

Contents

A R code	1
A.1 Morphological operators	2
A.1.1 Naive erosion and top-hat	3
A.1.2 Line segment erosion	4
A.1.3 Naive erosion for unequally spaced values	5
A.1.4 Continuous line segment erosion	7
A.2 Spectra normalisation	9
A.2.1 Empirical quantile normalisation	10
A.2.2 Pairwise spectra MA normalisation	11
A.3 Peak alignment	12
A.3.1 Calculate W matrix for an N - and M -alignment	13
A.3.2 Dendrogram peak alignment	15
A.4 Surrogate variable analysis	19
A.4.1 Get SVA adjusted expression matrix	20
A.5 Pairwise fusion linear discriminant analysis	22
A.5.1 Create a PFDA object	23
A.5.2 Predict class for new data and a PFDA object	26
A.6 Pareto Fronts for variable ranking	27
A.6.1 Calculate dominating features	28
A.6.2 Pareto Front wrapper functions	29

Appendix A

R code

All files referenced in the current appendix are available under the directory at:

<https://github.com/tystan/thesis/tree/master/R>.

A.1 Morphological operators

Description	File	Functions
Naive erosion and top-hat:	/00_erosion_slow.R	<code>erode()</code> , <code>dilate()</code> , <code>tophat()</code>
Line segment erosion:	/01_erosion_quick.R	<code>erode_quick()</code>
Naive erosion for unequally spaced values:	/02_cts_erosion_slow.R	<code>erode_cts_slow()</code>
Continuous line segment erosion:	/03_cts_erosion_quick.R	<code>erode_cts_quick()</code>

A.1.1 Naive erosion and top-hat

Below is a simple (and naive) implementation of an erosion, dilation and top-hat operator for $x \in \{1, 2, \dots, n\} = X$ and $f(x) \in \mathbb{R} \ \forall x \in X$. Because of the assumed even spacing of the elements of X , the R-function below simply requires the vector of intensities, f , and the size of the SE.

Please note the code checks the SE size provided is an odd integer because symmetric SE is not possible with an even SE size. The maximum and minimum statements on line 11 of the code segment below, namely $\max(1, i - k0)$ and $\min(nx, i + k0)$, check for when the SE sits over the ‘edge’ on the left or right of the series, respectively. This ensures only defined f values will be used.

```

1  ### f=f are evenly spaced intensity values
2  ### k is the SE length (size of B)
3  erode<-function(f,k) #f are the intensities
4  {
5      if(!(k %% 2)) return(NULL) #SE must be of odd length
6      nx<-length(f)
7      erode<-rep(0,nx)
8      # k0 is window coverage to the left and right of centre
9      k0<-(k-1)/2
10     for(i in 1:nx) #for each m/z point across the spectrum
11         erode[i]<-min(f[max(1,i-k0):min(nx,i+k0)])
12     return(erode)
13 }
14 ### dilate is the same as erode except use max instead of min, or:
15 dilate<-function(f,k) return(-erode(-f,k))
16 ### as defined  $\tau_B(f) = f - (f \ominus B) \oplus B$ 
17 tophat<-function(f,k) return(f-dilate(erode(f,k),k))

```

A.1.2 Line segment erosion

```

1  ### f=f are evenly spaced intensity values
2  ### k is the SE length (size of B)
3  erode_quick<-function(f,k){
4      nx<-length(f)
5      t1<-proc.time()[3] ### get start time
6      if(k>=nx){
7          cat("Warning: structuring element is >= in length as the input\n")
8          cat("The input vector has been output \n")
9          return(f)
10     }else{
11         if((k%2) != 1){
12             k<-k-1
13             cat("Structring Element not symmetric, using SE length -1 =",k,"\n")
14         }
15         # k0 is window coverage to the left and right of centre
16         k0<-(k-1)/2
17         # check whether series is a length that is a multiple of k
18         # if not, add points to series for algorithm then remove at end
19         add.pix<-k-(nx%k)
20         isMultiple<-(add.pix==k)
21         if(!isMultiple){
22             f<-c(f,rep(+Inf,add.pix))
23             rem.indxs<-(nx+1):(nx+add.pix)
24             nx<-nx+add.pix
25         }
26         # intialise g,h
27         g<-rep(0,nx); h<-rep(0,nx);
28         r_min<-rep(0,nx)
29         j<-nx
30         # compute g,h values
31         for(i in 1:nx){
32             g[i]<-ifelse(i%k==1,f[i],min(g[i-1],f[i]))
33             h[j]<-ifelse(j%k==0,f[j],min(h[j+1],f[j]))
34             j<-j-1
35         }
36         # only g values are required at the left
37         r_min[1:(k0+1)]<-g[(k0+1):k]
38         # vectorised min calculations
39         r_min[(k0+2):(nx-k0-1)]<-pmin(g[(k+1):(nx-1)],h[2:(nx-k)])
40         # only h values are required at the left
41         r_min[(nx-k0):nx]<-h[(nx-k+1):(nx-k0)]
42         if(!isMultiple) r_min<-r_min[-rem.indxs]
43         delta.t<-sprintf("%.2f",proc.time()[3]-t1) ### time elapsed
44         cat("Completed morphological erosion in",delta.t,"seconds\n")
45         return(r_min)
46     }
47 }

