Übung zur Vorlesung

Maschinelles Lernen 1

Wintersemester 2008/2009

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und
theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: krm@cs.tu-berlin.de

Blatt 8

Abgabe bis Mittwoch, 10. Dezember 2008, 14:00 Uhr
Ausarbeitung im Sekretariat FR6052, oder bei Mikio Braun, FR6058 (notfalls unter der Tür durchschieben),
praktischen Teil unter https://ml01.service.tu-berlin.de/~mikio/pass.pl?conf=blatt8.conf.

Aufgaben (K-Means Clustering und Stabilität)

Auf diesem Aufgabenzettel wird behandelt, wie man Stabilitätsuntersuchungen verwenden kann, um die "richtige" Anzahl von Clustern zu schätzen. Die Anführungszeichen sollen andeuten, dass es selbst theoretisch unklar ist, was für einen gegebenen Datensatz die richtige Anzahl von Clustern ist. Nimmt man zum Beispiel an, dass die Daten von einer Mixturverteiltung von Gaußschen Verteilungen gezogen wird, so hängt es von dem Abstand der Zentren ab, ob es wirklich zwei Cluster sind, oder nur einer.

Für Methoden wie k-Means-Clustering, die eine zufällige Komponente besitzen, kann man untersuchen, für welches k die gefundenen Lösungen am stabilsten sind. Wenn man zwei unterschiedliche Clusterlösungen $y_1, \ldots, y_n$ und $y'_1, \ldots, y'_n$ für dieselben Punkte hat (mit Werten zwischen 1 und k), so kann man diese vergleichen, indem man misst, ob Punktpaare in beiden Lösungen in derselben Klasse liegen oder nicht. Die co-cluster-Distanz ist definiert durch

$$d(y, y') = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} 1_{\{y_i = y_j \wedge y'_i = y'_j\}}.$$

Wir nehmen jetzt die erwartete Distanz für k-Means-Clusteringlösungen mit jeweils zufälliger Initialisierung als Maß für die Güte einer Anzahl k von Clustern. Der Erwartungswert von $d(y, y')$ wird approximiert durch den Mittelwert über eine gewisse Anzahl von Einzelläufen.

Ohne Weiteres kann man die Distanzen für verschiedene k nicht vergleichen, weil für eine höhere Anzahl von Klassen die Wahrscheinlichkeit, dass Punkte in derselben Klasse liegen, kleiner wird. Daher normalisiert man die Distanzen, indem man durch den Erwartungswert des zufälligen Clusterers teilt, der jedem Punkt gleichverteilt einen Wert aus $\{1, \ldots, k\}$ zuweisst.

1. Das Programmskelett führt die obige Analyse auf dem Datensatz vom letzten Übungszettel durch und berechnet dann noch einmal eine Clusterlösung für die gefundene Anzahl von Clustern. Ergänze das Programmskelett wie folgt:

   (a) `random_clustering`: Generiere eine Clusterlösung, indem Du jedem Punkt gleichverteilt eine Index aus $\{1, \ldots, k\}$ zuweist. **(5 Punkte)**

   (b) `cocluster_distance`: Bestimmte die Co-Cluster-Distanz für zwei Vektoren aus Klassenindices. Vermeide hierfür wieder die Verwendung von for-Schleifen. Verwende statt dessen `pwdist` für die paarweisen Vergleiche und logische Operationen auf den Matrizen, um die gewünschte Funktion zu berechnen. **(10 Punkte)**

   (c) `normalize`: Teile in der Resultatsmatrix die Ergebnisse für jedes k durch den Mittelwert der Stabilitätswerte des zufälligen Clusterers. **(5 Punkte)**

2. Versuche Clusterzentren so zu finden, dass die Stabilitätsuntersuchung die richtige Anzahl von Clustern findet. Editiere hierzu die Definition von *centers* am Anfang des Programmskeletts. **(10 Punkte)**

```
function sheet08

% Generate data
centers = [0, 0; 7, 3; -2, 4; 0, 10; -5, -5];
% (2): You centers here
```

```
% centers = [...]
X = generate_data(centers, 50);

MAXK = 8; % maximal number of clusters
ITERS = 10; % how many restarts.

% compute (in-)stability for k-means and a random clustering
DKMEANS = evaluate_clusterer(@k_means_clustering, X, MAXK, ITERS);
DRAND = evaluate_clusterer(@random_clustering, X, MAXK, ITERS);

% plot the two stability values
figure(1)
errorbar(2:8, mean(DKMEANS), std(DKMEANS));
hold on
errorbar(2:8, mean(DRAND), std(DRAND), 'r');
hold off
legend('k-means stability', 'random clusterer')

% normalize the stability by the random clusterer
D = normalize(DKMEANS, DRAND);

figure(2)
errorbar(2:8, mean(D), std(D));
title('normalized k-means')

% show clustering for most stable number of clusters
[dummy, OPTK] = min(mean(D)); OPTK = OPTK + 1;

Y = k_means_clustering(OPTK, X);
figure(3);
plot_clustering(X, Y);


% generate data from some centers
function X = generate_data(C, N)
X = [];
for I = 1:size(C, 1)
  X = [ X ; randn(N, 2) + repmat(C(I, :), N, 1) ];
end


% A variant of the k-means clustering algorithm which just iterates 100
% times.
function Y = k_means_clustering(K, X)
[N, D] = size(X);

% randomly select K points as centers
P = randperm(N);
MEANS = X(P(1:K), :);

% set up variables
for I = 1:100
  % compute nearest neighbor assignments
  dist = pwdist(MEANS, X);
  [dummy, Y] = min(dist);

  % compute new means
  for J = 1:K
    NJ = sum(Y == J);
    if NJ > 0
      MEANS(J, :) = sum(X(find(Y == J), :))/NJ;
```

```
    end
  end
end


% Plot the clustering and the centers.
function plot_clustering(X, Y)
gscatter(X(:, 1), X(:, 2), Y);


% Compute all pairwise distances quickly.
function D = pwdist(X, Y)
D = size(X, 2);
N = size(X, 1);
M = size(Y, 1);

XX = sum(X.*X, 2);
YY = sum(Y.*Y, 2);
D = repmat(XX, 1, M) + repmat(YY', N, 1) - 2*X*Y';


% evaluate a clustering algorithm for clusters 2:MAXK
% for ITERS restarts.
function D = evaluate_clusterer(method, X, MAXK, ITERS)
D = zeros(ITERS, MAXK-1);
for K = 2:MAXK
  fprintf('K = %d\n', K);
  for I = 1:ITERS
    Y1 = feval(method, K, X);
    Y2 = feval(method, K, X);
    D(I, K-1) = cocluster_distance(Y1, Y2);
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fill in your solutions below
%

% 1 (a). The random clusterer. It should return random labels from 1..K
% for each data point in X (stored in the rows.
function Y = random_clustering(K, X)
% ...

% 1 (b). Compute the co-cluster distance for two labelings Y1 and Y2.
function D = cocluster_distance(Y1, Y2)
% ...

% 1 (c). Normalize DKMEANS by dividing the result for a fixed
% K by the mean value of the corrsponding values in DRAND.
function D = normalize(DKMEANS, DRAND)
% ...
```
