インターネット計測とデータ解析 第10回

長 健二朗

2012年6月15日

前回のおさらい

トポロジーとグラフ

- 経路制御
- グラフ理論
- 最短経路探索
- 演習: 最短経路探索

今日のテーマ

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

異常とは

- トラフィック異常
- 経路異常、到達性異常
- DNS 異常
- 不正侵入
- CPU 負荷異常

異常原因

- アクセス集中
- 攻撃: DoS、ウィルス/ワーム
- 障害: 機器故障、回線故障、事故、停電
- メンテナンス

異常検出

- サービスの機能低下や停止による損失の回避と低減
- 個別項目の監視: 閾値を越えるとアラート
  - パッシブ
  - アクティブ
- 異常パターン検出:
  - 既知の異常とパターンマッチング
  - IDS: Intrusion Detection System
  - 未知の異常は検出できない
  - パターンを常に更新する必要
- 統計的手法による異常検出
  - 平常時からのずれを検出
  - 一般に「平常」の学習が必要

異常への対応

- 異常を管理者に知らせる
  - 警報通知など
- 異常タイプの識別
  - 運用者が異常原因を把握するための情報提示
  - 特に統計的手法の場合、異常の原因が分かり難い
- 対応の自動化
  - フィルタリングルールの自動生成、サービス切替えなど

異常の具体例

- Flash Crowd
  - サービスへのアクセス集中 (ニュース、イベント、etc)
- DoS/DDoS
  - 特定のホストにトラフィックを集中する攻撃
  - ゾンビ PC が使われる
- scan
  - 多くの場合、脆弱性を持つホストを発見する目的
- worm/virus
  - SQL Slammer, Code Red など多数の事例
- 経路ハイジャック
  - 他人の経路を広告 (多くは設定ミス)

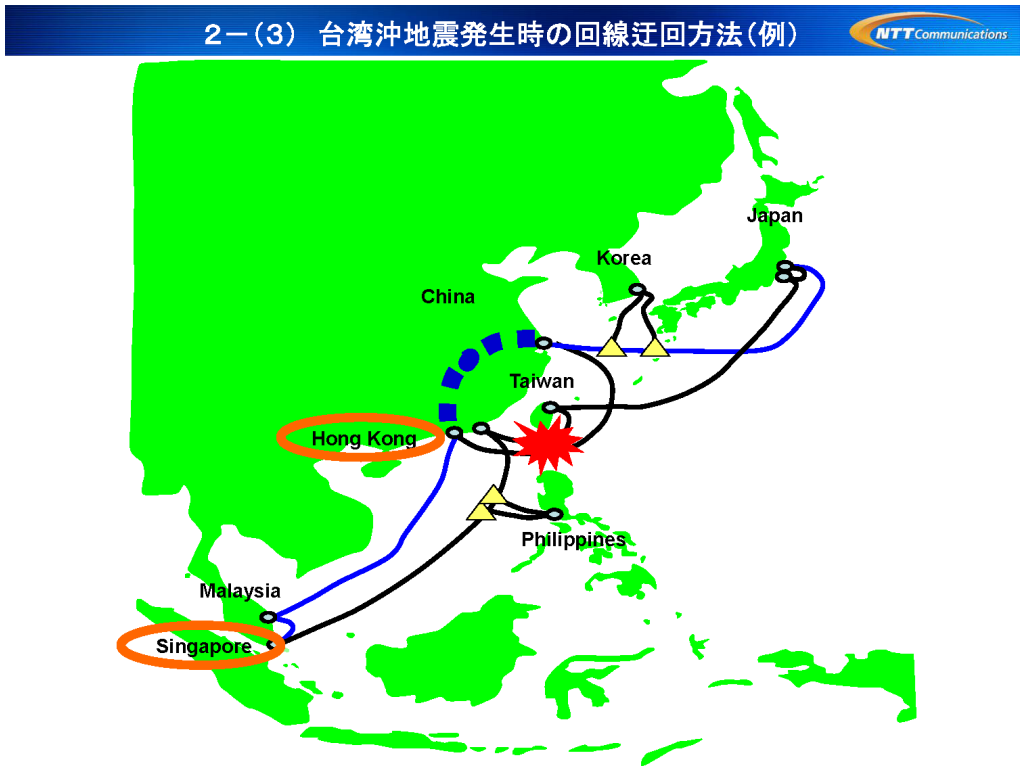
YouTube 接続のハイジャック

- 2008 年 2 月 24 日 世界中の YouTube への接続がパキスタンにリダイレクトされた事件
- 原因
  - パキスタン政府の要請で、Pakistan Telecom が国内から YouTube へ接続できないよう、BGP に YouTube の偽の経路を広告
  - 大手 ISP PCCW が、その経路を外部に伝搬
  - 結果、世界中の YouTube への接続が偽経路によってパキスタンにリダイレクトされた