```

A.1.3 Naive erosion for unequally spaced values

The function `get_lo_bounds()` creates a vector, `L0`, of all the lower bounds indexes such that $L0[i] = \arg \min_j x_j \geq x_i - k/2$ for each x_i , $i = 1, 2, \dots, n$. This function implements an $O(n)$ algorithm using two pointers that move along the input vector X from left to right. One pointer is the current position, the other is a lagging pointer that moves along the vector when required. To find the upper bounds, the same algorithm would be employed but the pointers start from the right and move down the vector with the second point lagging to the right.

```

1  ### get_lo_bounds() gets the index values of x[i]-k/2 for x[i], i = 1,2,...,n_x.
2  ### returns a vector of indexes length n_x
3  ### used by erode_otsu_slow()
4  get_lo_bounds<-function(x,k)
5  {
6    nx <- length(x)
7    k0 <- k/2
8    L0 <- rep(0,nx)
9    i <- 2
10   i_lo <- 1
11   L0[i_lo] <- i_lo # x[1] has lower bound 1
12   while(i<=nx)
13   {
14     if((x[i]-x[i_lo])<=k0){
15       L0[i] <- i_lo
16       i <- i+1
17     }else{
18       i_lo <- i_lo+1
19     }
20   }
21   return(L0)
22 }
23 ### same as get_lo_bounds() but for x[i]+k/2
24 get_hi_bounds<-function(x,k)
25 {
26   nx <- length(x)
27   k0 <- k/2
28   HI <- rep(0,nx)
29   i <- nx-1
30   i_hi <- nx
31   HI[i_hi] <- i_hi # x[nx] has upper bound nx
32   while(i>0)
33   {
34     if((x[i_hi]-x[i])<=k0){
35       HI[i] <- i_hi
36       i <- i-1
37     }else{
38       i_hi <- i_hi-1
39     }

```

```
40   }
41   return(HI)
42 }
43 ### x=X are unevenly (or evenly) spaced locations of the intensities
44 ### f=f are the corresponding intensity values
45 ### k is the SE length (size of B)
46 erode_cts_slow<-function(x,f,k)
47 {
48   nx<-length(x)
49   r_min<-rep(0,nx)
50   L0<-get_lo_bounds(x,k)
51   HI<-get_hi_bounds(x,k)
52   for(i in 1:nx) r_min[i]<-min(f[L0[i]:HI[i]])
53   return(r_min)
54 }
```

A.1.4 Continuous line segment erosion

```

1  ### x=X are unevenly (or evenly) spaced locations of the intensities
2  ### f=f are the corresponding intensity values
3  ### k is the SE length (size of B)
4  erode_ots_quick<-function(x,f,k){
5    nx<-length(x)
6    x_span<-x[nx]-x[1]
7    t1<-proc.time()[3]
8    isAppend<-FALSE
9    if(k>=x_span){
10     cat("Warning: structuring element spans the entire input set \n")
11     cat("The input f vector has been output \n")
12     return(f)
13   }else{
14     # check whether series is a length that is a multiple of k
15     # if not, add a single point (x1 + mk, ∞) to series for algorithm,
16     # then remove point at end
17     m<-ceiling(x_span/k)
18     mk<-m*k
19     if(!((x[1]+mk) == x[nx])){
20       x<-c(x,x[1]+mk)
21       f<-c(f,+Inf)
22       isAppend<-TRUE
23       nx<-nx+1
24       x_span<-x[nx]-x[1]
25     }
26     # create Θ
27     k_blocks<-c(0,findInterval(x,seq(x[1],x[1]+(m-1)*k,by=k)),m+1)
28     # p and q are used as current θ values running along the Θ vector
29     p<-k_blocks[1]
30     q<-k_blocks[nx+2]
31     # initialise g,h
32     g<-rep(0,nx)
33     h<-rep(0,nx)
34     r_min<-rep(0,nx)
35     i<-1
36     j<-nx
37     while(i<=nx){
38       this_p<-k_blocks[i+1]
39       this_q<-k_blocks[j+1]
40       g[i]<-ifelse(p==this_p,min(g[i-1],f[i]),f[i])
41       h[j]<-ifelse(q==this_q,min(h[j+1],f[j]),f[j])
42       p<-this_p
43       q<-this_q
44       i<-i+1
45       j<-j-1
46     }
47     k0<-k/2
48     # fast way to determine upper and lower indexes in R to avoid looping
49     lo_bounds<-nx-rev(findInterval(rev(-x),rev(-(x+k0))))+1

