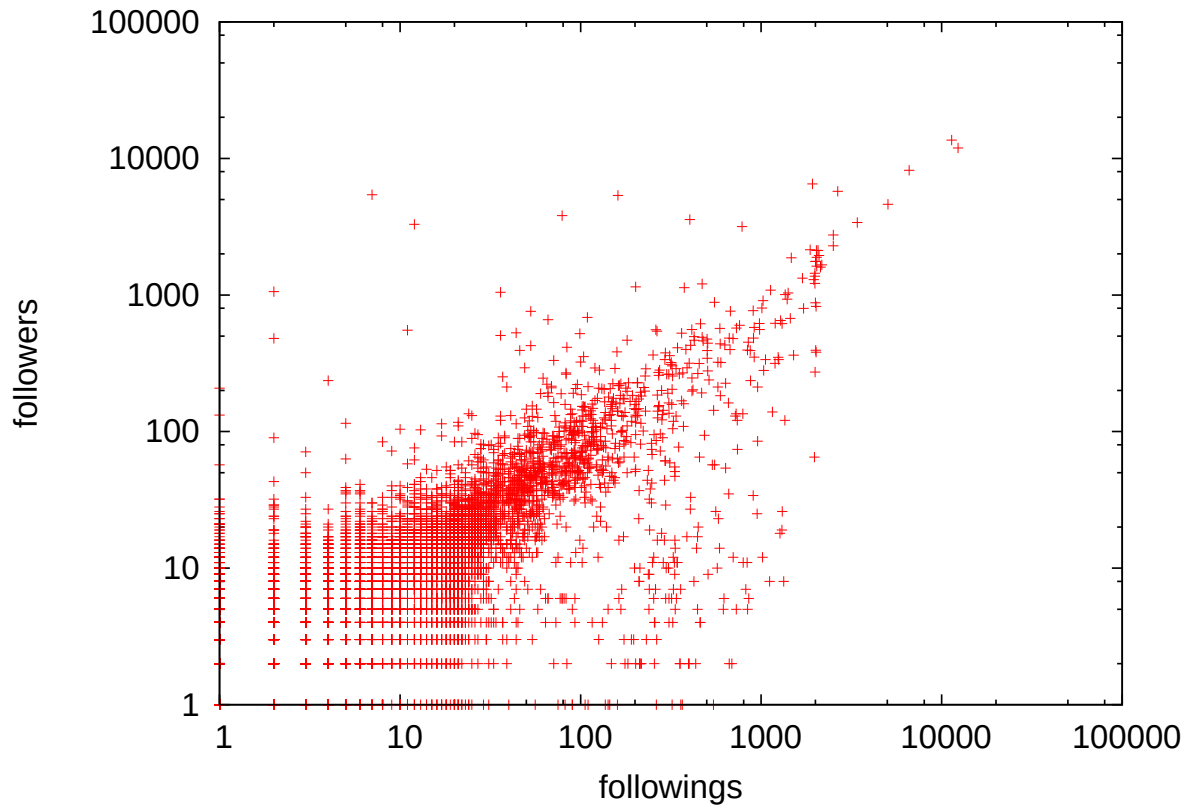
例: file を第 3 フィールドを数値とみて逆順にソート、一時ファイルは”/usr/tmp”に作る

```
$ sort -nr -k3,3 -T/usr/tmp file
```