参考資料:
`http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml`

台湾沖地震による通信障害の発生

- 2006 年 12 月 26 日台湾南西沖で M7.1 の地震発生
- 海底ケーブルが損傷、アジア向けの通信に障害が発生
- インドネシアでは一時国際向けの通信容量が 20%以下に
- 各 ISP は迂回経路でサービス復旧



出典: JANOG26 海底ケーブル、構築と運用の深イイ話

http://www.janog.gr.jp/meeting/janog26/doc/post-cable.pdf

ISP 間の接続遮断

- Tier1 ISP 同士が接続料金の負担をめぐって争いになった事例
- 2005 年 Level 3 が Cogent 側のトラフィック量が増加していると主張、無償のピアリングを解消し、有償による接続契約の変更を打診
- その他の事例
  - 2008 年 Cogent と Telia がピアリングを解消
  - 2008 年 Level 3 と Cogent がピアリングを解消
  - 2010 年 Level 3 と Comcast が対立し、交渉中

参考資料:
http://www.renesys.com/blog/2006/11/sprint-and-cogent-peer.shtml
http://wirelesswire.jp/Watching_World/201012011624.html

統計的手法による異常検出

- 時系列
- 相関
- 主成分分析
- クラスタリング
- エントロピー

機械学習

- 教師あり学習
  - 訓練データを用いて事前にトレーニングを行う
- 教師なし学習
  - 自動的に分類やパターン抽出を行うもの
  - トレーニングが不要
  - クラスター分析、主成分分析など

スパム判定

スパム: 迷惑メール
判定手法

- 送信者による判定
  - ホワイトリスト
  - ブラックリスト
  - グレーリスト
- コンテンツによる判定
  - ベイジアンフィルタ: スパム判定手法として広く普及
  - 迷惑メールの特徴を統計的な学習手法で分析し判定
  - 学習機能により精度が向上
  - メールからトークン (単語など) を抽出し、含まれるトークンからそのメールがスパムであるかどうか判定

条件付き確率

問題

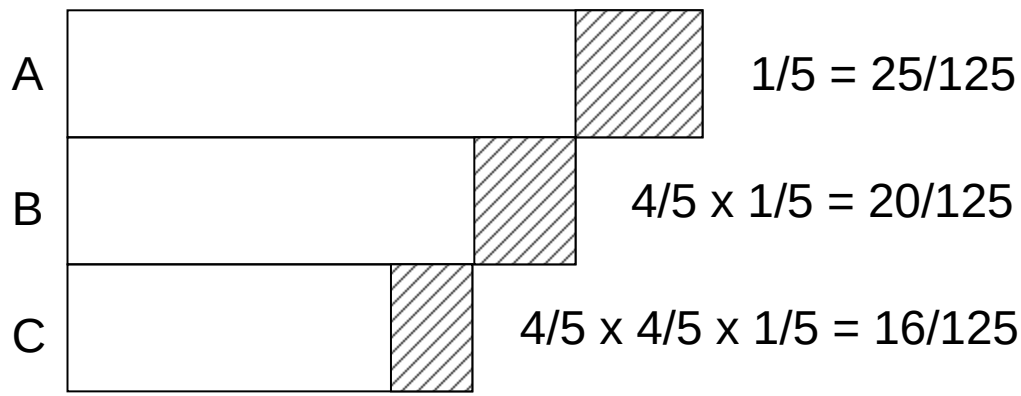
- 5 回に 1 回の割合で帽子を忘れるくせのある K 君が、正月に A、B、C 軒を順に年始回りをして家に帰ったとき、帽子を忘れてきたことに気がついた。2 軒目の家 B に忘れてきた確率を求めよ。 (昭和 51 年 早稲田大入試問題)