```

```

50     hi_bounds<-findInterval(x+k0,x)
51     # case 3
52     r_min<-pmin(h[lo_bounds],g[hi_bounds])
53     # case 1:  $\theta_{w_i^\nabla} = \theta_{w_i^\Delta} + 1$ 
54     which_lo<-which(k_blocks[lo_bounds]==k_blocks[hi_bounds+1])
55     r_min[which_lo]<-h[lo_bounds[which_lo]]
56     # case 2:  $\theta_{w_i^\nabla} + 1 = \theta_{w_i^\Delta}$ 
57     which_hi<-which(k_blocks[lo_bounds+1]==k_blocks[hi_bounds+2])
58     r_min[which_hi]<-g[hi_bounds[which_hi]]
59     if(isAppend) r_min<-r_min[-nx]
60     cat("Completed morphological erosion (cts scale) in"
61         ,sprintf("%.2f",proc.time()[3]-t1),"seconds \n")
62     return(r_min)
63 }
64 }
```

A.2 Spectra normalisation

Description	File	Functions
Empirical quantile normalisation:		
	/04_quant_norm.R	<code>quant_norm()</code>
Pairwise spectra MA normalisation:		
	/05_ma_adj.R	<code>ma_adj()</code>

A.2.1 Empirical quantile normalisation

```
1  ### Input: msData - the spectra intensities in matrix
2  ###           where columns are spectra 1,2,...,n
3  quant_norm<-function(msData)
4  {
5      msD<-msData
6      nSpec<-ncol(msD)
7      nDim<-nrow(msD)
8
9      orders<-apply(msD, 2, order)
10     reorders<-apply(orders, 2, order)
11     # order each column into ascending order
12     for(i in 1:nSpec) msD[,i]<-msD[orders[,i],i]
13     #replace ordered columns with row means
14     rmeans<-rowMeans(msD)
15     for(i in 1:nSpec) msD[,i]<-rmeans
16     #put back into the original order (with changed values)
17     for(i in 1:nSpec) msD[,i]<-msD[reorders[,i],i]
18
19     return(msD)
20 }
```

A.2.2 Pairwise spectra MA normalisation

```

1  ### m_adj(): Used by ma_adj(), performs LOESS regression on ordered MA-vals
2  ### Input: ordered dependent variable A with corresponding M
3  ### Returns: adjusted M values,  $M_t^*$ 
4  m_adj<-function(ordered_M,ordered_A)
5  {
6    MA_finites<-is.finite(ordered_M) #only include values > -∞
7    finites_M<-ordered_M[MA_finites]
8    finites_A<-ordered_A[MA_finites]
9
10   MAloess<-loess(finites_M~finites_A
11     ,span=0.40,degree=2,family="symmetric",normalize=FALSE)
12
13   finites_M<-finites_M-MAloess$fitted # make adjustments
14   ordered_M[MA_finites]<-finites_M # and return adjusted values
15   return(ordered_M)
16 }
17
18 ### ma_adj(): Perform MA adjustment on two vectors
19 ### Input: Two spectra vectors  $F_1$  and  $F_2$ 
20 ### Returns: MA adjusted  $F_1$  and  $F_2$  values,  $F_1^*$  and  $F_2^*$  respectively
21 ma_adj<-function(F1,F2)
22 {
23   t1<-proc.time()[3] ### get start time
24   V1<-log2(F1) # Will produce -∞ for log2(0)
25   V2<-log2(F2)
26   M<-V1-V2
27   A<-(V1+V2)/2
28   ### A is the dependent regression variable,
29   ### ordering required for m_adj function
30   ordered_idx<-order(A)
31   ordered_A<-A[ordered_idx]
32   ordered_M<-M[ordered_idx]
33   ordered_M<-m_adj(ordered_M,ordered_A)
34   ### get indexes of original ordering
35   orig_order<-order(ordered_idx)
36   M_dash<-ordered_M[orig_order]
37
38   orig_finites<-is.finite(M) #update values requiring updating
39   F1[orig_finites]<-2^(A[orig_finites]+M_dash[orig_finites]/2)
40   F2[orig_finites]<-2^(A[orig_finites]-M_dash[orig_finites]/2)
41
42   delta_t<-sprintf("%.2f",proc.time()[3]-t1) ### time elapsed
43   cat("Completed MA Normalisation in",delta_t,"seconds \n")
44   return(list(F1adj=F1,F2adj=F2))
45 }

