VisualRank using image similarity

Nikki Fitzherbert

Introduction

In 2008, two Google scientists presented a paper on a new algorithm called *VisualRank*. Up until that point, image searches by many commercial search engines had been conducted using textual cues from image filenames of captions associated with each image rather than visual cues from the images themselves (Cao et al., 2010).

At its core, the VisualRank algorithm leverages off the well-known PageRank algorithm by computing pairwise visual similarity scores between images from which a global ordering can be derived. This approach provides a robust and efficient computation of image similarities applicable to a large number of web queries and images (Jing & Baluja, 2008).

The intent of this analysis was two-fold. The first part developed a simple version of the VisualRank algorithm and used it to investigate the relative rankings when applied to a database of shape images. The second part explored the result of using VisualRank in conjunction with an image search query from an internet search engine.

Data

The image set used for this analysis was the MPEG-7 testing dataset, which contained 1,400 shape images in 'png' format. In addition, the image set was accompanied by the corresponding similarity matrix S and a cell array of corresponding $filenames$.

There were 70 broad groups of shape images in the MPEG-7 testing dataset, each containing 20 images.

Methodology

VisualRank uses 'visual hyperlinks' between images to determine the ranking of images in a given group. The process of developing this algorithm required five main steps. The first step was to form the adjacency matrix A for the graph that connected the images using the visual hyperlinks. Matrix A was a modified similarity matrix S, with all non-positive values and all values on the main diagonal set to zero. This meant that only images positively correlated with each other and/or pairs of different images were compared to each other.

The second step required computing the hyperlink matrix H, which was achieved by normalising matrix A. This essentially meant standardising the non-zero elements such that each row summed to one. The hyperlink matrix therefore described not only where the links between images were, but how likely the user was to click on them.

The next step was to form the random jump matrix J, which corresponded to a jump to a random image N: $J = \begin{bmatrix} {}^{1}/_{N} & \cdots & {}^{1}/_{N} \\ \vdots & \ddots & \vdots \\ {}^{1}/_{N} & \cdots & {}^{1}/_{N} \end{bmatrix}$.

The penultimate step was to use a 0.85 dampening factor[1] in forming the modified visual hyperlink matrix $Htilde$. This matrix stops the user and/or algorithm getting stuck when ending up on an image with no link to any other image in the database. It is related to the hyperlink matrix H and the random jump matrix J in the following manner: $\widetilde{H} = dH + (1 - d) \times J$.

Finally, the VisualRank algorithm was implemented by iterating: $v_{n+1} = v_n \widetilde{H}$. The VisualRank vector r is the vector of probabilities after an infinite number of clicks, and the final result is the same regardless of the starting vector v_0: $r = \lim_{n \to \infty} v_n = \lim_{n \to \infty} v_0 \widetilde{H}^n$.

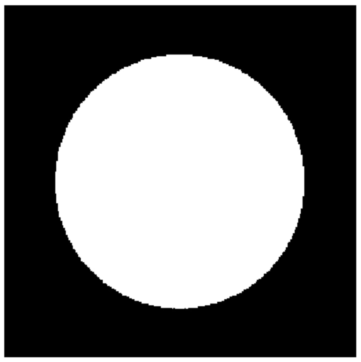
The same process was applied to a subset of 20 heart-shaped images (indices 81 through 100) with one exception: The adjacency matrix was formed by indexing the necessary rows and columns of adjacency matrix formed for the full dataset.

Part two of the analysis involved pretending that a search engine has been queried for an image search of the full database for the pentagon-looking shape with the filename 'device6-18.png'. The first step was to take the reciprocal of the original adjacency matrix, as this meant that similar images had smaller edge weights and therefore were closer to each other in a directed graph G. Taking the reciprocal of the adjacency matrix meant that Matlab's $nearest$ function could be used to determine which of the 1,399 other shape images were most-similar to 'device6-18.png'. Finally, the same VisualRank algorithm developed in the first part of the analysis was used to rank the ten most-similar shape images.

A copy of the Matlab code used to complete the analysis can be found at Appendix A.

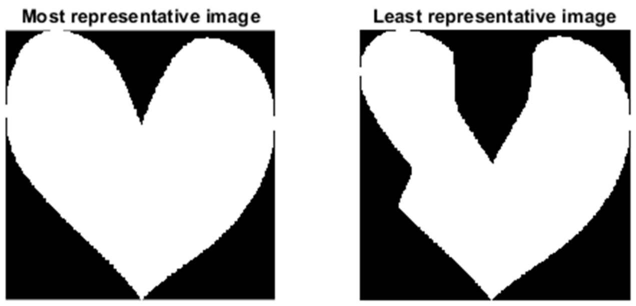
Results and Discussion

The VisualRank algorithm ranks images by how representative they are of the group as a whole. One of the algorithm's weaknesses is trying to rank images when there is a large number of them and they are very different(Cao et al., 2010). As a result, when applied to the 1,400 shape images in the MPEG-7 database, the image deemed most representative (or highest ranked) was the white circle 'device9-20.png' shown below.

