Supplementary Information for Prognostic Molecular Classification of Breast Cancer Based on Features Extracted from a Scale-Space

Yingchao Wu, Jeroen de Ridder and Marcel J.T. Reinders
Bioinformatics, Delft University of Technology, The Netherlands
Email: Y.Wu-4@student.tudelft.nl

1 Additional methods experiments and figures

In this section, the methods, which are experimented during our study but did not work out, are described. Moreover, figures, which are obtained throughout the study, but not included in the main thesis, are shown.

1.1 Scale space tree

Besides the Gaussian smoothing of one-dimensional (1D) signal with three pulses (values of them are 3, 3 and 6 respectively, and two pulses with same value are closer to each other in x-axis.) mentioned in thesis, we also try smoothing a signal including two pulses by Gaussian function. The values of these pulses are 3 and 6, respectively. Then the signal convolutes with the Gaussian function with $\sigma = 2$ to 6 of step-size 0.2. The convolution results are given in **Figure 1**. Finally, local extrema of every convolution result are extracted and combined with corresponding scale level to produce the scale space tree, as shown in **Figure 2**. Comparing to the scale space tree of signal with three pulses, positions in x-axis of the merged dot in high scale level in **Figure 2** are closer to the position in x-axis of pulse with large value. It means that two closer pulses strengthen each other during smoothing.

1.2 Dimensionality reduction of gene-to-gene distance (GGD) matrix and visualization mapping matrix by labeling specific genes

To find out appropriate gene relationship matrix, different distance matrices are explored and visualized in low dimension. The GGD matrix is firstly tried. Using the Multidimensional Scaling (MDS) by function '*mds*' and the t-Distributed Stochastic Neighbor Embedding (tSNE) by function '*tsne*' in DRToolbox, GGD matrix is mapped to a 1D matrix and a two-dimensional (2D) matrix, respectively. To visualize the dimensionality reduction results, 2D-scatterplots are made for mapped matrices.

Firstly, the scatter plot **Figure 3** is made with mapped 1D matrices produced by MDS and tSNE, where x-axis represents the mapped position of genes obtained by MDS, and y-axis represents the mapped position of genes obtained by tSNE. From this image, we can see three obvious clusters. Then, the scatter plots for two 2D mapped matrices are produced, as shown in **Figure 4** and **Figure 5**. There is no obvious distribution character in mapped result of MDS. On the other hand, there is one small cluster and two not very obvious large clusters in mapped space of t-SNE. From these two images, it suggests that the result obtained by t-SNE is better than MDS.

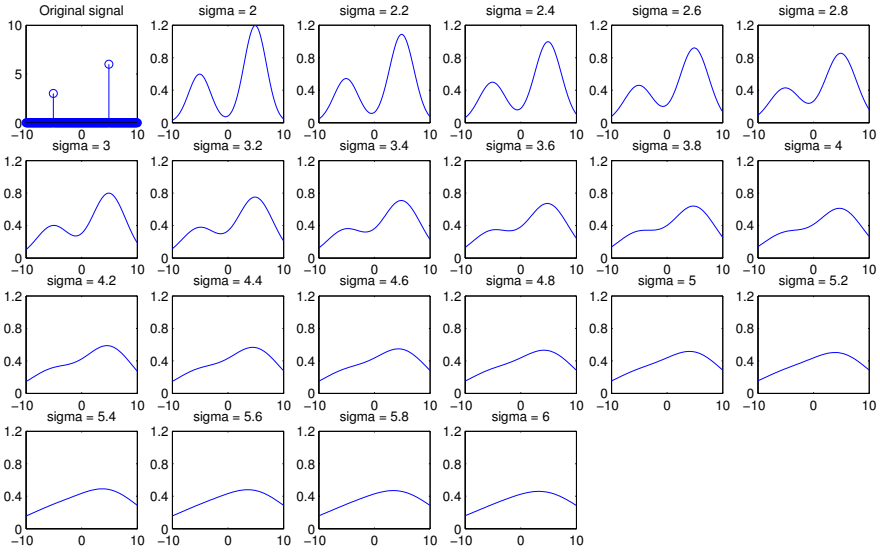

Figure 1: The convolution results of a 1D signal with two pulse

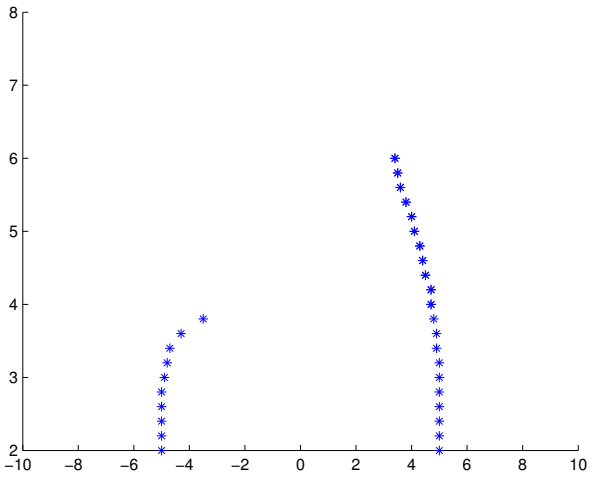

Figure 2: The scale space tree of a 1D signal with two pulse

To check whether the genes in mapping results are distributed in functional relationship, mapping matrices of GGD are visualized by labeling specific gene set. The GO term of genes are obtained corresponding gene EntrezIDs by the DAVID Functional Annotation Tool. Three GO terms used are 'membrane', 'biosynthetic process' and 'transporter activity'. We provide two labeling mapped results of GGD. Firstly, the genes with GO term 'transporter activity' are labeled by red color in the 2D mapping result produced by MDS, as shown in **Figure 6**. Secondly, the genes with GO term 'membrane' are labeled by red color meanwhile the genes with GO term 'biosynthetic process' are labeled by green color, as shown in **Figure 7**. From these two images, we can see that genes with same GO term spread over the whole space. There is no obvious cluster distribution. It means that the 2D mapping result of GGD by MDS is unsatisfactory and can not be applied as gene relationship matrix to produce scale space.

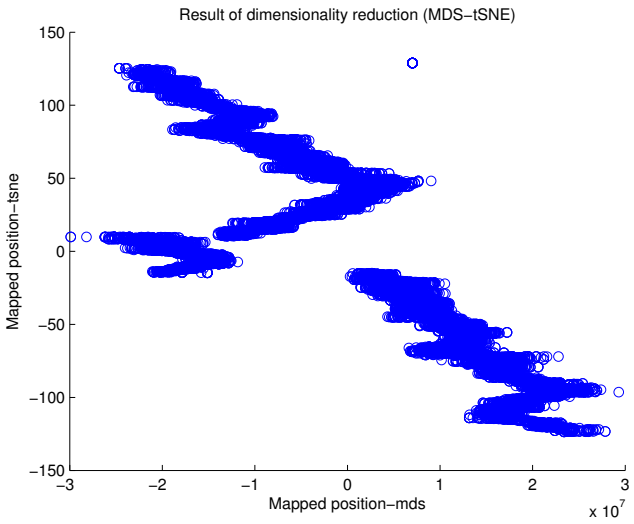

Figure 3: The mapped result of GGD obtained by MDS and tSNE

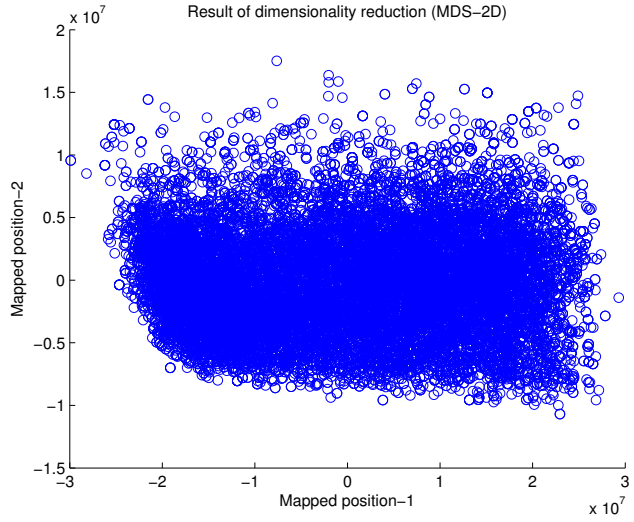

Figure 4: The mapped result of GGD obtained by MDS

We also tried labeling specific genes in mapping result of GGD matrix produced by tSNE. Genes with GO term 'transmembrane transporter activity' labeled by red color are shown in **Figure 8**. From this image, we can see that the genes with same GO term tend to cluster together. This labeled result is more satisfactory comparing to the labeled mapping result of MDS.

We find that the input of function '*tsne*' in DRToolbox is the position matrix of all points. However, our input matrix consists of the pair-wise Euclidean distances between genes. Thus, we should use the function '*tsne_d*' instead of function '*tsne*'. The scatter plot of 2D mapped result of GGD by function '*tsne_d*' is always an uniform distribution, as shown in **Figure 9**. We tried to solve this problem by adjusting parameters, such as 'perplexity'. But the result is still uniformly distributed. Finally, we find that the reason is the Euclidean distance conversion in middle step of GGD construction is failed. Thus, the mapping matrix of GGD by tSNE is also unsatisfactory to apply to produce scale space.