```

A.3 Peak alignment

Description	File	Functions
Calculate W matrix for an N - and M -alignment:	/06_create_w.R	<code>w_matrix()</code>
Dendrogram peak alignment:	/07_dendro_peak_align.R	<code>dendro_peak_align()</code>

A.3.1 Calculate W matrix for an N - and M -alignment

```

1 #####
2 ##### Function #####
3 #####
4 #
5 # w_matrix(): Create a peak similarity matrix between an N- and M-alignment
6 #
7 #####
8 ##### Input #####
9 #####
10 #
11 # Nmatchedpeaks: K x N matrix of matched pairs of the N alignment
12 # Npeaklists: a list object containing N matrices of
13 #     (time,intensityVector) pairs: ( $n_a \times (n_{Comp}+1)$  matrix,  $a=1,\dots,N$ )
14 # Mmatchedpeaks: L x M matrix of matched pairs of the M alignment
15 # Mpeaklists: a list object containing M matrices of
16 #     (time,intensityVector) pairs: ( $n_b \times (n_{Comp}+1)$  matrix,  $b=1,\dots,M$ )
17 #
18 #####
19 #
20 # e.g. Nmatchedpeaks =
21 # [ 1 0 1 0 ]
22 # [ 0 1 2 1 ]
23 # [ 0 0 0 2 ]
24 # [ 2 2 0 0 ]
25 # [ 3 3 3 3 ]
26 # [ 0 4 0 4 ]
27 # [ . . . ]
28 # [ . . . ]
29 # [ . . . ]
30 #
31 # here K x N (N=4) matrix
32 #
33 # Npeaklist=
34 # [[1]]
35 # [t1,1 t1,2 ... t1,n1 ]
36 # [x1,1 x1,2 ... x1,n1 ]
37 #
38 # [[2]]
39 # [t2,1 t2,2 ... t2,n2 ]
40 # [x2,1 x2,2 ... x2,n2 ]
41 #
42 # ...
43 #
44 # [[N]]
45 # [tN,1 tN,2 ... tN,nN ]
46 # [xN,1 xN,2 ... xN,nN ]
47 #
48 # where ti,j is the time point j-th peak for the i-th spectrum
49 # where xi,j is the vector of intensities (nComp long)

```

```

50 # i.e.  $n_{Comp} \times 1$  matrix for the  $j$ -th peak for the  $i$ -th spectrum
51 # NB: each list item is a  $(n_{Comp} + 1) \times n_N$  matrix
52 #
53 # the  $_i$  co-ord is the row in the Nmatchedpeaks
54 # the  $_a$  co-ord is the column number of Nmatchedpeaks (the spectrum number)
55 # the  $_p$  co-ord is the peak number for the  $_a$ -th spectrum
56 #
57 # the  $_j$ ,  $_b$  and  $_q$  co-ords are defined similarly for the M-alignment
58
59 w_matrix<-function(Nmatchedpeaks,Npeaklists,Mmatchedpeaks,Mpeaklists
60 ,D,expon,lambda){
61
62     K<-nrow(Nmatchedpeaks)
63     N<-ncol(Nmatchedpeaks)
64     L<-nrow(Mmatchedpeaks)
65     M<-ncol(Mmatchedpeaks)
66     W<-matrix(0,nrow=K,ncol=L)
67
68     for(i in 1:K){
69         for(j in 1:L){
70             numerator<-0
71             denominator<-0
72             for(a in 1:N){
73                 p<-Nmatchedpeaks[i,a]
74                 if(p>0){
75                     for(b in 1:M){
76                         q<-Mmatchedpeaks[j,b]
77                         if(q>0){
78                             t_a<-Npeaklists[[a]][1,p]
79                             p_a<-Npeaklists[[a]][-1,p]
80                             t_b<-Mpeaklists[[b]][1,q]
81                             p_b<-Mpeaklists[[b]][-1,q]
82                             numerator<-numerator+
83                                 PeakSim(p_a,t_a,p_b,t_b,D,expon,lambda)
84                             denominator<-denominator+1
85                         }
86                     }
87                 }
88             }
89             if(denominator>0) W[i,j]<-numerator/denominator
90             else W[i,j]<-0
91         }
92     }
93     return(W)
94 }