条件付き確率

問題

- 5 回に 1 回の割合で帽子を忘れるくせのある K 君が、正月に A、B,C 軒を順に年始回りをして家に帰ったとき、帽子を忘れてきたことに気がついた。2 軒目の家 B に忘れてきた確率を求めよ。 (昭和 51 年 早稲田大入試問題)

解



B で帽子を忘れた確率 / いずれかの場所で帽子を忘れた確率 $= 20/61$

ベイズ理論 (Bayes' theorem)

条件付き確率

- ある事象 A が起こるという条件の下で別の事象 B の起こる確率: $P(B|A)$
  - 全ての場合を事象 A として、そのうち B の起こる事象 $(A \cap B)$ を求める

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

ベイズの定理

- 上記の例とは逆に、A という原因で B が起こったときに、その原因が起こる確率を求める: $P(A|B)$
  - $P(A)$: 原因 A の存在確率 (事前確率)
  - $P(A|B)$: B が起こった場合の原因 A の確率 (事後確率)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)}$$

ベイズ理論の応用

観測結果から原因の確率を推測する: 多くの工学的応用

- 通信: ノイズの加わった受信信号から送信信号を求める
- 医学: 検査結果から実際に疾患を持つ可能性を求める
- スパム判定: 届いたメールの文面から迷惑メールであるか求める

病気検査の例

問題

- ある病気に掛かっている人口割合は 50/1000、この病気の検査は、この病気の患者の 90%が陽性が出るが、患者でない人も 10%は陽性反応がでる。
  あるひとがこの検査で陽性反応が出た場合、本当にこの病気にかかっている確率はいくらか？

病気検査の例

問題

- ある病気に掛かっている人口割合は 50/1000、この病気の検査は、この病気の患者の 90%は陽性が出るが、患者でない人も 10%は陽性反応がでる。
  あるひとがこの検査で陽性反応が出た場合、本当にこの病気にかかっている確率はいくらか？

解

病気にかかっている確率: $P(D) = 50/1000 = 0.05$
陽性反応が出る確率: $P(R) = P(D \cap R) + P(\bar{D} \cap R)$
陽性反応が出た場合、病気である事後確率

$$\begin{aligned} P(D|R) &= \frac{P(D \cap R)}{P(R)} \\ &= (0.05 \times 0.9)/(0.05 \times 0.9 + 0.95 \times 0.1) = 0.321 \end{aligned}$$

迷惑メール判定

- 迷惑メール (SPAM) とそうでないメール (HAM) を用意
- 迷惑メールに多く含まれる単語について
  - SPAM がこの単語を含む条件つき確率
  - HAM がこの単語を含む条件つき確率
- を計算しておき、この単語を含む未知のメールが SPAM である事後確率を求める

例: ある単語 A に関して、$P(A|S) = 0.3$, $P(A|H) = 0.01$, $\frac{P(H)}{P(S)} = 2$ の場合に $P(S|A)$ を求める

$$
\begin{aligned}
P(S|A) &= \frac{P(S)P(A|S)}{P(S)P(A|S) + P(H)P(A|H)} \\
&= \frac{P(A|S)}{P(A|S) + P(A|H)P(H)/P(S)} \\
&= \frac{0.3}{0.3 + 0.01 \times 2} = 0.94
\end{aligned}
$$

単純ベイズ分類器 (naive Bayesian classifier)

- 実際には、複数のトークンを利用
  - トークン同士の組合せを考慮すると膨大なデータが必要
- 単純ベイズ分類器: 各トークンが独立と仮定
  - 独立でない場合でも、実際には有効な場合が多い
  - 学習ステップ:
    - 判定済み学習サンプルから各トークンがスパムに含まれる確率を推定
  - 予測ステップ:
    - 判定が未知のメールに対し、含まれるトークンの推定スパム確率からメールがスパムである事後確率を計算、判定
- 学習ステップはトークン毎に独立計算なので簡単
- トークンスパム確率から結合スパム確率の算出にベイズの結合確率を使う

単純ベイズ分類器 (もう少し詳しく)

トークンを $x_1, x_2, \ldots, x_n$ とする。 これらが出現したとき SPAM である事後確率は

$$P(S|x_1, \ldots, x_n) = \frac{P(S)P(x_1, \ldots, x_n|S)}{P(x_1, \ldots, x_n)}$$