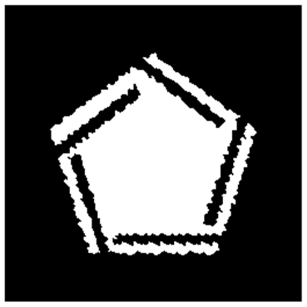


[1] A dampening factor of 0.85 is standard. It implies that there is an 85 per cent chance that the user will follow random links between images and a 15 per cent chance the user will jump to a random image.

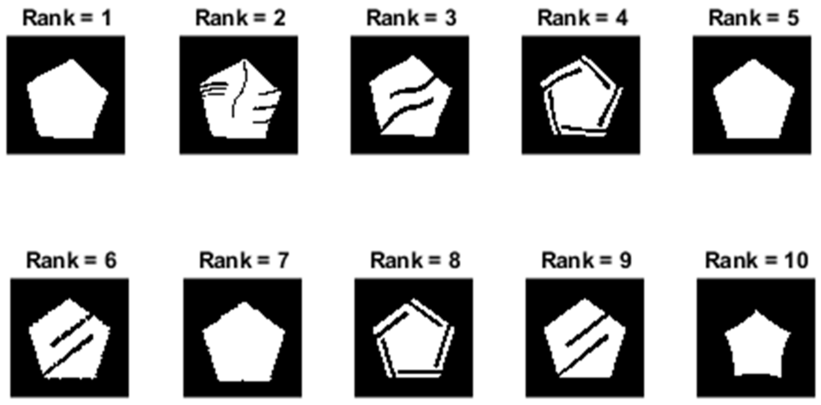
The algorithm provided more meaningful results when restricted to a smaller subset of images such as the 20 heart-shaped images in the database. In this subset, the image deemed most representative (or most 'heart-shaped') was 'Heart-1.png' and the image deemed least representative (least 'heart-shaped') was 'Heart-20.png' as shown over the page.



In the final part of the analysis, the shape image being used for the search engine query was:



The VisualRank algorithm using the reciprocal of the full adjacency matrix concluded that the following ten images[2] were the most similar to that image out of all 1,399 other shape images in the dataset (displayed in order from most to least similar).



[2] Their corresponding filenames are: 'device6-6.png', 'device6-5.png', 'device6-4.png', 'device6-3.png', 'device6-20.png', 'device6-17.png', 'device6-15.png', 'device6-13.png', 'device6-12.png', 'device6-11.png'.

References

Cao, L., Pozo, A., Jin, X., Luo, J., Han, J., & Huang, T. (2010). *RankCompete: simultaneous ranking and clustering of web photos.* Paper presented at the International World Wide Web Conference, Raleigh, North Carolina. doi:10.1145/1772690.1772809

Jing, Y., & Baluja, S. (2008). Visualrank: Applying pagerank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 30*(11), 1870-1890. doi:10.1109/TPAMI.2008.121