課題 2 解答: twitter ユーザの following/follower 数散布図

散布図

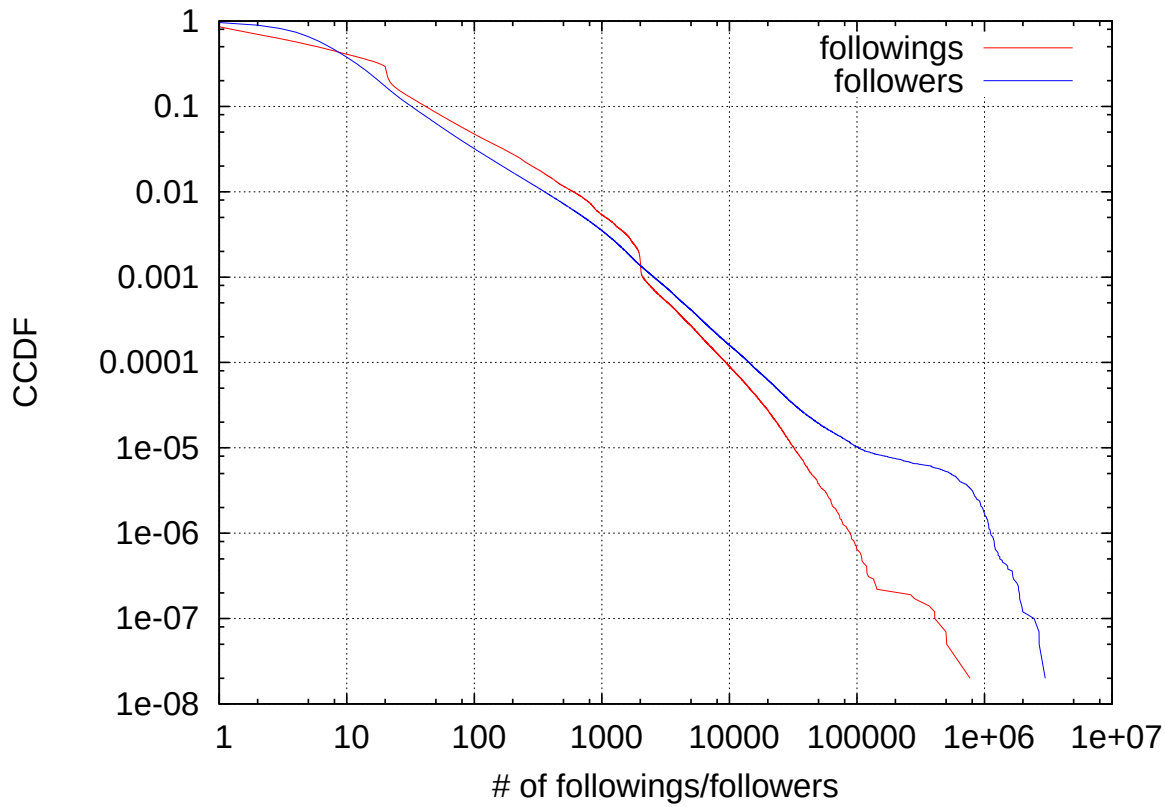
- 10,000 人分のデータを用いて、X 軸に following、Y 軸に follower 数を取り log-log プロット



課題 2 解答: CCDF プロット

CCDF プロット

- X 軸に following/follower 数を取り log-log プロット



課題 2 解答: フォローワ数トップ 50 ユーザの表 (1/2)

フォローワ数の多いトップ 50 ユーザの表

- ランク、ユーザ ID、スクリーン名、フォロー数、フォローワ数

# rank	id	screenname	followings	followers
1	19058681	aplusk	183	2997469
2	15846407	TheEllenShow	26	2679639
3	16409683	britneyspears	406238	2674874
4	428333	cnnbrk	18	2450749
5	19397785	Oprah	15	1994926
6	783214	twitter	55	1959708
7	16190898	RyanSeacrest	137	1885782
8	813286	BarackObama	770155	1882889
9	19757371	johncmayer	64	1844499
10	17461978	THE_REAL_SHAQ	563	1843561
11	25365536	KimKardashian	73	1790771
12	19554706	mrskutcher	99	1691919
13	15485441	jimmyfallon	131	1668193
14	18220175	iamdiddy	173	1657119
15	16727535	lancearmstrong	103	1651207
16	807095	nytimes	177	1524048
17	18863815	coldplay	2633	1517067
18	27104736	mileycyrus	54	1477423
19	14075928	TheOnion	369569	1380160
20	17220934	algore	8	1377332
21	18091904	ashleytisdale	75	1318909
22	18222378	50cent	13	1318378
23	20536157	google	162	1278103
24	21879024	tonyhawk	118	1277163
25	19329393	PerezHilton	328	1269341