```

A.3.2 Dendrogram peak alignment

```

1 #####
2 ##### Function #####
3 #####
4 #
5 # dendro_peak_align(): for peak list data, create successive N- and M-alignments
6 #           until all spectra are aligned.
7 #
8 #####
9 ##### Input #####
10 #####
11 #
12 # msD: MS Data, a  $T \times n$  matrix of MS intensities. One column per spectra.
13 # peaklistlist: see below
14 # in.param: [ D expon lambda G maxM ]-tuple as a vector
15 #
16 #####
17 ##### Output #####
18 #####
19 #
20 ### A list containing the following elements:
21 # dendro:
22 # stepwise.peaks: a list where each element is the successive amalgamation data
23 # amalpeaks: the final matrix of aligned peaks. Columns are named spectra, rows
24 #   are
25 #
26 #####
27
28 ### peaklistlist=
29 # [[1]]
30 # [t_{1,1} t_{1,2} ... t_{1,n-1} ]
31 # [x_{1,1} x_{1,2} ... x_{1,n-1} ]
32 #
33 # [[2]]
34 # [t_{2,1} t_{2,2} ... t_{2,n-2} ]
35 # [x_{2,1} x_{2,2} ... x_{2,n-2} ]
36 #
37 # .
38 # .
39 # .
40 #
41 # [[N]]
42 # [t_{N,1} t_{N,2} ... t_{N,n-N} ]
43 # [x_{N,1} x_{N,2} ... x_{N,n-N} ]
44 #
45 # where t_{i,j} is the time point j-th peak for the i-th spectrum
46 # where x_{i,j} is the vector of intensities (nComp long i.e. nComp x 1 matrix)
47 #           for the j-th peak for the i-th spectrum
48 # NB: each list item is a (nComp+1) x n_N matrix
49

```



```

50 dendro_peak_align<-function(msD,peaklistlist,in.param)
51 {
52
53   D<-in.param[1]
54   nC<-nrow(peaklistlist[[1]])-1
55   expon<-in.param[2]
56   lambda<-in.param[3]
57   G<-in.param[4]
58   maxM<-in.param[5]
59
60   nPat<-length(peaklistlist)
61   Pats<-1:nPat
62
63   cat("Calculating merge sequence for spectra \n")
64
65   fordist<-t(msD$intensity)
66   hc<-hclust(as.dist(fordist,diag=FALSE,upper=FALSE),"average")
67
68   ### find amalgamation sequence
69   ### see ?hclust for information on the merge matrix:
70   # "an n-1 by 2 matrix. Row i of merge describes the merging of clusters
71   # at step i of the clustering. If an element j in the row is negative,
72   # then observation -j was merged at this stage. If j is positive then
73   # the merge was with the cluster formed at the (earlier) stage j of the
74   # algorithm. Thus negative entries in merge indicate agglomerations of
75   # singletons, and positive entries indicate agglomerations of
76   # non-singletons."
77   amalg<-hc$merge
78
79   nAmal<-nrow(amalg)
80   # alignment of peaklistlist (a.pll)
81   a.pll<-vector(length=nAmal,mode="list")
82   Npeaks<-NULL
83   Npeaklist<-NULL
84   Mpeaks<-NULL
85   Mpeaklist<-NULL
86
87   # start
88   for(aindx in 1:nAmal)
89   {
90
91     # if patsToGetN or patsToGetM are positive,
92     # ... it is a single spectrum (1-alignment)
93     # if negative, it is a previous N/M-alignment (N,M>1)
94     patsToGetN<-amalg[aindx,1]
95     patsToGetM<-amalg[aindx,2]
96
97     printPatsN<-sprintf("%03d",patsToGetN)
98     printPatsM<-sprintf("%03d",patsToGetM)
99     amalg.str<-"Amalgamting patient"
100    if(patsToGetN>0 && patsToGetM>0){
101      cat(amalg.str,"s ",printPatsN," and ",printPatsM,"\n",sep="")