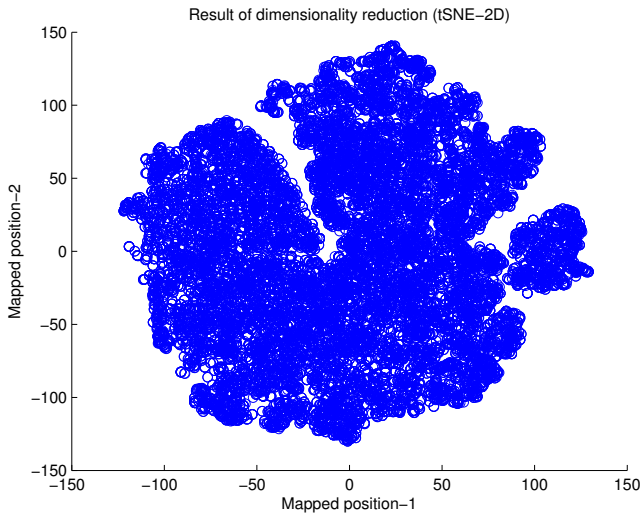

Figure 5: The mapped result of GGD obtained by tSNE

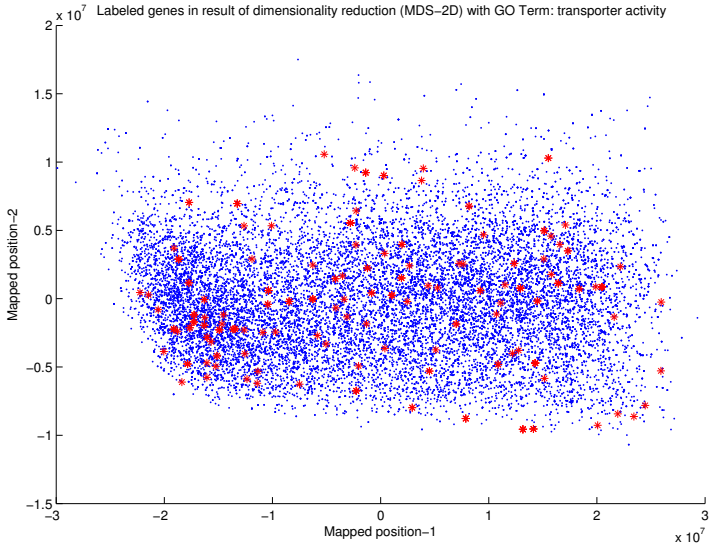

Figure 6: Labeling genes with GO term 'transporter activity' in mapping result of GGD by MDS

1.3 Dimensionality reduction of gene-to-gene correlation (GGC) matrix and visualization mapping matrix by labeling specific genes

The second relationship matrix, which has been experimented, is GGC matrix. It is computed by gene expression values. Each element in matrix is $1 - correlation$. The 1D and 2D mapped results of GGC are obtained by function '*mds*' and '*tsne_d*' in DRToolbox. The scatter plot of 1D mapping results is shown in **Figure 10**, where x-axis represents the mapped position of genes produced by MDS, y-axis represents the mapped position of genes produced by tSNE. The 2D mapping results of MDS and tSNE are shown in **Figure 11** and **Figure 12**, respectively. From these three images, we do not see obvious distribution character. It means that GGC matrix can not be used as gene

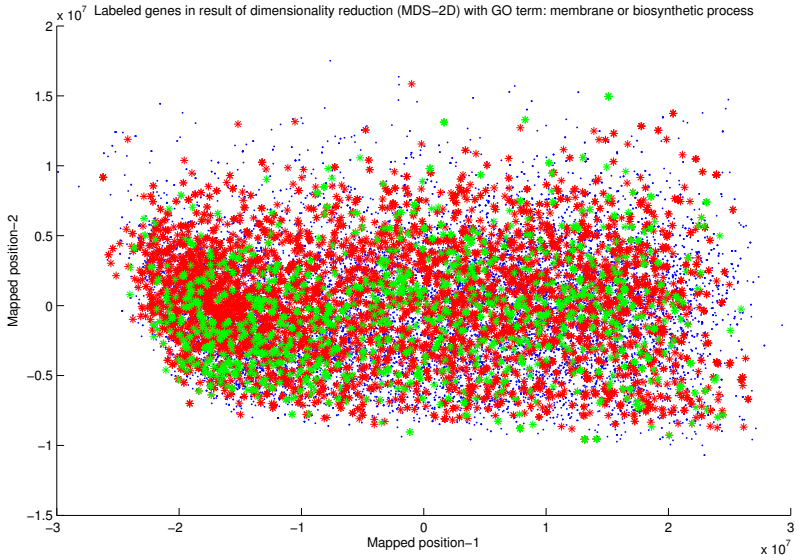

Figure 7: Labeling genes with GO term 'membrane' or 'biosynthetic process' in mapping result of GGD by MDS

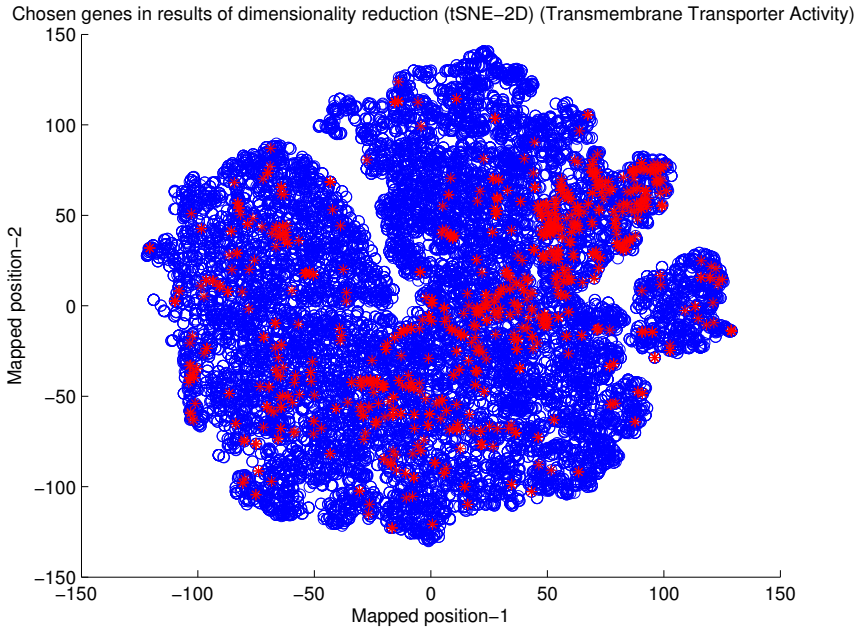

Figure 8: Labeling genes with GO term 'transmembrane transporter activity' in mapping result of GGD matrix produced by function 'tsne'

relationship matrix to build scale space.

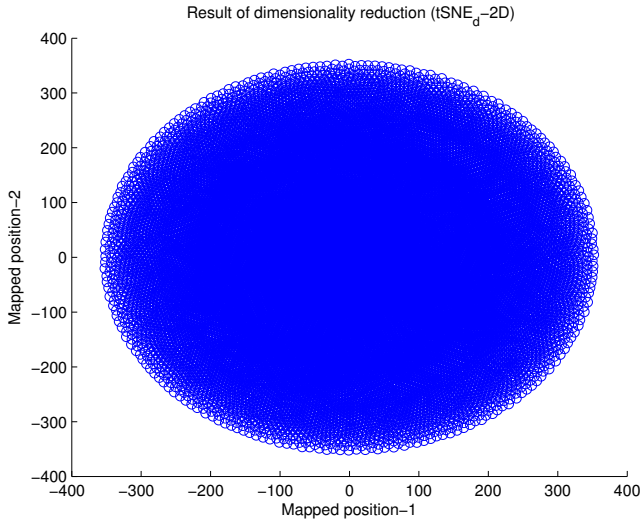

Figure 9: The mapped result of GGD obtained by function *tsne_d*'

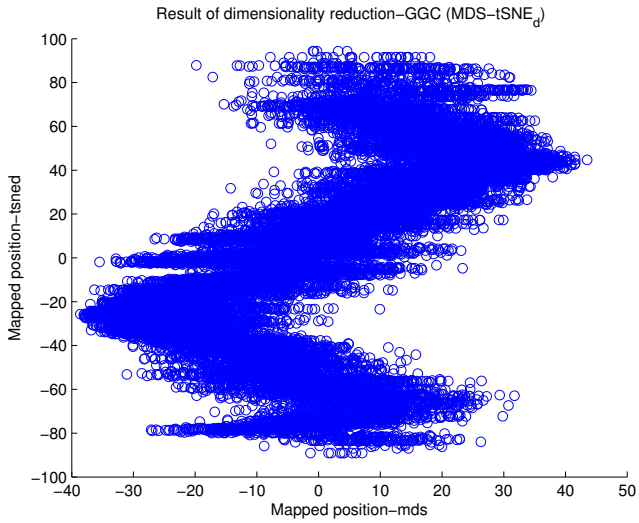

Figure 10: The mapped result of GGC obtained by MDS and tSNE

1.4 Labeling specific genes in 1D mapped result of gene-to-family distance (GFD) matrix

We know that the 2D mapped result of GFD produced by tSNE and labeled mapping result are satisfactory, as shown in Figure 6 and Figure 7 of thesis. The gene relationship matrix, which is applied to construct scale space, is one-dimensional. Thus, the 1D mapped result of GFD should be to ensure good performance before build the scale space. The 1D mapped result of GFD matrix is obtained firstly by tSNE. Then, the genes with GO term 'organic acid transmembrane transporter activity' are labeled by red in the mapped result, as shown in **Figure 13**. From this image, we can see that genes with same GO term tend to cluster together. Thus, the scale space representation can be built based on this 1D mapped result of GFD matrix.

