Tieteellinen laskenta harjoitustyöraportti

Itsenäisten komponenttien analyysi

Harjoitustyö toteutettu Matlab veriolla R2017b (9.3.0.713579)

# Harjoitustyön aihe

Harjoitustyön aiheena oli itsenäisten komponenttien analyysifunktioiden tutkiminen ja testaus.

Harjoitustyössä käytössä on 4 signaalia, joita merkitään . Pidetään signaaleja matriisissa S, jossa rivit ovat eri signaaleja. Signaalit ja ovat informaatiota, joista ollaan kiinnostuneita ja ja ovat kohinaa. Näistä on muodostettu 4 uutta signaalia, joihin on summattu signaalit S eri painoarvoilla kerrottuna. Merkitään näitä signaaleja , ja pidetään niitä matriisissa X. Siis

Signaalit matriisissa X sisältävät siis signaaleja ja , mutta myös kohinasignaaleja ja .

Tarkoitus on etsiä alkuperäiset signaalit ja signaalijoukosta X. Tähän käytetään Matlabiin saatavilla olevaa pakettia [1]. Paketissa ovat funktiot fastICA() ja kICA(), jotka molemmat suorittavat itsenäisten komponenttien analyysin annetulle matriisille. Paketissa on myös funktio PCA(), joka suorittaa pääkomponenttianalyysin matriisille. Paketista löytyy myös apufunktioita, kuten normalizeAudio(), joka skaalaa matriisien arvot välille [-1, 1].

Funktiot palauttavat joukon signaaleja, jotka tallennetaan matriiseihin Y1, Y2 ja Y3. Saatuja signaaleja verrataan alkuperäisiin signaaleihin ja . Vertailussa käytetään euklidista etäisyyttä, joka lasketaan kaavalla

missä on alkuperäisen signaalin :s näyte ja on analyysistä saadun signaalin :s näyte. Mitä pienempi signaalien välinen erotus on, sitä tehokkaammin analyysi toimii ja alkuperäiset funktiot on eroteltu signaalijoukosta X.

# Koodin toiminta

Koodi testaa yhdellä ajokerralla signaalien erottelun jokaisen funktion (fastICA(), kICA() ja PCA()) avulla. Lähdekoodin alussa voidaan määrittää, kuinka monta signaalia analysoidaan ja kuinka monta signaalia pyritään etsimään. Voidaan määrittää myös analysoitavien näytteiden määrä, sekä ICA erottelun tyyppi.

Muuttujat koodin alussa:

L = 50000; % samples per signal

RowsToAnalyze = 4; % rows of the matricies to input to the analysing functions

RowsToFind = 2; % rows to output from the analysing functions (number of signals we are looking for)

ICAType = 'negentropy'; % Type of ica for function fastICA. 'kurtosis' or 'negentropy'

ICAType määrittelee minkälaista riippumattomuutta (nongaussianity) algoritmi pyrkii lisäämään. Se voi olla joko ’kurtosis’ eli huipukkuus, jolloin algoritmi pyrkii lisäämään käyrien terävyyttä, tai ’negentropy’, jolloin algoritmi minimoi käyrien entropiaa.

Koodissa erilaisia tehtäviä on jaettu omiin funktioihin. Tehtävät on lyhyesti esitelty alla.

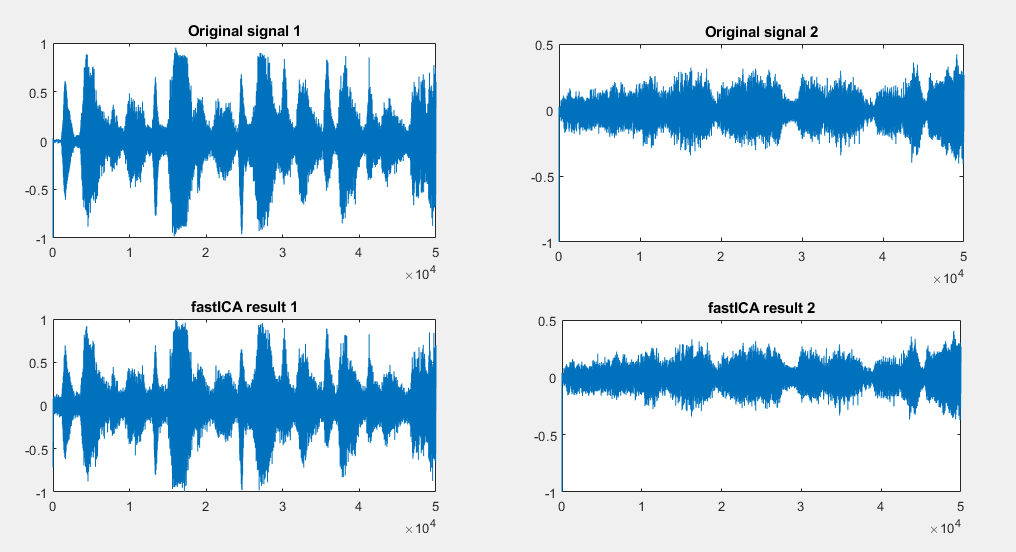
|  |  |
| --- | --- |
| plotMatrix | Helpottaa signaalijoukon piirtämistä piirtämällä signaaleja sisältävän matriisin tietylle riville. |
| matchMatrices | Lajittelee matriisin toisen matriisin signaalien mukaan niin, että samalla rivillä olevat signaalit ovat mahdollisimman lähellä toisiaan. |
| findClosest | Etsii signaaleja sisältävästä matriisista signaalin, joka on lähinnä annettua signaalia. |
| calculateDifference | Laskee kahden signaalin välisen etäisyyden. |

