

Internet Measurement and Data Analysis (11)

Kenjiro Cho

2013-12-18

review of previous class

Class 10 Anomaly detection and machine learning (12/11)

- ▶ Anomaly detection
- ▶ Machine Learning
- ▶ SPAM filtering and Bayes theorem
- ▶ exercise: naive Bayesian filter

today's topics

Class 11 Data Mining

- ▶ Pattern extraction
- ▶ Classification
- ▶ Clustering
- ▶ exercise: clustering

data mining

- ▶ huge volume of data
 - ▶ difficult to handle with traditional methods
 - ▶ need to extract information hidden in data that is not readily evident
- ▶ Data Mining
 - ▶ huge volume, multi-dimensional diverse data, non-trivial distributions
 - ▶ methods often derived from ideas in machine learning, AI, pattern recognition, statistics, database, signal processing
- ▶ data processing becomes practical by growing computing power (e.g., cloud computing)

Data Mining methods

definition: non-trivial extraction of implicit, previously unknown and potentially useful information from data

- ▶ pattern extraction: find existing models and patterns in data
 - ▶ correlation
 - ▶ time-series
- ▶ classification: automatically create new classes that do not exist in the original data
 - ▶ rule-based methods
 - ▶ naive Bayesian filter
 - ▶ neural networks
 - ▶ support vector machine (SVM)
 - ▶ dimensionality reduction (e.g., PCA)
- ▶ clustering: compute the distance (or similarity) between data points and group them
 - ▶ distance based, density based, graph based
 - ▶ k-means, DBSCAN
- ▶ anomaly detection: find deviation from normal state using statistical methods
 - ▶ univariate, multivariate
 - ▶ outlier detection

distances (review)

various distances

- ▶ Euclidean distance
- ▶ standardized Euclidean distance
- ▶ Minkowski distance
- ▶ Mahalanobis distance

similarities

- ▶ binary vector similarities
- ▶ n-dimensional vector similarities

properties of distance

a metric of distance $d(x, y)$ between 2 points (x, y) in space
positivity

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

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

symmetry

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

triangle inequality

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

Euclidean distance

word “distance” usually means “Euclidean distance”
a distance of 2 points (x, y) in a n -dimensional space

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

standardized Euclidean distance

- ▶ when variances are different among variables, distances are affected.
- ▶ standard Euclidean distance: normalized by dividing the Euclidean distance by the variance of each variable

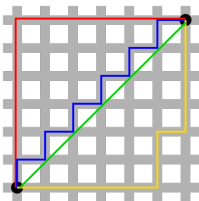
$$d(x, y) = \sqrt{\sum_{k=1}^n \frac{(x_k - y_k)^2}{s_k^2}}$$

Minkowski distance

generalization of Euclidean distance: as parameter r grows, a short cut crossing different axes is preferred more

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

- ▶ $r = 1$: Manhattan distance
 - ▶ Hamming distance: for 2 strings of equal length, the number of positions at which the corresponding symbols are different.
 - ▶ example: the hamming distance of 111111 and 101010 is 3
- ▶ $r = 2$: Euclidean distance



Manhattan distance vs. Euclidean distance

vector norm (1/2)

vector norm: the length of a vector

$$\|x\| \text{ where } x \text{ is a vector}$$

the l_n -norm of x is defined by Minkowski distance as

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

l_0 -norm: the total number of non-zero elements in a vector

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

l_1 -norm: sum of absolute difference

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

l_2 -norm: Euclidean distance

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

l_∞ -norm: the maximum entry's magnitude of a vector

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

vector norm (2/2)

For the example vector $x = (1, 2, 3)$

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

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

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

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

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

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



unit circles of l_p -norm with various values of p

Mahalanobis distance

a distance that takes correlations into account, when correlation exists between variables

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

here, Σ^{-1} is the inverse matrix of its covariance matrix

similarities

similarity

- ▶ numerical measure of how alike 2 data objects are

properties of similarity

positivity

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

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

symmetry

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

in general, triangle inequality does not apply to similarities

similarity between binary vectors

Jaccard coefficient

- ▶ used for similarity between binary vectors in which the occurrences of 1 is much smaller than the occurrences of 0
- ▶ example: as a metric of similarity by occurrences of words in documents
- ▶ many words do not appear in both documents \Rightarrow not considered
- ▶ the following table shows the relationship of each item