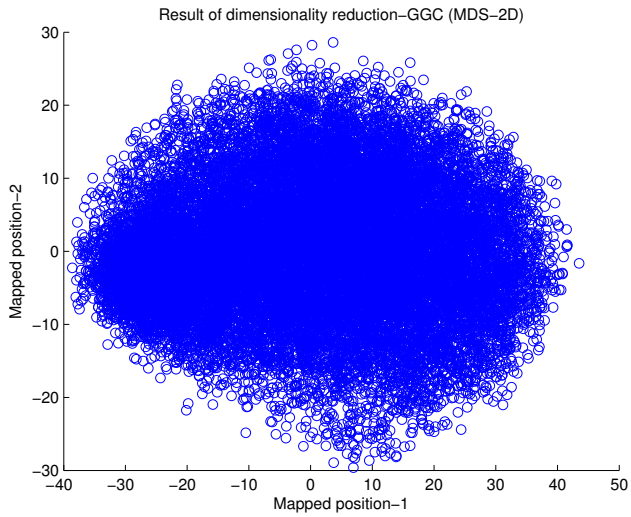

Figure 11: The 2D mapped result of GGC obtained by MDS

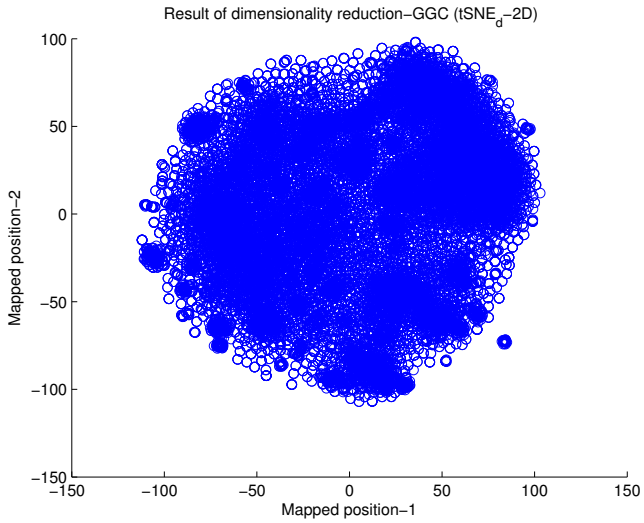

Figure 12: The 2D mapped result of GGC obtained by tSNE

1.5 Dimensionality reduction of protein-to-family distance (PFD) matrix and visualization mapping matrix by labeling specific genes

To check whether the tSNE are applied correctly to reduce high dimensionality, we reduce the dimensionality of PFD with function '*tsne*' in DRToolbox and compare to the mapped result shown in [1]. The mapped result is a uniform distribution. Then, we tried function '*compute_mapping*' in Matlab to implement tSNE. The mapped result is similar to the result obtained in [1], as shown in **Figure 14**. After comparing the code and data used in our study to [1], we find that the distance value in PFD matrix is 'single' data type but not original 'double' data type. Then, we apply the function '*tsne*' on PFD matrix of 'double' data type, and obtain a satisfactory mapping result, as shown in **Figure 15**.

To check whether the labeling method of genes used in our study is correct, we use this method on mapping result of PFD matrix, and compare to the labeling results shown in [1]. By using the

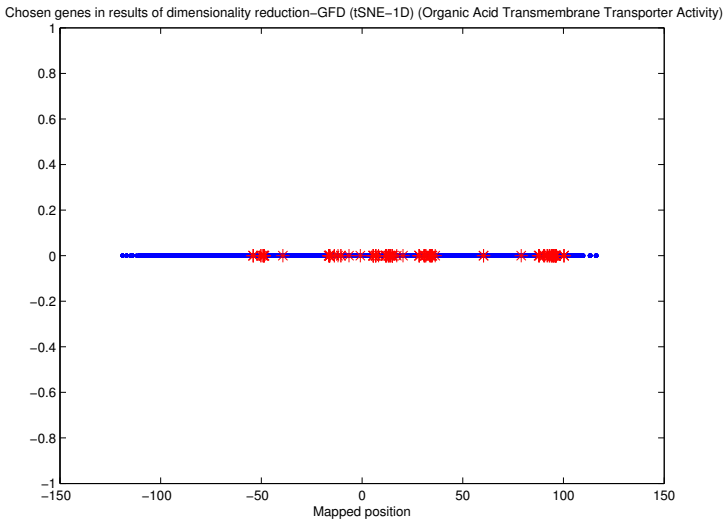

Figure 13: Labeled genes in 1D mapped result of GFD matrix

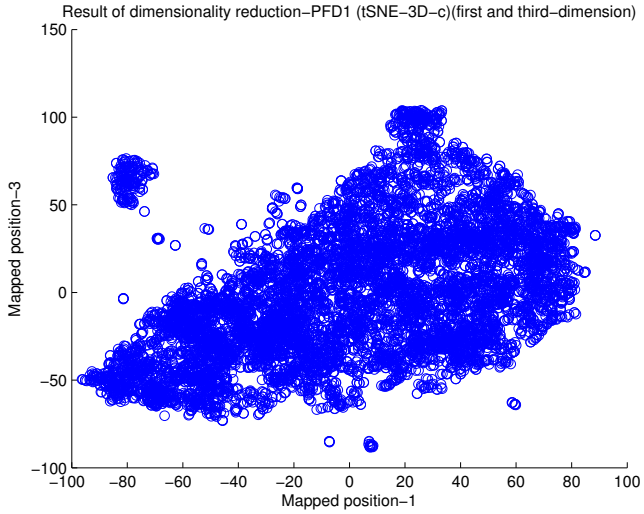

Figure 14: Mapping result of PFD matrix obtained by function 'compute_mapping'

DAVID Functional Annotation Tool, we input the EnsemblTranscriptID list of proteins, and obtain a functional annotation summary. Firstly, we choose the annotation category 'GOTERM-MF-ALL', and select the protein set with GO term 'transmembrane transporter activity'. The labeled proteins with this GO term are shown in **Figure 16**. This labeled result is similar with the result (Fig 3.3) shown in [1]. Then, we choose the annotation categories 'GOTERM-CC-ALL' and 'GOTERM-BP-ALL', and select the protein sets with GO terms 'plasma membrane' and 'regulation of biosynthetic process', respectively. These two protein sets are labeled by different colors, as shown in **Figure 17**, where the red points represent proteins with GO term 'plasma membrane', and the green points represent proteins with GO term 'regulation of biosynthetic process'. This labeled result is also similar with the result (Fig 3.1) shown in [1]. From these two image, we can see that the labeling method used in our study is acceptable.

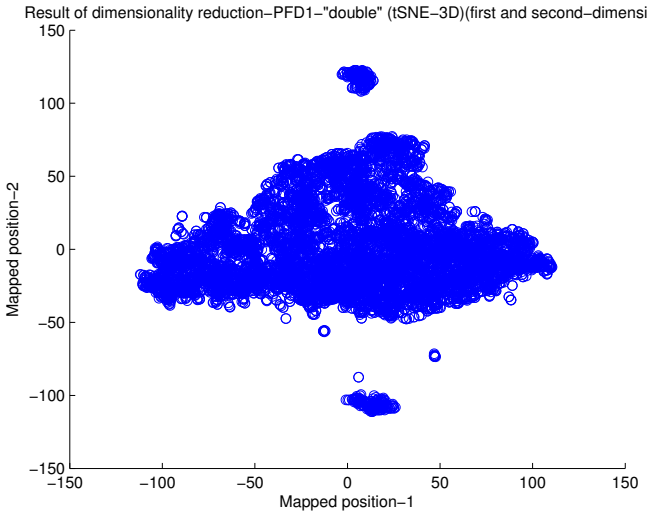

Figure 15: Mapping result of PFD matrix of 'double' data type obtained by function 'tsne'

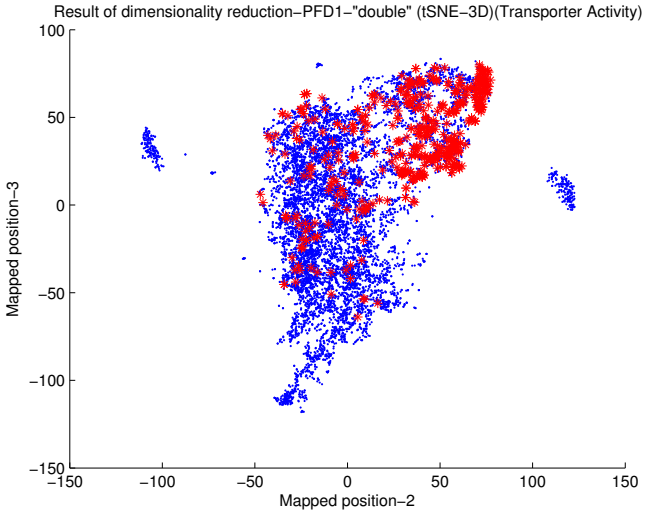

Figure 16: Labeling proteins with GO term 'transmembrane transporter activity' in mapping result of PFD

1.6 Feature selection based on scale space data

To explore the differences between scale space trees of two sample classes, we firstly split the scale space dataset to two subset according the label list. Then, the probability density estimations of trees for two sample classes in 8 single scale levels are shown in **Figure 18**. From the image, we can see that in lower scale levels $(1 : 40)$ the differences between trees in two classes are very small, while in higher scale levels there are obvious differences between trees of two patient classes.