Tarkemmat rajapinnat ja määrittelyt löytyvät lähdekoodista jokaisen funktion yläpuolelta.

Analyysifunktiot eivät ota kantaa signaalien järjestykseen, eikä niiden skaalaukseen. Tämän vuoksi tulossignaaleille täytyy etsiä oikeat alkuperäiset signaalit. matchMatrices on juuri tätä varten. Kaikki signaalit myös skaalataan välille [-1, 1], jotta niitä voidaan verrata. Analyysissä kuitenkin signaalit on saatettu skaalata negatiivisella luvulla, jolloin tulossignaalit ovat käänteisiä alkuperäisiin signaalehin nähden. Tämän vuoksi oikeaa signaalia etsittäessä huomioidaan myös käänteiset signaalit.

# Tulokset

Suorittamalla koodia huomataan selvästi, että funktioista fastICA() antaa parhaimmat tulokset. kICA() ja PCA() antavat huomattavasti enemmän eroavat tulossignaalit. Tämä on ymmärrettävää ainakin PCA():n kohdalla, sillä pääkomponenttianalyysi pyrkii säilyttämään mahdollisimman paljon informaatiota, eikä niinkään välitä komponenttien itsenäisyydestä. kICA() käyttää eri algoritmia kuin fastICA() ja antaa selvästi huonompia tuloksia.



fastICA funktion tulokset näyttävät olevan kohtalaisen lähellä alkuperäisiä signaaleja. Tarkasti katsoen niissä esiintyy kuitenkin jonkin verran kohinaa.

Tulossignaaleja voidaan kuunnella Matlabin komennolla sound(). Kun tuloksia kuunnellaan, huomataan että signaaleissa esiintyy vieläkin kohinaa, mutta selvästi vähemmän kuin syötteinä käytettyjä signaaleja. Tuloksissa ei myös kuulu toista informaatiota sisältänyttä signaalia, joten signaalit on saatu eriteltyä hyvin.

Jos analysoitavien signaalien määrää vähennetään neljästä kolmeen, huomataan että ICA funktiot antavat huonompia tuloksia, mutta PCA funktion tulokset eivät juuri muutu. Tämä tulos vastaa odotuksia, sillä ICA:n saama informaatio vähenee, joten signaalien erottelu vaikeutuu, kun taas PCA ei alun perinkään pyri erottelemaan signaaleja.

Jos ICA:n tyyppien vaikutusta kokeillaan, huomataan että arvolla ’negentropy’ saadaan jonkin verran parempia tuloksia kuin arvolla ’kurtosis’.

# Lähdekoodi

% setup variables

L = 50000; % samples per vector

RowsToAnalyze = 4; % rows of the matricies to input to the analysing functions

RowsToFind = 2; % rows to output from the analysing functions (number of signals we are looking for)

ICAType = 'kurtosis'; % Type of ica for function fastICA. 'kurtosis' or 'negentropy'

% constants

f = 44100; % sampling frequency

Ts = 1/f; % sample time

fprintf('Searching for %i signals from the group of %i signals\n', RowsToFind, RowsToAnalyze);

% Load audio files into matrices

s1 = audioread('samples/wave\_1.wav')';

s2 = audioread('samples/wave\_2.wav')';

s3 = audioread('samples/wave\_3.wav')';

s4 = audioread('samples/wave\_4.wav')';

S = [s1(1:L);s2(1:L);s3(1:L);s4(1:L)];

S = normalizeAudio(S);

x1 = audioread('samples/mixed\_1.wav')';

x2 = audioread('samples/mixed\_2.wav')';

x3 = audioread('samples/mixed\_3.wav')';

x4 = audioread('samples/mixed\_4.wav')';

X = [x1(1:L);x2(1:L);x3(1:L);x4(1:L)];

X = normalizeAudio(X);

% Plot the original signals

plotMatrix(S, 4, RowsToAnalyze, 1, 'Original signal');

% Plot the mixed signals

plotMatrix(X, 4, RowsToAnalyze, 2, 'Mixed signal');

% Do the different types of analysis

Y1 = fastICA(X(1:RowsToAnalyze, :), RowsToFind, ICAType, 0);

Y2 = kICA(X(1:RowsToAnalyze, :), RowsToFind);

Y3 = PCA(X(1:RowsToAnalyze, :), RowsToFind);

% Normalize results to range 0-1

Y1 = normalizeAudio(Y1);

Y2 = normalizeAudio(Y2);

Y3 = normalizeAudio(Y3);

% The analysis mixes up the order of the signals so we need to match them

% ourselves.