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

Jaccard coefficient:

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

similarity between vectors

similarity between (non-binary) vectors

- ▶ example: similarity of documents where frequencies of words are also taken into consideration

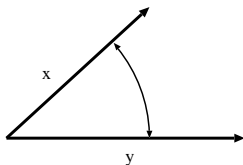
cosine similarity

- ▶ take the angle (cosine) of (x, y) of vectors
- ▶ normalized by the length of the vector \Rightarrow length is not considered

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

$x \cdot y = \sum_{k=1}^n x_k y_k$: product of vectors

$\|x\| = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{x \cdot x}$: length of the vector



example: cosine similarity

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

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

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

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

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

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

clustering

important technique for classifying data with complex relationship

compute the distance (or similarity) of variables to make them into groups

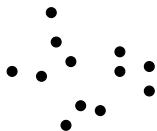
- ▶ to classify and understand data
- ▶ to summarize data

various applications

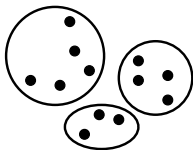
- ▶ business: grouping customers for marketing purposes
- ▶ meteorology: finding patterns in complex weather data
- ▶ biology: classifying genes and proteins
- ▶ medical science and pharmacy: complex relationship of symptoms and effects

clustering methods

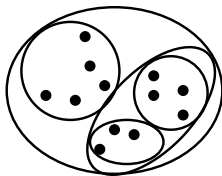
- ▶ partitional clustering
 - ▶ k-means method
- ▶ hierarchical clustering
 - ▶ MST method
 - ▶ DBSCAN method



original points



partitional clustering



hierarchical clustering

k-means method

- ▶ partitional clustering
- ▶ specify the number of cluster, k
- ▶ basic algorithm is simple
 - ▶ each cluster has centroid (usually mean)
 - ▶ assign each object to the closest cluster
 - ▶ repeat re-computation of centroids and cluster assignments
- ▶ limitations
 - ▶ need to specify the number of clusters, k , beforehand
 - ▶ sensitive to the selection of initial points
 - ▶ clusters are supposed to have similar sizes and densities, and a round shape
 - ▶ sensitive to outliers

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 centroid
- 4: recompute the centroid of each cluster
- 5: **until** the centroids don't change

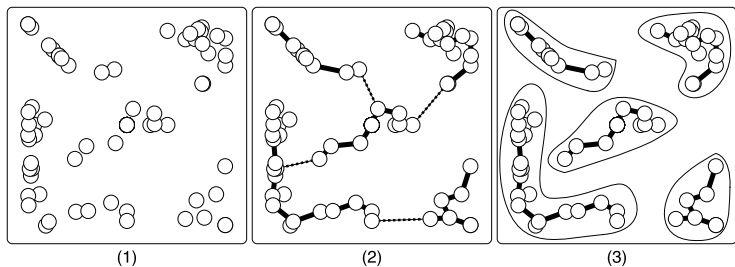
hierarchical clustering

- ▶ generate clusters using a tree structure
 - ▶ the cluster structure can be explained by the tree
- ▶ no need to specify the number of clusters beforehand
- ▶ 2 approaches
 - ▶ agglomerative: start with data points as individual clusters, and repeat merging the closest clusters
 - ▶ divisive: start with one all-inclusive cluster, and repeat splitting clusters

MST clustering

Minimum Spanning Tree clustering

- ▶ divisive hierarchical clustering
- ▶ start with an arbitrary point, and create MST
- ▶ repeat dividing clusters by removing the longest edge