The density estimations for two original classes, where 'poor' class consists of 37 samples, while 'good' class consists of 156 samples, in 8 single scale levels are shown in **Figure 19**. By comparing to the **Figure 18**, we can see that the phenomena mentioned above is unobvious in **Figure 19**. In this aspect, the preprocessing, which produces two sample classes with similar sizes, is certainly necessary.

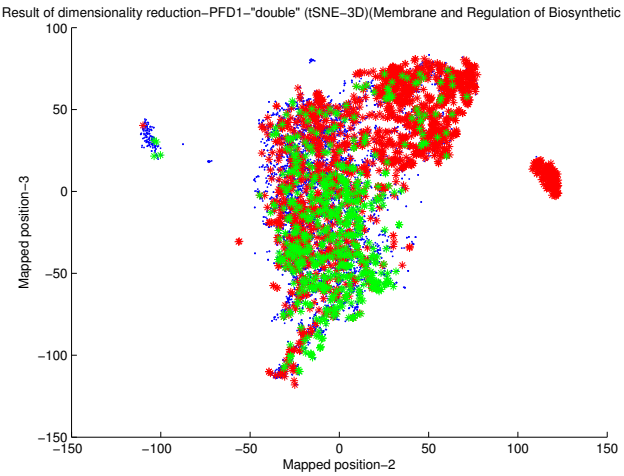

Figure 17: Labeling proteins with GO term 'plasma membrane' or 'regulation of biosynthetic process' by different colors in mapping result of PFD

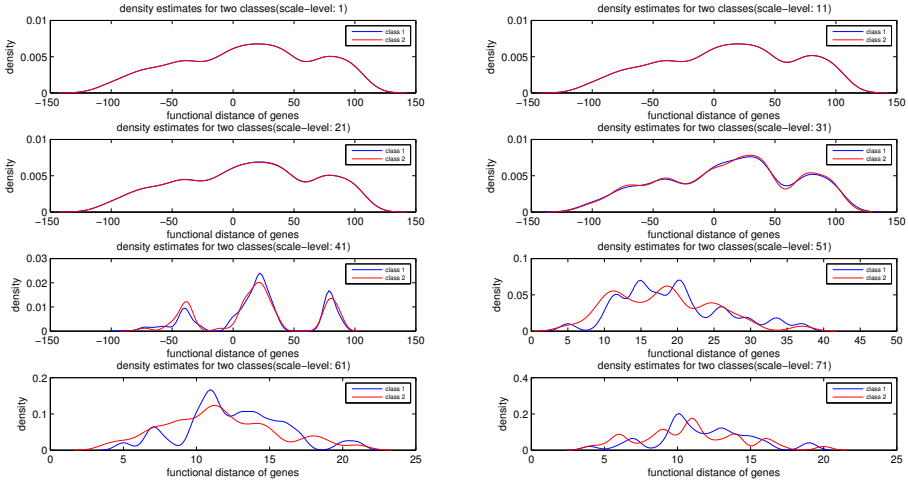

Figure 18: The density estimation of trees in two sample classes with same class-size in 8 single scales(1:10:71)

T-test is another method to find significant feature for classification based on scale space. Firstly, one gene in one scale level can be seen as a feature, thus there are 144000 (2000*72) features in our scale space data. Comparisons between two patient classes based on these 144000 features can be obtained by function '*ttest2*' in Matlab. The results are 144000 p-values. The feature with small p-value can be seen as a significant feature. To visualize these significant features in scale space tree, the average scale space tree of all samples was built, and colored by logarithmic p-values to pay more attention to smaller p-value, as shown in **Figure 20**. The points with blue color, are significant features.

In scale space tree of position data of 2000 genes, the genes in one scale level with p-value smaller than 0.02 are marked by red color, as shown in **Figure 21**. 144 significant features are found out

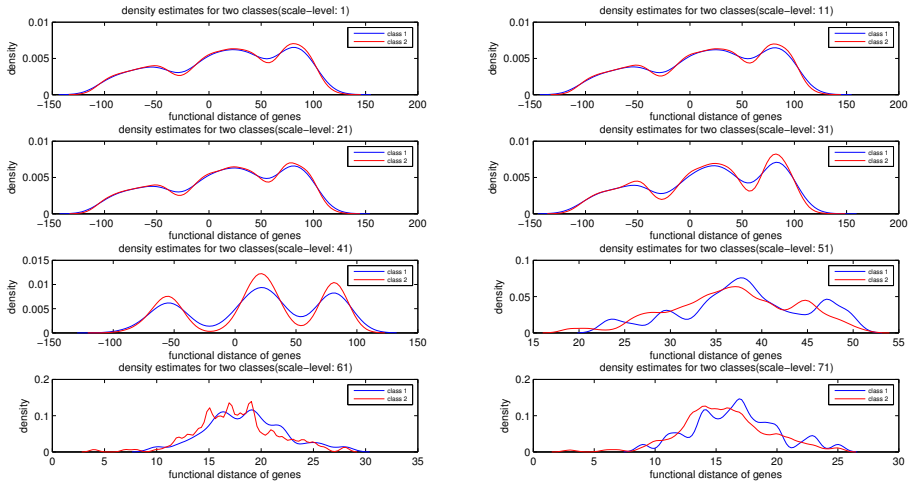

Figure 19: The density estimation of trees in two sample classes with different class-sizes in 8 single scales(1:10:71)

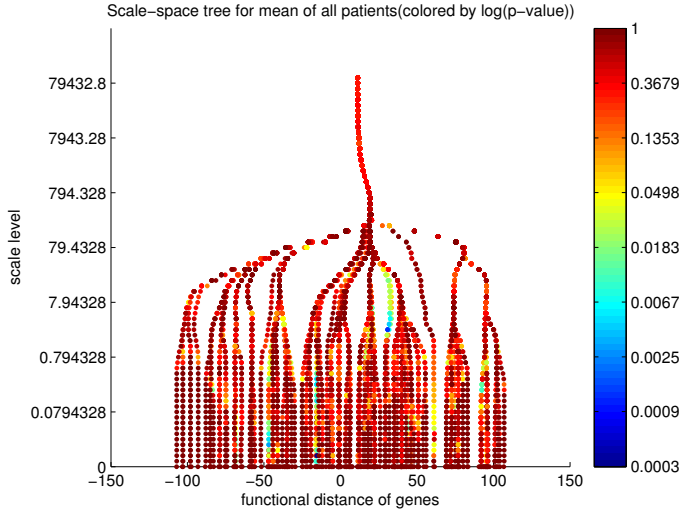

Figure 20: The scale space tree for mean of all patient samples (colored by logarithmic p-value)

after removing duplicated genes. While, in scale space tree of expression-level data of 2000 genes, one gene in one scale level with p-value smaller than 0.01 was marked by red color in **Figure 22**. 51 significant features are found out.