% The matrices outputted from the analysis functions are matched by finding

% the original signal in S that is closest to each of the outputted

% signals.

Y1 = matchMatrices(S, Y1, RowsToFind);

Y2 = matchMatrices(S, Y2, RowsToFind);

Y3 = matchMatrices(S, Y3, RowsToFind);

plotMatrix(Y1, 4, RowsToAnalyze, 3, "fastICA result");

plotMatrix(Y3, 4, RowsToAnalyze, 4, "PCA result");

% Print out the results

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y1(i,:));

fprintf('The difference in signal #%i from fastICA: %f\n', i, d);

end

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y2(i,:));

fprintf('The difference in signal #%i from kICA: %f\n', i, d);

end

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y3(i,:));

fprintf('The difference in signal #%i from PCA: %f\n', i, d);

end

%==== functions to help with plotting, calculating the difference between

%vectors and matching the matrices =====

% Plots the individual rows of the given matrix using subplot()

%

% Parameters:

% mat - the matrix

% rowCount - the amount of rows in the subplot

% colCount - the amount of rows to draw from the matrix, each to

% different column of the subplot

% row - the row of subplot to draw the rows of the matrix

%

function [] = plotMatrix(mat, rowCount, colCount, row, titl)

[r, c] = size(mat);

e = min([colCount, r]);

for i = 1:e

subplot(rowCount,colCount,(row-1) \* colCount + i);

plot(mat(i,:));

title(strcat(titl, {' '}, num2str(i)));

end

end

% Matches the rows in the second matrix with the rows of the first one

% by finding the ones that are closest to each other in terms of euclidean

% distance.

% If matrices row counts dont match, add all zero rows to mat2

%

% Parameters:

% mat1 - first matrix, the one that will be sorted

% mat2 - second matrix

% rows - the amount of rows to sort, starting with 1

%

% Returns:

% mat - sorted version of mat2

%

function [mat] = matchMatrices(mat1, mat2, rows)

mat = mat2;

[r, c] = size(mat);

[r2, c2] = size(mat1);

while r2 > r

r = r + 1;

mat(r, :) = zeros(1, c);

end

for i = rows:-1:1 % start from the end row => priorisize the first rows

[index, inverse, row] = findClosest(mat1, mat(i, :)); % find the index of the closest row

temp = mat(i, :); % swap the rows

mat(i, :) = mat(index, :);

mat(index, :) = temp;

if inverse == 1 % if inverse then inverse the row

mat(i, :) = mat(i, :) \* -1;

end

end

end

% Finds the row of the given matrix that is closest to the given vector

% Also checks inversed versions of each rows (each sample \*= -1)

%

% Parameters:

% mat - The matrix

% vec - The vector

%

% Returns:

% index - The index of the row that is closest to the given vector

% inverse - True if the row is inversed, False if not

% row - the closest row in mat, inversed if closest that way

%

function [index, inverse, row] = findClosest(mat, vec)

[r, c] = size(mat);

if length(vec) ~= c

error("Vector length and matrix column count do not match.");

else

min = calculateDifference(mat(1,:), vec);

inverse = 0;

index = 1;

row = mat(1,:);

for i = 2:r

dif = calculateDifference(mat(i,:), vec);

if(dif < min)

min = dif;

index = i;

row = mat(i, :);

end

end

for i = 1:r

dif = calculateDifference(mat(i,:), vec \* -1);

if(dif < min)

min = dif;

index = i;

inverse = 1;

row = mat(i, :) \* -1;

end

end

end

end

% Calculates the difference between two vectors.

% The diffenrece is the euclidean distance between the vectors.

% It is calculated with the formula Sqrt((a1 - b1)^2 + (a2 - b2)^2 + .... + (an - bn)^2)

%

% Parameters:

% vec1 - first vector

% vec2 - second vector

%

% Returns:

% diff - The difference between the vectors

%

function [diff] = calculateDifference(vec1, vec2)

if length(vec1) ~= length(vec2)

error("Vectors must have the same length.");

else

diff = 0;

for i = 1:length(vec1)

diff = diff + (vec1(i) - vec2(i))^2;

end

diff = sqrt(diff);

end

end

# Paketit

[1] https://se.mathworks.com/matlabcentral/fileexchane/38300-pca-and-ica-package