DBSCAN

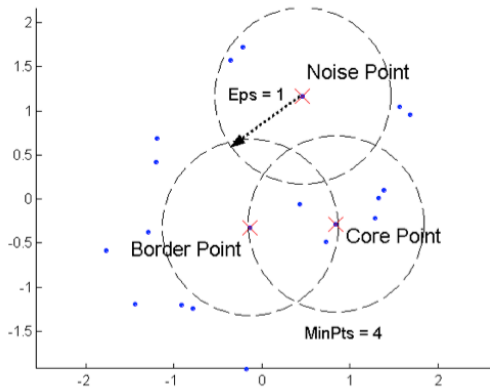
Density-Based Spatial Clustering

- ▶ density-based: combine data points within the specified distance
- ▶ can extract arbitrary (non-round) shapes of clusters
- ▶ robust against noise and outliers
- ▶ distance threshold Eps and point threshold $MinPts$
 - ▶ Core points: within the distance Eps , more than $MinPts$ neighbors exist
 - ▶ Border points: not Core, but have a core within the distance Eps
 - ▶ Noise points: have no core within the distance Eps
- ▶ limitations: clusters with different densities, or with large number of parameters

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

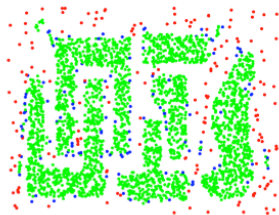


source: Tan, Steinbach, Kumer. Introduction to Data Mining

DBSCAN: example of Core, Border, and Noise Points



Original Points

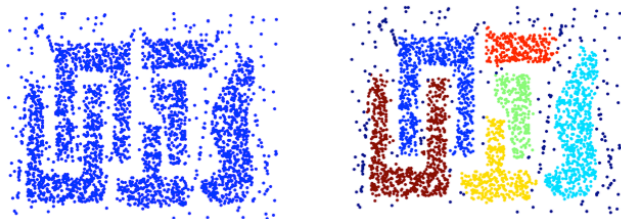


Point types: **core**, **border**
and **noise**

Eps = 10, MinPts = 4

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

DBSCAN: example clusters



Clusters

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

assignment 2: twitter data analysis

- ▶ purpose: processing realworld big data
- ▶ data sets:
 - ▶ twitter data for about 40M users by Kwak et al. in July 2009
 - ▶ <http://an.kaist.ac.kr/traces/WWW2010.html>
 - ▶ twitter_degrees.zip (164MB, 550MB uncompressed)
 - ▶ user_id, followings, followers
 - ▶ numeric2screen.zip (365MB, 756MB uncompressed)
 - ▶ user_id, screen_name
- ▶ items to submit
 1. CCDF plot of the distributions of twitter users' followings/followers
 - ▶ log-log plot, the number of followings/followers on X-axis
 2. list of the top 30 users by the number of followers
 - ▶ rank, user_id, screen_name, followings, followers
 3. optional
 - ▶ other analysis of your choice
 4. discussion
 - ▶ describe what you observe from the data
- ▶ submission: upload your report in the PDF format via SFC-SFS
- ▶ submission due: 2013-12-12 (Thu)

twitter data sets

twitter_degrees.zip (164MB, 550MB uncompressed)

```
# 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, 756MB uncompressed)