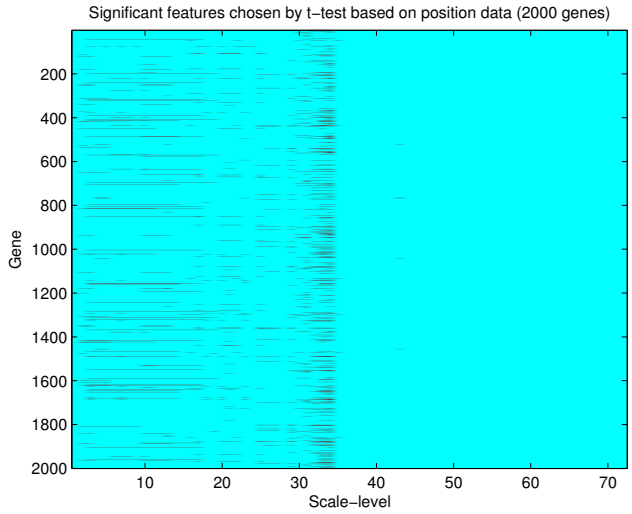

Figure 21: Significant features chosen by t-test based on position data of genes (threshold by 0.02)

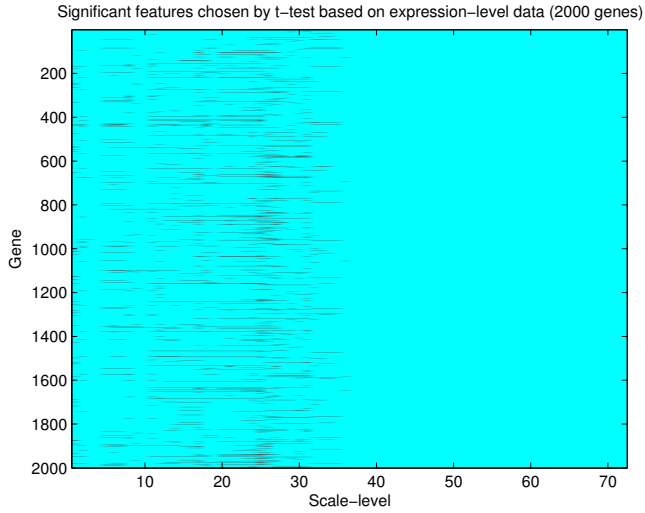

Figure 22: Significant features chosen by t-test based on expression-level data of genes (threshold by 0.01)

2 Matlab code applied in thesis work

Main_code.m

```
% Molecular Classification on Breat cancer


%% Getting original data
load DMMPLCN_080423

subset = pre_proc(DMMPLCN, 'Miller');
data_74 = subset.data;
label_74 = subset.nlab;
```

```
% Getting the data_retain
[data_74_retain1, entrezid_74_retain1] = gen_entrezidlist(data_74', DMMPLCN);
[data_74_retain, entrezid_74_retain, ensgid_74_retain, protein_74_retain] =
    gen_proteinlist(data_74_retain1, entrezid_74_retain1);


%% Classification based on original data
data = data_74_retain';
var_gene = var(data);
[var_gene_s, index_var] = sort(var_gene, 'descend');
data_part = data(:, index_var(1 : 2000));
data_part = dataset(data_part, label_74);

% cross-validation
e_ori = crossval(data_part, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3),
    loglc}, 10, 5);
e_ori = crossval(data_part, {nmc}, 10, 5);

% feature extraction
% PCA
[e_ori_pca, dim_ori_pca] = crossval_featsel(data_part, pca, 0.9, {nmc, ldc,
    qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);
% LDA
[e_ori_lda, dim_ori_lda] = crossval_featsel(data_part, fisherm, [], {nmc, ldc,
     qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);

% dissimilarity representation
% option 1: first PCA then dissimilarity representation
w_pca = pca(data_part);
data_pca = data_part * w_pca;
% L1: CityBlock distance
w = data_pca * proxm([], 'c');
dis_ori_l1 = data_pca * w;
% L2: Euclidean distance
w = data_pca * proxm([], 'd');
dis_ori_l2 = data_pca * w;
% cross-validation
e_ori_disl1_pca = crossval(dis_ori_l1, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);
e_ori_disl2_pca = crossval(dis_ori_l2, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);

% option 2: only dissimilarity representation
% L1: CityBlock distance
w = data_part * proxm([], 'c');
dis_ori_l1 = data_part * w;
% L2: Euclidean distance
w = data_part * proxm([], 'd');
dis_ori_l2 = data_part * w;
% cross-validation
e_ori_disl1 = crossval(dis_ori_l1, {nmc, ldc, qdc, fisherc, parzenc, knnc([],
    3), loglc}, 10, 5);
e_ori_disl2 = crossval(dis_ori_l2, {nmc, ldc, qdc, fisherc, parzenc, knnc([],
    3), loglc}, 10, 5);


% feature selection
% feature set
[fea_ori_for,list_ori_for] = fsel(data_part, 'forward', 'NN', 100);
```

```
% feature selection and cross-validation (change function 'crossval_feasel')
[e_ori_featself, dim_ori_featself] = crossval_featsel(data_part, featself,
    100, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);

% feature curve
fc_nmc = clevalf(data_part, nmc, [10, 20, 40, 80, 160, 320, 640, 1280, 2000],
    [], 2);
fc_ldc = clevalf(data_part, ldc, [10, 20, 40, 80, 160, 320, 640, 1280, 2000],
    [], 2);

%fc = clevalf(data_part, {nmc, ldc, qdc}, [10, 20, 40, 80, 160, 320, 640,
    1280, 2000], 0.9, 2);
fc = clevalf(data_part, {nmc, ldc, qdc}, [], 0.9, 5);

plote(fc_nmc,'g');
legend('nmc','location','best');
hold on
plote(fc_ldc,'b');
legend('ldc','location','best');
hold on
plote(fc);


%% Classification based on scale-space data

% construct the scale-space tree
clear
load data_74_retain
load mapped_gfd_tsne1

data = data_74_retain';
var_gene = var(data);
[var_gene_s, index_var] = sort(var_gene, 'descend');

data_part = data(:, index_var(1 : 2000));
gene_part = mapped_gfd_tsne1(index_var(1 : 2000));

H = logspace(-2, 5, 71);% -5/-2 100/71

p = gene_part';
xclust = cell(length(data_part(:, 1)), 1);
fclust = cell(length(data_part(:, 1)), 1);
mclust = cell(length(data_part(:, 1)), 1);

for i = 1 : length(data_part(:, 1))% calculate scale-space for each patient
    g = abs(data_part(i, :));
    [xclust{i} fclust{i} mclust{i}] = gkc_MS_1D_clust(p, g, H);
end

sst_p = zeros(length(data_part(:, 1)), length(H) + 1, length(p));%
    (74*72*2000)
sst_e = zeros(length(data_part(:, 1)), length(H) + 1, length(p));

for i = 1 : length(data_part(:, 1))
    sst_p(i, 1, :) = gene_part;
    sst_e(i, 1, :) = data_part(i, :);
end

for i = 1 : length(data_part(:, 1))
```

```
    for j = 1 : length(H)
        for k = 1 : length(p)
            sst_p(i, j + 1, k) = xclust{i}{j}(mclust{i}{j}(k));
            sst_e(i, j + 1, k) = fclust{i}{j}(mclust{i}{j}(k));
        end
    end
end

% visualization
scale_level = H;
sst_p1 = sst_p(1, :, :);
sst_p1 = squeeze(sst_p1);
y = (1 : length(H) + 1);

figure; plot(sst_p1, y, 'b')

axis([-150 150 1 length(H) + 10])
set(gca,'Ytick', 1 : 10 : length(H))
set(gca,'YtickLabel', {0, scale_level(10), scale_level(20), scale_level(30),
    scale_level(40), scale_level(50), scale_level(60), scale_level(70)})
xlabel('functional distance of genes')
ylabel('scale level')
title('Scale-space tree for one patient')


% construct the dataset based on scale-space tree
load label_74

% new matrix
sst_p = sst_p(:, 1:2:end, :);
sst_e = sst_e(:, 1:2:end, :);

% matrix (sst_p or sst_e) transpose (from 74*72*2000 to 74*2000*72)
sst_p_t = zeros(length(sst_p(:, 1, 1)), length(sst_p(1, 1, :)), length(sst_p
    (1, :, 1)));
sst_e_t = zeros(length(sst_e(:, 1, 1)), length(sst_e(1, 1, :)), length(sst_e
    (1, :, 1)));

for i = 1 : length(sst_p(:, 1, 1))
    sst_p_t(i, :, :) = squeeze(sst_p(i, :, :))';
    sst_e_t(i, :, :) = squeeze(sst_e(i, :, :))';
end

sst_p_2d = sst_p_t(:, :);
sst_e_2d = sst_e_t(:, :);
sst_p_dat = dataset(sst_p_2d, label_74);
sst_e_dat = dataset(sst_e_2d, label_74);



% feature extraction
% PCA
[e_ss_p_pca, dim_ss_p_pca] = crossval_featsel(sst_p_dat, pca, 0.9, {nmc, ldc,
    qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);
[e_ss_e_pca, dim_ss_e_pca] = crossval_featsel(sst_e_dat, pca, 0.9, {nmc, ldc,
    qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);
% LDA
[e_ss_p_lda, dim_ss_p_lda] = crossval_featsel(sst_p_dat, fisherm, [], {nmc,
    ldc, qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);
```

```
[e_ss_e_lda, dim_ss_e_lda] = crossval_featsel(sst_e_dat, fisherm, [], {nmc,
    ldc, qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);


% dissimilarity representation
% L1: CityBlock distance
w1_p = sst_p_dat * proxm([], 'c');
dis_ss_p_l1 = sst_p_dat * w1_p;
w1_e = sst_e_dat * proxm([], 'c');
dis_ss_e_l1 = sst_e_dat * w1_e;
% L2: Euclidean distance
w2_p = sst_p_dat * proxm([], 'd');
dis_ss_p_l2 = sst_p_dat * w2_p;
w2_e = sst_e_dat * proxm([], 'd');
dis_ss_e_l2 = sst_e_dat * w2_e;

% cross-validation
e_ss_p_disl1 = crossval(dis_ss_p_l1, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);
e_ss_e_disl1 = crossval(dis_ss_e_l1, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);

e_ss_p_disl2 = crossval(dis_ss_p_l2, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);
e_ss_e_disl2 = crossval(dis_ss_e_l2, {nmc, ldc, qdc, fisherc, parzenc, knnc
    ([], 3), loglc}, 10, 5);

e_ss_p = crossval(sst_p_dat, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3),
    loglc}, 10, 5);
e_ss_e = crossval(sst_e_dat, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3),
    loglc}, 10, 5);



% feature selection
% feature set
[fea_ss_p_ind,list_ss_p_ind] = fsel(sst_p_dat, 'individual', 'NN', 2000);
[fea_ss_e_ind,list_ss_e_ind] = fsel(sst_e_dat, 'individual', 'NN', 2000);

sst_p_dat_ind = sst_p_dat * fea_ss_p_ind;
sst_e_dat_ind = sst_e_dat * fea_ss_e_ind;
% 100; 50; 10; 20; 30; 40; 60; 70; 80; 90
[fea_ss_p_for,list_ss_p_for] = fsel(sst_p_dat_ind, 'forward', 'NN', 90);
[fea_ss_e_for,list_ss_e_for] = fsel(sst_e_dat_ind, 'forward', 'NN', 90);

% feature selection and cross-validation
[e_ss_p_featself, dim_ss_p_featself] = crossval_featsel(sst_p_dat, featself,
    100, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);
[e_ss_e_featself, dim_ss_e_featself] = crossval_featsel(sst_e_dat, featself,
    100, {nmc, ldc, qdc, fisherc, parzenc, knnc([], 3), loglc}, 10, 5);

% feature curve
fc_p_nmc = clevalf(sst_p_dat_ind, nmc, [10, 20, 40, 80, 160, 320, 640, 1280,
    2000], [], 2);
fc_e_nmc = clevalf(sst_e_dat_ind, nmc, [10, 20, 40, 80, 160, 320, 640, 1280,
    2000], [], 2);

%fc_p = clevalf(sst_p_dat_ind, {nmc, ldc, qdc}, [10, 20, 40, 80, 160, 320,
    640, 1280, 2000], 0.9, 2);
```

```
fc_p = clevalf(sst_p_dat_ind, {nmc, ldc, qdc}, [], 0.9, 2);
%fc_e = clevalf(sst_e_dat_ind, {nmc, ldc, qdc}, [10, 20, 40, 80, 160, 320,
    640, 1280, 2000], 0.9, 2);
fc_e = clevalf(sst_e_dat_ind, {nmc, ldc, qdc}, [], 0.9, 2);
plote(fc_p_nmc,'g');
legend('nmc','location','best');
```