```

```

102 }else if(patsToGetN>0 && patsToGetM<0){
103   cat(amalg.str,printPatsN,"to previously amalgamated patients\n")
104 }else if(patsToGetN<0 && patsToGetM>0){
105   cat(amalg.str,printPatsM,"to previously amalgamated patients\n")
106 }else{
107   cat("Amalgamting two clusters of previously amalgamated patients\n")
108 }
109
110 ### prepare N-Alignment data
111 if(patsToGetN>0){ # if a single spectrum (1-alignment)
112   Npeaks<-matrix(1:ncol(peaklistlist[[patsToGetN]]),ncol=1)
113   Npeaklist<-peaklistlist[patsToGetN]
114 }else{ # if a previously aligned N-alignment (N>1)
115   Npeaks<-a.pll[[-patsToGetN]]
116   patsToGetN<-as.numeric(colnames(Npeaks))
117   Npeaklist<-peaklistlist[patsToGetN]
118 }
119
120 ### prepare M-Alignment data
121 if(patsToGetM>0){ # if a single spectrum (1-alignment)
122   Mpeaks<-matrix(1:ncol(peaklistlist[[patsToGetM]]),ncol=1)
123   Mpeaklist<-peaklistlist[patsToGetM]
124 }else{ # if a previously aligned M-alignment (M>1)
125   Mpeaks<-a.pll[[-patsToGetM]]
126   patsToGetM<-as.numeric(colnames(Mpeaks))
127   Mpeaklist<-peaklistlist[patsToGetM]
128 }
129
130 ### use Wmatrix() function
131 Wm<-Wmatrix(Npeaks,Npeaklist,Mpeaks,Mpeaklist,D,expon,lambda)
132 ### use S-W alignment function to estimate maximum path
133 ### see: https://code.google.com/p/swalign/
134 estPM<-SWalign(Wm,G,maxM)
135 ### estPM is a data.frame of (i,j) locations of the maximum path
136 ### the data.frame is 2 columns for i,j points
137
138 nN<-ncol(Npeaks) ### no. of peaks in N-align
139 nM<-ncol(Mpeaks) ### no. of peaks in M-align
140 nK<-nrow(estPM) ### no. of peaks in new N:M-align
141 ### apllTemp:
142 ### (a)lignment of (p)eak (l)ist (l)ist, (temp)orary
143 ### Matrix of peak indicators. The  $n_K$  rows represent the  $n_K$  peaks
144 ### from the N:M-alignment.
145 ### Entries apllTemp[i,j] are ==
146 ### { 0 if that N:M-aligned peak does not exist in spec j (column j)
147 ### { _else_ a non-zero indicator, the peak number from within the
148 ### 1-alignment from spectrum j (column j)
149 apllTemp<-matrix(0,nrow=nK,ncol=nN+nM)
150 mzValsTemp<-NULL
151 AveMzValsTemp<-NULL
152 for(n.k in 1:nK)
153 {

```

```

154     if(estPM[n.k,1]>0) # if the peak exists in the N-alignment
155     {
156         # transfer peak info from N-align to new N:M-align matrix
157         apllTemp[n.k,1:nN]<-Npeaks[estPM[n.k,1],]
158         for(i in 1:nN) if(apllTemp[n.k,i]>0) mzValsTemp<-
159             c(mzValsTemp,Npeaklist[[i]][1,apllTemp[n.k,i]])
160     }
161     if(estPM[n.k,2]>0) # if the peak exists in the M-alignment
162     {
163         # transfer peak info from N-align to new N:M-align matrix
164         apllTemp[n.k,(nN+1):(nN+nM)]<-Mpeaks[estPM[n.k,2],]
165         for(i in (nN+1):(nN+nM)) if(apllTemp[n.k,i]>0) mzValsTemp<-
166             c(mzValsTemp,Mpeaklist[[i-nN]][1,apllTemp[n.k,i]])
167     }
168     # get ave m/z of all aligned peaks
169     AveMzValsTemp<-c(AveMzValsTemp,mean(mzValsTemp))
170     mzValsTemp<-NULL
171 }
172 ### change row order if averaging m/z has changed peak location order
173 mzReOrder<-order(AveMzValsTemp)
174 apllTemp<-apllTemp[mzReOrder,]
175
176 allPat<-c(patsToGetN,patsToGetM)
177 colnames(apllTemp)<-allPat
178 ### clean up N:M-alignment to preserve spectrum order
179 patOrder<-order(allPat)
180 apllTemp<-apllTemp[,patOrder]
181
182 a.pll[[aindx]]<-apllTemp
183 }
184 ### return list() object of peak amalgamation/alignment,
185 ### including intermediate steps
186 outlist<-list(dendro=hc,stepwise.peaks=a.pll[-nAmal],amalpeaks=a.pll[[nAmal]])
187 return(outlist)
188
189 }