分子の部分は、これらのトークンが出現し、かつ SPAM である同時確率なので、以下のように書け、条件つき確率の定義を繰り返し適用すると

$$\begin{aligned} P(S, x_1, \ldots, x_n) &= P(S)P(x_1, \ldots, x_n|S) \\ &= P(S)P(x_1|S)P(x_2, \ldots, x_n|S, x_1) \\ &= P(S)P(x_1|S)P(x_2|S, x_1)P(x_3, \ldots, x_n|S, x_1, x_2) \end{aligned}$$

ここで、各トークンが条件付きで他のトークンと独立だと仮定すると

$$P(x_i|S, x_j) = P(x_i|S)$$

すると上記の同時確率は

$$P(S, x_1, \ldots, x_n) = P(S)P(x_1|S)P(x_2|S)\cdots P(x_n|S) = P(S)\prod_{i=1}^{n} P(x_i|S)$$

したがって、各トークンが独立だとの仮定の下で、SPAM である事後確率は

$$P(S|x_1, \ldots, x_n) = \frac{P(S)\prod_{i=1}^{n} P(x_i|S)}{P(S)\prod_{i=1}^{n} P(x_i|S) + P(H)\prod_{i=1}^{n} P(x_i|H)}$$

今回の演習: 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
```

今回の演習: 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

今回の演習: 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
```

今回の演習: 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

  ...
```

今回の演習: 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
```

今回の演習: 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
```

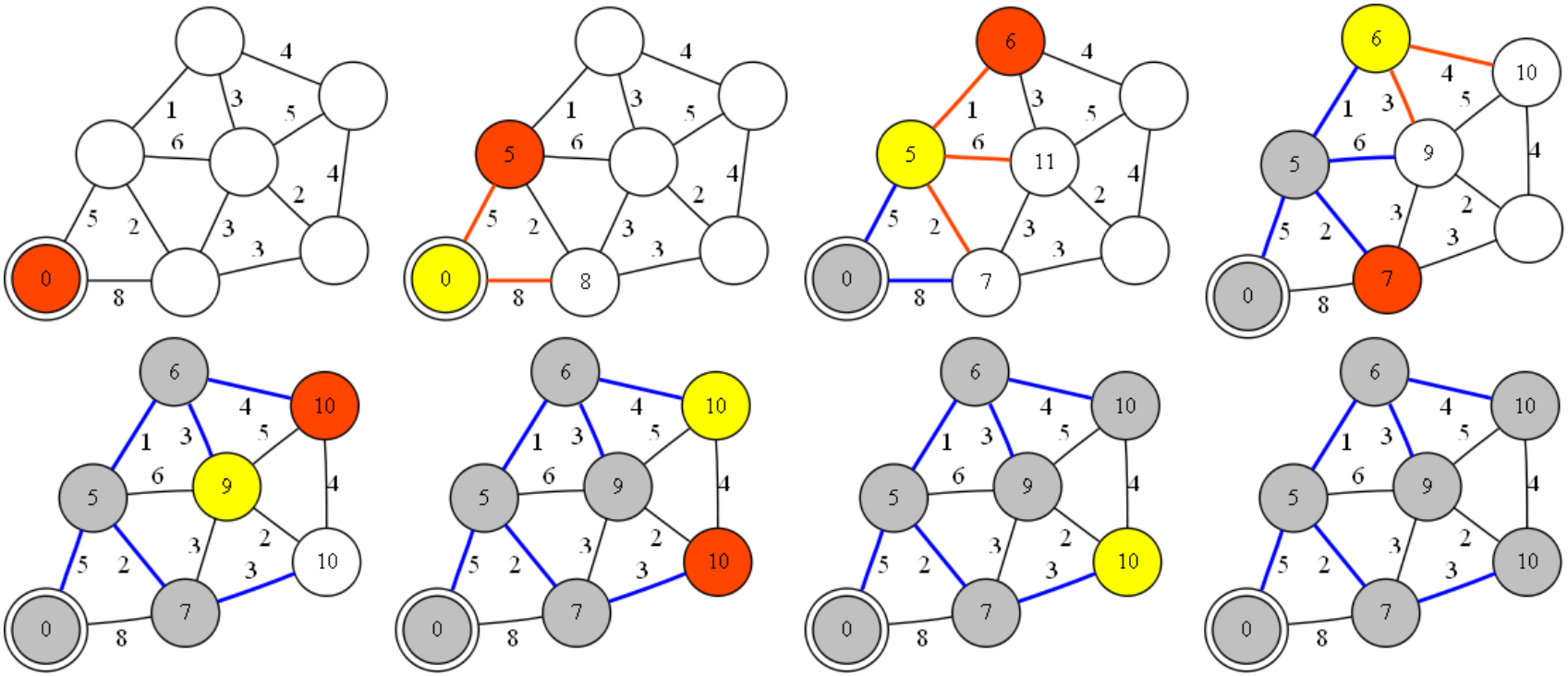
前回の演習: Dijkstra アルゴリズム

- トポロジファイルを読んで、最短経路木を計算する