Preprocessing.m

```
function data_new = pre_proc(data, setname)

% Input:  data        matrix of data will be processed
%         setname     string of the name of feature selection algorithm will be
     used

sc1 = strcmp(setname, 'Desmedt');
sc2 = strcmp(setname, 'Miller');
sc3 = strcmp(setname, 'Minn');
sc4 = strcmp(setname, 'Pawitan');
sc5 = strcmp(setname, 'Loi');
sc6 = strcmp(setname, 'Chin');

if sc1 == 1
    subdata = data.data(:, 1 : 147);% Desmedt
    % build the class label(t_dmfs,e_dmfs)
    t = data.Clinical.t_dmfs(1 : 147);
    e = data.Clinical.e_dmfs(1 : 147);
end

if sc2 == 1
    subdata = data.data(:, 148 : 394);% Miller
    % build the class label(t_sos,e_sos)
    t = data.Clinical.t_sos(148 : 394);
    e = data.Clinical.e_sos(148 : 394);
end

if sc3 == 1
    subdata = data.data(:, 395 : 490);% Minn
    % build the class label(t_dmfs,e_dmfs)
    t = data.Clinical.t_dmfs(395 : 490);
    e = data.Clinical.e_dmfs(395 : 490);
end

if sc4 == 1
    subdata = data.data(:, 491 : 646);% Pawitan
    % build the class label(t_sos,e_sos)
    t = data.Clinical.t_sos(491 : 646);
    e = data.Clinical.e_sos(491 : 646);
end

if sc5 == 1
    subdata = data.data(:, 647 : 824);% Loi
    % build the class label(t_dmfs,e_dmfs)
    t = data.Clinical.t_dmfs(647 : 824);
    e = data.Clinical.e_dmfs(647 : 824);
end

if sc6 == 1
    subdata = data.data(:, 825 : 947);% Chin
```

```
    % build the class label(t_dmfs,e_dmfs)
    t = data.Clinical.t_dmfs(825 : 947);
    e = data.Clinical.e_dmfs(825 : 947);
end

len_s = length(subdata(1, :));
label = zeros(1, len_s)-1;
for i = 1 : len_s
    if t(i) <= 60 && e(i) == 1% e=1 not censoring
        label(i) = 2;% poor
    elseif t(i) > 60 && e(i) == 0% e=0 censoring
        label(i) = 3;% good
    else
        label(i) = NaN;
    end
end

% build a new dataset with same size of different class
IKeep_g = find(label == 3);
IKeep_p = find(label == 2);
len_g = length(IKeep_g);
len_p = length(IKeep_p);

if len_g >= len_p
    t_g = t(label == 3);
    [t_g_s, index_t_g] = sort(t_g, 'descend');
    label(IKeep_g(index_t_g(1 : len_p))) = 1;% good
    %label(IKeep_g(1 : len_p)) = 1;% good
    label(IKeep_p) = 0;% poor
else
    t_p = t(label == 2);
    [t_p_s, index_t_p] = sort(t_p);
    label(IKeep_p(index_t_p(1 : len_g))) = 0;% poor
    %label(IKeep_p(1 : len_g)) = 0;% poor
    label(IKeep_g) = 1;% good
end

IKeep = label==1 | label==0;
label_new = label(IKeep);
subdata_new = subdata(:, IKeep);
data_new = dataset(subdata_new',label_new');
```

Generate_entrezidlist.m

```
function [data_retain1, entrezid_retain1] = gen_entrezidlist(data,
    data_entrezid)

% Input:  data              Data of data will be processed.
%         data_entrezid     Original data with entrezid list.
%
% Output: data_retain1      New dataset that is removed genes with 'NaN' in
%                           original entrezid list.
%         entrezid_retain1  New entrezid list that is removed genes with
%                           'NaN' in original entrezid list.

entrezid = data_entrezid.Entrez;
retain1 = find(~isnan(entrezid));
entrezid_retain1 = entrezid(retain1);
data_retain1 = data(retain1, :);
```

Generate_proteinlist.m

```
function [data_retain, entrezid_retain, ensgid_retain, protein_retain] =
    gen_proteinlist(data_retain1, entrezid_retain1)

% Build corresponding EnsemblGeneID list for retained genes
fid = fopen('conv.txt', 'r');
x = textscan(fid,'%s %s');
fclose(fid);
entrezid_convert = cellfun(@(x) str2double(x(1 : end)), x{1, 1});
ensgid_convert = cellfun(@(x) str2double(x(10 : end)), x{1, 2});
ensgid_pre = zeros(length(entrezid_retain1), 945);
% 12518 unique EntrezID; 13463 obtained EnsemblGeneID;13463-12518=945

for i = 1 : length(entrezid_retain1)
    c = 1;
    for j = 1 : length(entrezid_convert)
        if entrezid_retain1(i) == entrezid_convert(j)
            ensgid_pre(i, c) = ensgid_convert(j);
            c = c + 1;
        end
    end
    if c == 1
        ensgid_pre(i, 1) = NaN;
    end
end

% Remove all-zero columns in EnsemblGeneID list(result: max=16)
q = 0;

for i = 1 : 945
    num = find(ensgid_pre(:, i) ~= 0);
    if isempty(num) == 0
        q = q + 1;
    else
        break
    end
end

ensgid = ensgid_pre(:, 1 : q);
% Remove genes with 'unknown' EntrezIDs or EntrezIDs converted
% unsuccessfully to EnsemblGeneID
retain2 = find(~isnan(ensgid(:, 1)));
entrezid_retain2 = entrezid_retain1(retain2);
ensgid_retain2 = ensgid(retain2, :);
data_retain2 = data_retain1(retain2, :);


% Using EnsemblGeneID list, get the ProteinNumber (defined in Sepideh's data)
    list
% Build corresponding ProteinNumber (defined in Sepideh's data) list for
    retained genes
fid=fopen('file_map.txt', 'r');
x=textscan(fid,'%s %s');
fclose(fid);
pid_s=cellfun(@(x) str2double(x(2:end)),x{1, 1});
ensgid_s=cellfun(@(x) str2double(x(10:15)),x{1, 2});
protein_pre = zeros(length(ensgid_retain2));
```

```
for i = 1 : length(ensgid_retain2)
    a = 1;
    for j = 1 : length(ensgid_retain2(1, :))
        if ensgid_retain2(i, j) ~= 0
            for k = 1 : length(ensgid_s)
                if ensgid_retain2(i, j) == ensgid_s(k)
                    protein_pre(i, a) = pid_s(k);
                    a = a + 1;
                end
            end
        else
            break
        end
    end
    if a == 1
        protein_pre(i, 1) = NaN;
    end
end

% Remove all-zero columns in ProteinNumber list (result: max=78)
q = 0;

for i = 1 : length(protein_pre)
    nump = find(protein_pre(:, i) ~= 0);
    if isempty(nump) == 0
        q = q + 1;
    else
        break
    end
end

protein = protein_pre(:, 1 : q);

% Remove genes that have not corresponding proteins
retain = find(~isnan(protein(:,1)));
protein_retain = protein(retain,:);
entrezid_retain = entrezid_retain2(retain);
ensgid_retain = ensgid_retain2(retain,:);
data_retain = data_retain2(retain,:);
```