```

A.4 Surrogate variable analysis

Description	File	Functions
Get SVA adjusted expression matrix:	/08_do_sva.R	do_sva()

A.4.1 Get SVA adjusted expression matrix

Please note the function `getH()` (line 47 below) is the code available in the **DanteR** package to determine the number of significant surrogate variables. The function `mulReg(Y,X)` performs sequential linear regressions on the columns of the input **Y** using a fixed effects design matrix **X**. `mulReg()` returns a list containing the following vectors and matrices: **RES**_{*n*×*P*}, residual matrix after **Y** has been regressed; **BETA**_{*d*×*P*}, matrix of the regression coefficients, *P* columns for each regression; **TVALS**_{*d*×*P*}, the corresponding *t*-statistics; **PVALS**_{*d*×*P*} the corresponding *p*-values of **TVALS**; **FPVALS**_{*P*×1}, *p*-value for each linear regression corresponding to the null model *F*-statistic.

```

1 ##### FUNCTION #####
2 #### do_sva: Perform SVA using the model:
3 ####       $Y_j = \mu_j + X\alpha_j + Z\beta_j + W\delta_j + e_j$ 
4 ##### INPUTS #####
5 #### Y: is a  $n \times p$  matrix, where each p columns are regressed
6 #### Intercept: boolean; do we want to fit a mean value? (yes, in most cases)
7 #### X: is a  $n \times d_\alpha$  design matrix of the factors of interest
8 #### Z: is a  $n \times d_\beta$  design matrix of the incidental experimental factors
9 #### nosigsv: the number (referred to as H in some papers) of significant
10 ####      eigen vecs if NULL, the function will determine. If less than
11 ####      1, no W computed
12 #### verbose: boolean, whether the surrogate variable matrix, W is returned
13 #### seed: an integer to feed into 'set.seed()' for reproducible results
14 ##### OUTPUTS #####
15 #### Ytilde: the Y matrix with  $Z\beta_j + W\delta_j$  removed
16 #### pvals: the p-values of Ytilde regressed on  $\mu_j + X\alpha_j + Z\beta_j + W\delta_j$ 
17 #### tvals: the corresponding t-statistics
18 #### betas: the corresponding  $\alpha_j, \beta_j, \delta_j$  estimates
19 #### paramlabels: a combination of I (intercept), X, Z, W to signify the
20 ####      relevent rows of p-vals/t-vals/betas
21 #### W: the eigen vectors matrix
22 #### H: the number of columns of W (used eigen-vectors)
23 #####
24 do_sva<-function(
25   Y,Intercept=TRUE,X=NULL,Z=NULL,nosigsv=NULL,verbose=FALSE,seed=NULL
26 ){
27   n<-nrow(Y)
28   thisInt<-IXZ<-NULL
29   if(Intercept) thisInt<-matrix(1,nrow=n,ncol=1,dimnames=list(NULL,"Intcpt"))
30   if(is.null(thisInt) && is.null(X) && is.null(Z))
31   {
32     cat("At least one of: Intercept, X and Z must be specified \n")
33     return(NULL)
34   } else IXZ<-cbind(thisInt,X,Z)
35   kparam<-ncol(IXZ)
36   colmarkers<-rep("",kparam)
37   indx<-0
38   if(!is.null(thisInt)) colmarkers[indx<-indx+1]<-"I"