```
# id screenname
```

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

items to submit

CCDF plot

- ▶ log-log plot, the number of followings/followers on X-axis
- ▶ plot the 2 distributions in a single graph

list of the top 30 users by the number of followers

- ▶ rank, user_id, screen_name, followings, followers
- ▶ you need to sort and merge 2 files

#	rank	id	screenname	followings	followers
1		19058681	aplusk	183	2997469
2		15846407	TheEllenShow	26	2679639
3		16409683	britneyspears	406238	2674874
4		428333	cnbrk	18	2450749
5		19397785	0prah	15	1994926
6		783214	twitter	55	1959708
...					

sort command

sort command: sorts lines in a text file

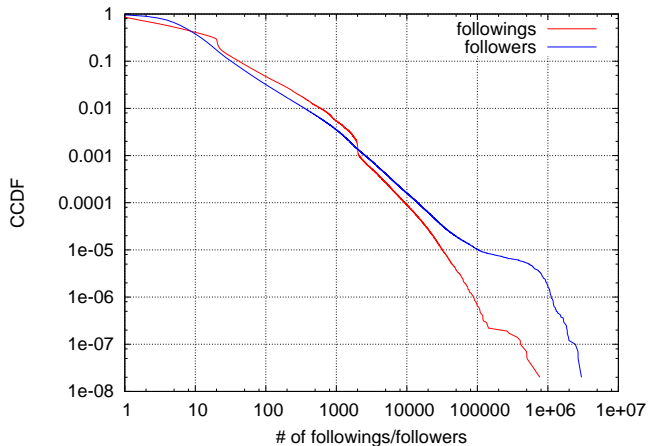
```
$ sort [options] [FILE ...]
```

- ▶ options (relevant to the assignment)
 - ▶ -n : compare according to string numerical value
 - ▶ -r : reverse the result of comparisons
 - ▶ -k POS1[,POS2] : start a key at POS1, end it at POS 2 (origin 1)
 - ▶ -t SEP : use SEP instead of non-blank as the field-separator
 - ▶ -m : merge already sorted files
 - ▶ -T DIR : use DIR for temporary files

example: sort “file” using the 3rd field as numeric value in the reverse order , use “/usr/tmp” for temporary files

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

assignment 2 answer: CCDF plot



list of the top 30 users by the number of followers

#	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
26		16827333	souljaboytellem	94	1241331
27		20	ev	912	1213787
28		972651	mashable	1934	1210996
29		26885308	ahsimpsonwentz	32	1200472
30		6273552	MCHammer	27413	1195089

previous exercise: SPAM filtering

- ▶ SPAM filtering using naive bayesian classifier
 - ▶ based on the code from “Programming Collective Intelligence” Chapter 6

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

naive bayesian classifier for the exercise

compute the probability of a document to be classified into a specific category by words appearing in the document

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

- ▶ $P(C)$: the probability of the category
- ▶ $\prod_{i=1}^n P(x_i|C)$: product of the conditional probability of each word in the category

select the category with the highest probability

- ▶ threshold : the probability of the best category should be *thresh* times higher than that of the second best category

SPAM classifier script

► training and classifier

```
# 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
```

script: 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

  ...
end
```

script: 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
```

script: NaiveBayes Class

```
# naive bayesian 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
  end
end
```

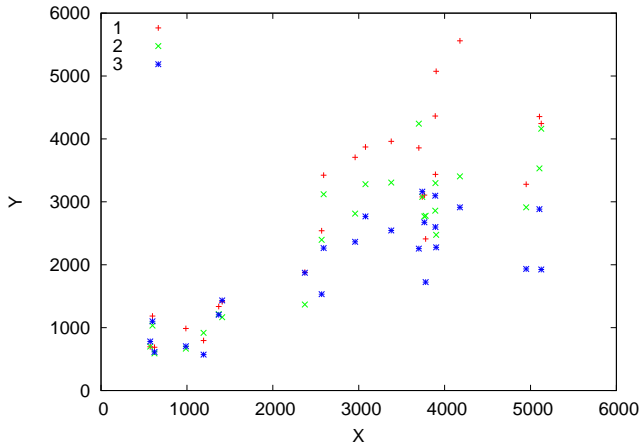
debug: dumping the feature probabilities

internal states after the training:

```
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
```

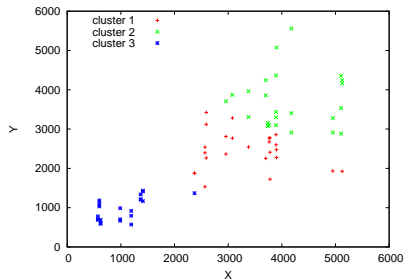
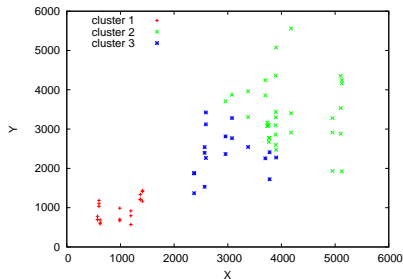
today's exercise: k-means clustering

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



k-means clustering results

- ▶ different results by different initial values



k-means code (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
```

k-means code (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
```

summary

Class 11 Data Mining

- ▶ Pattern extraction
- ▶ Classification
- ▶ Clustering
- ▶ exercise: clustering

next class

Class 12 Search and Ranking (12/25)

- ▶ Search systems
- ▶ PageRank
- ▶ exercise: PageRank algorithm