```
% cat topology.txt
a - b 5
a - c 8
b - c 2
b - d 1
b - e 6
c - e 3
d - e 3
c - f 3
e - f 2
d - g 4
e - g 5
f - g 4
% ruby dijkstra.rb -s a topology.txt
a: (0) a
b: (5) a b
c: (7) a b c
d: (6) a b d
e: (9) a b d e
f: (10) a b c f
g: (10) a b d g
%
```

Dijkstra アルゴリズム

1．初期化：スタートノード値 = 0、他のノード値 = 未定義
2．ループ：
　　（1）未確定ノード中、最小値のノードを確定
　　（2）確定したノードの隣接ノードのコスト更新



dijkstra algorithm

sample code (1/4)

```
# dijkstra's algorithm based on the pseudo code in the wikipedia
# http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
#
require 'optparse'

source = nil # source of spanning-tree

OptionParser.new {|opt|
  opt.on('-s VAL') {|v| source = v}
  opt.parse!(ARGV)
}

INFINITY = 0x7fffffff # constant to represent a large number
```

sample code (2/4)

```
# read topology file and initialize nodes and edges
# each line of topology file should be "node1 (-|->) node2 weight_val"
nodes = Array.new # all nodes in graph
edges = Hash.new # all edges in graph
ARGF.each_line do |line|
  s, op, t, w = line.split
  next if line[0] == ?# || w == nil
  unless op == "-" || op == "->"
    raise ArgumentError, "edge_type should be either '-' or '->'"
  end
  weight = w.to_i
  nodes << s unless nodes.include?(s) # add s to nodes
  nodes << t unless nodes.include?(t) # add t to nodes
  # add this to edges
  if (edges.has_key?(s))
    edges[s][t] = weight
  else
    edges[s] = {t=>weight}
  end
  if (op == "-")  # if this edge is undirected, add the reverse directed edge
    if (edges.has_key?(t))
      edges[t][s] = weight
    else
      edges[t] = {s=>weight}
    end
  end
end
# sanity check
if source == nil
  raise ArgumentError, "specify source_node by '-s source'"
end
unless nodes.include?(source)
  raise ArgumentError, "source_node(#{source}) is not in the graph"
end
```

sample code (3/4)

```
# create and initialize 2 hashes: distance and previous
dist = Hash.new # distance for destination
prev = Hash.new # previous node in the best path
nodes.each do |i|
  dist[i] = INFINITY # Unknown distance function from source to v
  prev[i] = -1 # Previous node in best path from source
end

# run the dijkstra algorithm
dist[source] = 0 # Distance from source to source
while (nodes.length > 0)
  # u := vertex in Q with smallest dist[]
  u = nil
  nodes.each do |v|
    if (!u) || (dist[v] < dist[u])
      u = v
    end
  end
  if (dist[u] == INFINITY)
    break # all remaining vertices are inaccessible from source
  end
  nodes = nodes - [u] # remove u from Q
  # update dist[] of u's neighbors
  edges[u].keys.each do |v|
    alt = dist[u] + edges[u][v]
    if (alt < dist[v])
      dist[v] = alt
      prev[v]  = u
    end
  end
end
```

sample code (4/4)

```
# print the shortest-path spanning-tree
dist.sort.each do |v, d|
  print "#{v}: " # destination node
  if d != INFINITY
    print "(#{d}) " # distance
    # construct path from dest to source
    i = v
    path = "#{i}"
    while prev[i] != -1 do
      path.insert(0, "#{prev[i]} ") # prepend previous node
      i = prev[i]
    end
    puts "#{path}" # print path from source to dest
  else
    puts "unreachable"
  end
end
```

まとめ

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

次回予定

第 11 回 データマイニング (6/22)

- 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

- ゲストトーク