Appendix A – Matlab code

```
%% Assessment 5 - VisualRank using image similarity
% 13848336 Nikki Fitzherbert
%
%% Introduction
% The following code demonstrates how a version of Google's PageRank
% algorithm can be applied to images (the 'VisualRank' algorithm). The
% algorithm makes uses of 'visual hyperlinks' between images to determine
% which image is most representative of the set provided as input.
%
% The code is divided up into three parts:
%  Part 1 ranks all 1400 images
%  Part 2 ranks only the heart-shaped images
%  Part 3 searches for images similar to a given image before refining the
%    results using the algorithm

%% Part 1: Ranking all 1400 shape images using VisualRank
% ensuring the workspace is clear before starting
clear all
clc

% loading the data file into matlab. This file contains a similarity
% matrix 'S' and a cell array of corresponding filenames
load('sim.mat');

% using 'S' to form the adjacency matrix 'A'
A = S;
% retaining only positively correlated visual hyperlinks between the images
A(A <= 0) = 0;
% removing elements corresponding to loop edges
A = A - diag(diag(A));

% using 'A' to create the hyperlink matrix 'H'
H = zeros(1400,1400);
[row, col] = size(A);
for i = 1:col
   H(i,:) = A(i,:) / sum(A(i,:));
end

% forming the random jump matrix 'J' where 'N' is the total number of
% images
N = row;
J = ones(row,col) / N;
```

```
% creating the modified visual hyperlink matrix with a dampening factor of
% d = 0.85
d = 0.85;
Htilde = d * H + (1 - d) * J;

% determining the VisualRank vector 'r' using an initial vector 'v0'
% setting the parameters required for the iteration including
% a break point
v0 = [0,1,zeros(1,1398)];
% v0 = ones(1,col) / N;
r(1,:) = v0 * Htilde;

for n = 2:50
    r(n,:) = v0 * (Htilde^n);
    % setting a break so the loop stops when the difference between two
    % successive iterations is marginal
    if(all(abs(r(n,:) - r(n-1,:)) <= 0.0001))
        break;
    end
end

rvec = r(end,:)*100;

% determining which image is ranked highest and displaying this image
[~,I] = max(rvec);
imshow(filenames(Cao et al.; Jing & Baluja));


%% Part 2: Ranking the 20 heart-shaped images using VisualRank
% clearing the workspace
clear all
clc

% the next four lines to create the full adjacency matrix are exactly the
% same as in the previous part (lines 23 to 30) so the detailed commentary
% has not been repeated
load('sim.mat');
A = S;
A(A <= 0) = 0;
A = A - diag(diag(A));

% making a smaller adjacency matrix using just the rows and columns
% corresponding to the heart-shaped images
A = A(81:100, 81:100);
```

```
% forming the corresponding visual hyperlink matrix and finding the
% VisualRank vector
% this code is (almost) exactly the same as in the previous part
% (lines 33 to 47) so the detailed commentary has not been repeated
[row, col] = size(A);
H = zeros(20,20);
for i = 1:col
   H(i,:) = A(i,:) / sum(A(i,:));
end

N = row;
J = ones(row,col) / N;

d = 0.85;
Htilde = d * H + (1 - d) * J;

% adjusting the starting vector for the smaller 20x20 matrix
v0 = [0,1,zeros(1,18)];
r(1,:) = v0 * Htilde;

for n = 2:50
    r(n,:) = v0 * (Htilde^n);
    % setting a break so the loop stops when the difference between two
    % successive iterations is marginal i.e. a steady-state is reached
    if(all(abs(r(n,:) - r(n-1,:)) <= 0.0001))
        break;
    end
end

rvec = r(end,:)*100;

% determining which heart-shaped image is most representative of the 20
% given heart-shaped images and displaying this image
[~,Imax] = max(rvec);
imshow(filenames{Imax + 80})
title('Most representative image')

% determining which heart-shaped image is the least representative (least
% 'heart-shaped') and displaying this image
[~,Imin] = min(rvec);
figure, imshow(filenames{Imin + 80});
title('Least representative image')

% ... or diplaying the two images in the same figure/array
figure

subplot(1,2,1)
imshow(filenames{Imax + 80})
title('Most representative image')

subplot(1,2,2)
imshow(filenames{Imin + 80})
title('Least representative image')
```

```
%% Part 3: Searching for similar images and refining the results using
Visual Rank
% clearing the workspace
clear all
clc

% the next four lines to create the full adjacency matrix are exactly the
% same as in the previous part (lines 23 to 30) so the detailed commentary
% has not been repeated
load('sim.mat');
A = S;
A(A <= 0) = 0;
A = A - diag(diag(A));

% forming a new adjacency matrix for which the elements are the reciprocal
% of those in the original adjacency matrix 'A', and forming the graph 'G'
% corresponding to this new adjacency matrix
Arec = 1 ./ A;

G = digraph(Arec);
GG = digraph(Arec, 'omitselfloops');

% finding the 10 images nearest to 'device6-18.png'
position = find(strcmp(filenames, 'device6-18.png'));

[nodeIDs, dist] = nearest(G, position, 1.5);
top10 = nodeIDs(1:10);

% ranking the 10 nearest images using VisualRank and displaying them on a
% figure in their ranked order

% making a smaller adjacency matrix using just the rows and columns
% corresponding to the images identified by line 176 above
A = A(top10, top10);

% forming the corresponding visual hyperlink matrix and finding the
% VisualRank vector
% this code is (almost) exactly the same as in part A
% (lines 33 to 47) so the detailed commentary has not been repeated
[row, col] = size(A);
H = zeros(10,10);
for i = 1:col
   H(i,:) = A(i,:) / sum(A(i,:));
end

N = row;
J = ones(row,col) / N;

d = 0.85;
Htilde = d * H + (1 - d) * J;
```

```
% adjusting the starting vector for the smaller 10x10 matrix
v0 = [0,1,zeros(1,8)];
r(1,:) = v0 * Htilde;

for n = 2:50
    r(n,:) = v0 * (Htilde^n);
    % setting a break so the loop stops when the difference between two
    % successive iterations is marginal
    if(all(abs(r(n,:) - r(n-1,:)) <= 0.0001))
        break;
    end
end

rvec = r(end,:)*100;

% attaching the filename indexes onto 'r' and sorting into descending order
% (so highest ranked image is first etc)
r_ind = cat(1, top10', rvec);
r_ind = sort(r_ind, 2, 'descend');
r_ind_pos = r_ind(1,:)';

figure

%subplot(2,7,[1,9]);
%imshow(filenames{position})
%title('queried image')

subplot_i = 1;
for j = 1:5
    subplot(2,5,subplot_i);
    imshow(filenames{r_ind_pos(j)})
    title(['VisualRank = ', num2str(j)])

    subplot_i = subplot_i + 1;
end

subplot_ii = 6;
for jj = 6:10
    subplot(2,5,subplot_ii);
    imshow(filenames{r_ind_pos(jj)})
    title(['VisualRank = ', num2str(jj)])

    subplot_ii = subplot_ii + 1;
end
```