Crossvalidation_featureseletion_function.m

```
function [err, dim] = crossval_featsel(data, featsel, featsel_p, classifier,
    num_folds_c, num_times)
%function [err, dim] = crossval_featsel(data, featsel, featsel_p1, featsel_p2,
     classifier, num_folds_c, num_times)

% Input:  data          Dataset of data will be processed.
%         featsel       The name of feature selection algorithm
%                       will be used (e.g. pca, fisherm, featseli,
%                       featself, featselb, featsellr, featselb).
%         featsel_p     The parameter of feature selection algorithm.
%        (featsel_p1    The first parameter of feature selection algorithm.)
%        (featsel_p2    The second parameter of feature selection algorithm.)
%         classifier    Cell of the name of classifier will be used. (e.g.
%                       {nmc}, {ldc}, {qdc}, {fisherc}, {parzenc}, {loglc}
%                       {knnc([],3)}).
%         num_folds_c   The number of folds the data will be divided for
%                       classification.
```

```
%          num_times      The number of repetitions.
%
% Output:  err            Classification error
%          dim            Dimension of feature space after feature selection
%
% This function is used to obtain unbiased error estimation by
% cross-validation after feature selection. The classifier is trained on
% 'num_folds_c'-1 parts and the remaining part is used for testing. This is
% rotated over all folds. Error of this classifier is the avarage of all
% error result for each fold. If the 'num_times' > 1, the result 'err' is
% the mean error over all repetitions.




% Generate prior
data = setprior(data,getprior(data));

% Check the feature selection will be used
%if ~iscell(featsel)
%    featsel = {featsel};
%end

% Check the classifier will be used
if ~iscell(classifier)
    classifier = {classifier};
end

num_f = length(featsel);
num_c = length(classifier);

seed = rand('state');
rand('state', seed);

dim_t = zeros(1, 1);
error_t = zeros(1, num_c);

for i = 1 : num_times

    % Check the input dataset
    if ~isdataset(data)
        fprintf('Warning: The type of input data is not "dataset"')
        break;
    end

    % Check the number of feature selection algorithm
    if num_f ~= 1
        fprintf('Warning: The number of feature selection algorithm used
            should be one')
        break;
    end

    % Check the number of repetations
    if num_times < 1
        fprintf('Warning: The number of repetations should be large than 0')
        break;
    end

    len_s = length(data);
```

```
% Check the number of folds divided for classification
if num_folds_c > len_s
    fprintf('Warning: Number of folds for classification too large: reset
        to leave-one-out')
            num_folds_c = len_s;
    elseif num_folds_c <= 1
            fprintf('Warning: Wrong number of divided folds for
                classification')
    break;
end



% Generate prior
    %data = setprior(data,getprior(data));

%seed = rand('state');
%rand('state', seed);

error_c = zeros(1, num_c);
dim_c = zeros(1, 1);

s = getsize(data);
c = classsizes(data);
J = randperm(s(1));

if all(c >= num_folds_c)
    K = zeros(1, s(1));

    for l = 1 : length(c)
        L = findnlab(data(J, :), l);
        M = mod(0 : c(l) - 1, num_folds_c) + 1;
        K(L) = M;
    end

else
    K = mod(1 : s(1), num_folds_c) + 1;
end


for k = 1 : num_folds_c
    % Generate the training dataset and testing dataset
    OUT = find(K == k);
    JOUT = J(OUT);
    JIN = J;
    JIN(OUT) = [];
    train = data(JIN, :);
    test = data(JOUT, :);

    %train = setprior(train,getprior(train));
    %test = setprior(test,getprior(test));

    %seed = rand('state');
    %rand('state', seed);

    %s1 = num2str('featsel');
    %s2 = num2str('pca');
    %s3 = num2str('fisherm');
    %sc1 = strcmp(s1, s2);
```

```
        %sc2 = strcmp(s1, s3);

        %if sc1 == 1 || sc2 == 1
        %    w1 = featsel(train, featsel_p);
        %else
        %    w1 = featsel(train, [], featsel_p);
        %end

        %w1 = featsel(train, featsel_p);% pca, fisherm
        w1 = featsel(train, [], featsel_p);% featseli, featself, featselb ...
        %w1 = featsel(train, featsel_p1, featsel_p2);
        s_dim = getsize(w1);
        dimension = s_dim(2);
        dim_c = dim_c + dimension;

        mapped_train = train * w1;
        mapped_test = test * w1;

        for m = 1 : num_c
            w2 = mapped_train * classifier{m};
            error = testc(mapped_test, w2);
            error_c(m) = error_c(m) + error;
        end

    end

    dim_c = dim_c / num_folds_c;
    dim_t = dim_t + dim_c;
    error_c = error_c / num_folds_c;
    error_t = error_t + error_c;

end

dim = dim_t / num_times;
dim = round(dim);
err = error_t / num_times;
```

Generate_gfd_matrix.m

```
function gfd = gen_gfd(entrezid_retain, protein_retain)

% Build PFD matrix (e-value)
load ./bayue/evalue1
load ./bayue/evalue2
load ./bayue/evalue3
load ./bayue/evalue4
load ./bayue/evalue5
load ./bayue/evalue6
load ./bayue/evalue7
load ./bayue/evalue8

evalue_pre = [evalue1 evalue2 evalue3 evalue4 evalue5 evalue6 evalue7 evalue8
    ];
pfd_retain = evalue_pre';% transpose to (protein*family)

% Build the gene-to-protein relationship matrix (19844*55106)
load ./bayue/p
gene_protein = zeros(length(entrezid_retain), length(p));
```

```
for i = 1 : length(protein_retain(:, 1))
    for j = 1 : length(protein_retain(1, :))
        if protein_retain(i, j) ~= 0
            a = p == protein_retain(i, j);
            gene_protein(i, a) = 1;
        else
            break
        end
    end
end

% third step: build the GFD matrix (19844*11912)
gfd = zeros(length(entrezid_retain), length(pfd_retain(1, :)));

for i = 1 : length(entrezid_retain)
    p_id = gene_protein(i, :) == 1;
    p_sub = pfd_retain(p_id, :);
    for j = 1 : length(pfd_retain(1, :))
        gfd(i, j) = min(p_sub(:, j));
    end
end
```

Produce_scale_space_tree_for_one_patient.m

```
% FIRST STEP: choose a part of genes (2000 most variable genes)
clear
load data_retain
data = data_retain';
var_gene = var(data);
[var_gene_s, index_var] = sort(var_gene, 'descend');

%SECOND STEP: combine the mapped matrix with measurement value (gene
    expression value) of all patients
load mapped_gfd_tsne1
data_part = data(:, index_var(1 : 2000));
gene_part = mapped_gfd_tsne1(index_var(1 : 2000));

% THIRD STEP: Construct the scale-space tree for every patient
% 72 scale levels (0) and (0.01, ..., 10^5)
H = zeros(1, 71);

for a = -2 : 0.1 : 5
    H(int8(a * 10 + 21)) = exp(reallog(10) * a);
end

p = gene_part';
xclust = cell(length(data_part(:, 1)), 1);
fclust = cell(length(data_part(:, 1)), 1);
mclust = cell(length(data_part(:, 1)), 1);

for i = 1 : length(data_part(:, 1))% calculate scale-space for each patient
    g = abs(data_part(i, :));
    [xclust{i} fclust{i} mclust{i}] = gkc_MS_1D_clust(p, g, H);
end

sst_p = zeros(length(data_part(:, 1)), length(H) + 1, length(p));%
    (193*72*2000) or (74*72*2000) or (74*72*2000)
sst_e = zeros(length(data_part(:, 1)), length(H) + 1, length(p));
%map = zeros(length(data_part(:, 1)), length(H) + 1, length(p));
```

```
for i = 1 : length(data_part(:, 1))
    sst_p(i, 1, :) = gene_part;
    sst_e(i, 1, :) = data_part(i, :);
end

for i = 1 : length(data_part(:, 1))
    for j = 1 : length(H)
        for k = 1 : length(p)
            sst_p(i, j + 1, k) = xclust{i}{j}(mclust{i}{j}(k));
            sst_e(i, j + 1, k) = fclust{i}{j}(mclust{i}{j}(k));
        end
    end
end

% visualization
scale_level = H;

sst_p1 = sst_p(1, :, :);
sst_p1 = squeeze(sst_p1);
y = (1 : length(H) + 1);

figure; plot(sst_p1, y, 'b')

axis([-150 150 1 length(H) + 10])
set(gca,'Ytick', 1 : 10 : length(H))
set(gca,'YtickLabel', {0, scale_level(10), scale_level(20), scale_level(30),
    scale_level(40), scale_level(50), scale_level(60), scale_level(70)})
xlabel('functional distance of genes')
ylabel('scale level')
title('2D scale space tree for one patient')
```