課題 2 解答: フォロー数トップ 50 ユーザの表 (2/2)

# rank	id	screenname	followings	followers
26	16827333	souljaboytellem	94	1241331
27	20	ev	912	1213787
28	972651	mashable	1934	1210996
29	26885308	ashsimpsonwentz	32	1200472
30	6273552	MCHammer	27413	1195089
31	5741722	nprpolitics	119716	1193870
32	19877186	chelsealately	11	1191340
33	14293310	TIME	79	1189952
34	6211972	SaraBareilles	9	1186592
35	21324258	MarthaStewart	32	1177233
36	19637934	rainnwilson	126	1176930
37	16264006	petewentz	80	1164295
38	14515734	drdrew	151	1159766
39	2883841	eonline	33	1129659
40	19394188	SenJohnMcCain	64	1123254
41	7400702	BBCClick	12	1119182
42	19248106	MariahCarey	22	1116010
43	15131310	WholeFoods	498700	1112628
44	657863	kevinrose	167	1110063
45	14224719	DowningStreet	505613	1105469
46	19923144	NBA	718	1103534
47	16998020	lilyroseallen	49	1083786
48	5248441	AFineFrenzy	119	1081761
49	7861312	feliciaday	110	1079749
50	17266725	tonyrobbins	217	1078763

課題 2 の考察

散布図から分かる事

- フォロー数とフォロワー数が非対象なユーザが存在
- 大半のユーザは 200 以下、一方で非常に多いユーザも存在
- 多くのユーザは重なってプロットされているため散布図だけでは分布が分からない

CCDF から分かる事

- フォロー数、フォロワー数ともに概ねべき分布に従う
  - 極端に少ない/多い部分はその限りでない
  - 両方とも 10 万を越えるテイル部分は様子が異なる特殊なユーザ
- フォロー数は、20 と 2000 あたりにギャップ
  - 20: 初期設定でフォローする 20 人を薦められる
  - 2000: 以前は最大 2000 人までしかフォローできなかった

フォローワ数の多いトップ 50 ユーザの表

- 巨大なデータのマージ作業の演習
  - 方法 1: 事前に共通項目である ID でソートしてマージ
  - 方法 2: トップ 50 を先に抜き出しておいて、screenname とマッチ

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

  # when the sample size is small, fprob is not reliable.
  # so, make it start with 0.5 and converge to fprob as the number grows
  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
```

debug: feature probabilities のダンプ

トレーニング後の内部状態:

```
fprob for "nobody":    good:0.333 bad:0.000
fprob for "owns":      good:0.333 bad:0.000
fprob for "the":       good:1.000 bad:0.500
fprob for "water":     good:0.333 bad:0.000
fprob for "quick":     good:0.667 bad:0.500
fprob for "rabbit":    good:0.333 bad:0.000
fprob for "jumps":     good:0.667 bad:0.000
fprob for "fences":    good:0.333 bad:0.000
fprob for "buy":       good:0.000 bad:0.500
fprob for "pharmaceuticals":   good:0.000 bad:0.500
fprob for "now":       good:0.000 bad:0.500
fprob for "make":      good:0.000 bad:0.500
fprob for "money":     good:0.000 bad:0.500
fprob for "online":    good:0.000 bad:0.500
fprob for "casino":    good:0.000 bad:0.500
fprob for "brown":     good:0.333 bad:0.000
fprob for "fox":       good:0.333 bad:0.000
```

まとめ

第 11 回 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 12 回 検索とランキング (6/30)

- 検索システム
- ページランク
- 演習: PageRank