Produce_density_plot.m

```
% Scale-space tree analysis
clear
load scales_72_74_improved

%% FIRST STEP: split the whole data to two class according label
load label_74
% 'poor' patient samples
ss1 = zeros(length(find(label_74 == 0)), length(sst_p(1, :, 1)), length(sst_p
    (1, 1, :)));
% 'good' patient samples
ss2 = zeros(length(find(label_74 == 2)), length(sst_p(1, :, 1)), length(sst_p
    (1, 1, :)));
a = 1;
b = 1;

for i = 1 : length(sst_p(:, 1, 1))
    if label_74(i) == 0
        ss1(a, :, :) = sst_p(i, :, :);
        a = a + 1;
    else
        ss2(b, :, :) = sst_p(i, :, :);
        b = b + 1;
    end
end
```

```
%% SECOND STEP: measurement and comparison of the scale-space tree of patients
     in different class
% subplot for single scale
for j = 1 : 10 : 71
    ss1_2d = reshape(ss1(:, j, :), [], 1);
    ss2_2d = reshape(ss2(:, j, :), [], 1);
    if(j == 1)
        subplot(4, 2, j)
    else
        subplot(4, 2, (j - 1) / 10 + 1)
    end
    [f1, x1] = ksdensity(ss1_2d);
    plot(x1, f1, 'b')
    hold on
    [f2, x2] = ksdensity(ss2_2d);
    plot(x2, f2, 'r')

    s_legend = legend('class 1', 'class 2');
    set(s_legend, 'Fontsize', 6)
    xlabel('functional distance of genes')
    ylabel('density')
    title(sprintf('density estimates for two classes(scale-level: %d)', j))
end

% subplot for multi-scales
for j = 1 : 10 : 71
    if(j == 71)
        ss1_2d = reshape(ss1(:, j, :), [], 1);
        ss2_2d = reshape(ss2(:, j, :), [], 1);
    else
        ss1_2d = reshape(ss1(:, j : j + 9, :), [], 1);
        ss2_2d = reshape(ss2(:, j : j + 9, :), [], 1);
    end

    if(j == 1)
        subplot(4, 2, j)
    else
        subplot(4, 2, (j - 1) / 10 + 1)
    end
    [f1, x1] = ksdensity(ss1_2d);
    plot(x1, f1, 'b')
    hold on
    [f2, x2] = ksdensity(ss2_2d);
    plot(x2, f2, 'r')

    s_legend = legend('class 1', 'class 2');
    set(s_legend, 'Fontsize', 6)
    xlabel('functional distance of genes')
    ylabel('density')
    if(j == 71)
        title(sprintf('density estimates for two classes(scale-level: %d)', j)
            )
    else
        title(sprintf('density estimates for two classes(scale-level: %d-%d)',
             j, j + 9))
    end
end
```

Produce_mean_tree.m

```
% matrix transpose (from 74*72*2000 to 74*2000*72)
ss1_t = zeros(length(ss1(:, 1, 1)), length(ss1(1, 1, :)), length(ss1(1, :, 1))
    );
ss2_t = zeros(length(ss2(:, 1, 1)), length(ss2(1, 1, :)), length(ss2(1, :, 1))
    );
for i = 1 : length(ss1(:, 1, 1))
    ss1_t(i, :, :) = squeeze(ss1(i, :, :))';
    ss2_t(i, :, :) = squeeze(ss2(i, :, :))';
end

ss1_2d = ss1_t(:, :);
ss2_2d = ss2_t(:, :);

% mean-tree
[h, p, ci, stats] = ttest2(ss1_2d, ss2_2d, [], [], 'unequal');
scale_level = H;

m = mean(sst_p);
m = squeeze(m)';
y = (1 : 72);
len = length(sst_p(1, 1, :));
z = zeros(len, 72);

for i = 1 : 72
    z(:, i) = p((1 + len * (i - 1)) : (len * i))';
end

a = isnan(z);
z(a == 1) = 1;
z = log(z);

for j = 1 : len
    hold on;
    scatter3(m(j, :), y, z(j, :), 10, z(j, :), 'filled')
end

cbar = colorbar;
set(cbar,'YtickLabel',{'0.0003','0.0009','0.0025','0.0067','0.0183','0.0498','
    0.1353','0.3679','1'})
axis([-150 150 1 length(H) + 10])
set(gca,'Ytick', 1 : 10 : length(H))
set(gca,'YtickLabel', {0, scale_level(10), scale_level(20), scale_level(30),
    scale_level(40), scale_level(50), scale_level(60), scale_level(70)})
xlabel('functional distance of genes')
ylabel('scale level')
%title('Scale-space tree for mean of all patients(colored by log(p-value))')
title('Scale-space tree for mean of all patients(colored by log(p-value))-
    threshold')
```

Extract_significant_features_by_p_value_and_classification.m

```
%% THIRD STEP: find significant features in scale-space tree based on p-value
clear
load scales_72_74_improved
load p-value_72_74_position
load p-value_72_74_expression

len_scales = length(sst_p(1, :, 1));
```

```
len_samples = length(sst_p(1, 1, :));

% position data (p = 0.05)
z_p = zeros(len_samples, len_scales);
thr_p = zeros(len_samples, len_scales);

for i = 1 : len_scales
    z_p(:, i) = p((1 + len_samples * (i - 1)) : (len_samples * i))';
    for j = 1 : len_samples
        if z_p(j, i) < 0.05
            thr_p(j, i) = 1;% significant feature
        else
            thr_p(j, i) = 0;
        end
    end
end

figure; imagesc(thr_p)
xlabel('Scale-level')
ylabel('Gene')
title('Significant features chosen by t-test based on position data (2000
    genes)')

% expression-level data (p = 0.02)
z_e = zeros(len_samples, len_scales);
thr_e = zeros(len_samples, len_scales);

for i = 1 : len_scales
    z_e(:, i) = p_e((1 + len_samples * (i - 1)) : (len_samples * i))';
    for j = 1 : len_samples
        if z_e(j, i) < 0.02
            thr_e(j, i) = 1;% significant feature
        else
            thr_e(j, i) = 0;
        end
    end
end

figure; imagesc(thr_e)
xlabel('Scale-level')
ylabel('Gene')
title('Significant features chosen by t-test based on expression-level data
    (2000 genes)')

thr_p_column = sum(thr_p, 2);
thr_e_column = sum(thr_e, 2);
max_p_column = max(thr_p_column);% 25
max_e_column = max(thr_e_column);% 17

gene_p_scales = zeros(length(thr_p_column), max_p_column - 15);
gene_e_scales = zeros(length(thr_e_column), max_e_column - 7);

for i = 1 : max_p_column - 15
    gene_p_scales(thr_p_column == i + 15, i) = 1;
end

for i = 1 : max_e_column - 7
    gene_e_scales(thr_e_column == i + 7, i) = 1;
end
```

```
gene_p_row = sum(gene_p_scales, 1);
gene_e_row = sum(gene_e_scales, 1);

% matrix (sst_p or sst_e) transpose (from 74*72*2000 to 74*2000*72)
sst_p_t = zeros(length(sst_p(:, 1, 1)), length(sst_p(1, 1, :)), length(sst_p
    (1, :, 1)));
sst_e_t = zeros(length(sst_e(:, 1, 1)), length(sst_e(1, 1, :)), length(sst_e
    (1, :, 1)));

for i = 1 : length(sst_p(:, 1, 1))
    sst_p_t(i, :, :) = squeeze(sst_p(i, :, :))';
    sst_e_t(i, :, :) = squeeze(sst_e(i, :, :))';
end

% position data:

% thr_p_column = 16
thr_p_row = thr_p(gene_p_scales(:, 1) == 1, :);% 2*72
gene_p = sum(thr_p_row, 1);
f1 = sst_p_t(:, gene_p_scales(:, 1) == 1, gene_p == 2);% 74*2*16
sf1 = squeeze(f1(:, 1, :));
% do the same work on other chosen genes

% expression value:

% thr_e_column = 8
thr_e_row = thr_e(gene_e_scales(:, 1) == 1, :);% 1*72
gene_e = sum(thr_e_row, 1);
f9 = sst_e_t(:, gene_e_scales(:, 1) == 1, gene_e == 1);% 74*1*8
sf9 = squeeze(f9(:, 1, :));
% do the same work on other chosen genes

%% FOURTH STEP: classification and evaluation
load label_74
sf_p = cat(2, sf1, sf2, sf4, sf5, sf6, sf7, sf8);
sf_e = cat(2, sf9, sf10, sf11, sf12);

sst_f_p = dataset(sf_p, label_74');
sst_f_e = dataset(sf_e, label_74');

% cross-validation
cv_p_nmc = crossval(sst_f_p, nmc, 10, 5);
cv_e_nmc = crossval(sst_f_e, nmc, 10, 5);
% do the same work with other classifiers: 'ldc' 'qdc' 'knnc'
```

References

[1] S. Babaei, E. van den Akker, J. de Ridder, and MJ. Reinders, "Integrating protein family sequence similarities with gene expression to find signature gene networks in breast cancer metastasis," *PRIB*, vol. 7036, pp. 247259, 2011.