```

```

39   if(!is.null(X)) colmarkers[(indx<-indx+1):(indx<-indx+ncol(X)-1)]<- "X"
40   if(!is.null(Z)) colmarkers[(indx<-indx+1):(indx<-indx+ncol(Z)-1)]<- "Z"
41
42   RIXZ<-multReg(Y,IXZ,createNAvals=TRUE,seed=seed)
43   thissvd<-svd(RIXZ$RES)
44
45   W<-H<-NULL
46   if(is.null(nosigsv)){
47     H<-getH(RIXZ$RES,IXZ,nullsig=0.1,verbose=FALSE)
48     if(H<1) cat("No significant surrogate variables found \n")
49   }else{
50     H<-nosigsv
51   }
52   if(H<1){
53     cat("No surrogate variables will be used \n")
54   }else{
55     cat("Using H=",H," significant surrogate variables \n",sep="")
56     W<-as.matrix(thissvd$u[,1:H])
57     colnames(W)<-paste("W",1:H,sep="")
58     colmarkers<-c(colmarkers,rep("W",H))
59   }
60   IXZW<-cbind(IXZ,W)
61   Rtilde<-multReg(Y,IXZW)
62   removecols<-colmarkers %in% c("Z","W")
63   ZBetaWDelta<-0
64   if(sum(removecols)) ZBetaWDelta<-as.matrix(IXZW[,removecols]) %*%
65     as.matrix(Rtilde$BETA[removecols,])
66   Ytilde<-Y-ZBetaWDelta
67   if(verbose) return(list(Ytilde=Ytilde,paramlabels=colmarkers,W=W,H=H))
68   else return(Ytilde)
69 }

```

A.5 Pairwise fusion linear discriminant analysis

Description	File	Functions
Create a PFDA object:	/09_create_pfda_obj.R	<code>create_pfda_obj()</code>
Predict class for new data and a PFDA object:	/10_pfda_predict.R	<code>pfda_predict()</code>

A.5.1 Create a PFDA object

```

1 ##### FUNCTION: create_pfda_obj()
2 ### estimate parameters of PF-DA model, so that a discrim function created
3
4 ##### input:
5 ### X: a n x p matrix, of n obs and p variables
6 ### Xclass: a vector of length n of the classes (must be a factor variable)
7 ### priors: a vector of length K (#classes) with elements in (0,1)
8
9 create_pfda_obj<-function(X,Xclass,lambdar=1,priors=NULL,alpha=NULL,wt=NULL)
10 {
11   N<-length(Xclass)
12   P<-ncol(X)
13   nks<-table(Xclass)
14   classnames<-levels(Xclass)
15   K<-length(classnames)
16
17   ### if not supplied, make  $\hat{\pi}_k$  data proportions
18   if(is.null(priors)) priors<-nks/N
19
20   if(length(priors)!=K){
21     cat("The length of priors and the total number
22         of groups must be equal \n")
23     return(NULL)
24   }else if(is.null(alpha) & (N<P)){
25     cat("Alpha is suggested for n<p data \n")
26   }else if(N!=nrow(X)){
27     cat("The length of Xclass and the number
28         of rows in X must agree \n")
29     return(NULL)
30   }else if(!all(nks>1)){
31     cat("There needs to be at least two obs in each
32         group for variances to be computed \n")
33     return(NULL)
34   }
35
36   Xclassint<-as.integer(Xclass)
37   transMeans<-colMeans(X)
38   X<-X-matrix(rep(transMeans,N),nrow=N,byrow=TRUE)
39
40   ##### create  $\mu_k = [\mu_{k1}, \dots, \mu_{kp}]$  vectors
41   ##### place on top of each other to get KxP matrix
42   MuMat<-matrix(0,nrow=K,ncol=P)
43   for(k in 1:K) MuMat[k,]<-colMeans(X[Xclassint==k,])
44   MuIter<-MuMat
45
46   ##### create  $\Sigma$ 
47   Sigma<-matrix(0,nrow=P,ncol=P)
48   for(k in 1:K)
49   {

```



```

50     rowuse<-which(Xclassint==k)
51     Sigma<-Sigma+length(rowuse)*cov.wt(X[rowuse,],cor=FALSE
52                                         ,center=TRUE,method="ML")$cov
53 }
54 Sigma<-Sigma/N
55 ##### extract Diag elements
56 sigmasqs<-diag(Sigma)
57
58 if(!(is.null(alpha) | is.null(wts))) sigmasqs<-sigmasqs+alph*wts
59 else if(!is.null(alpha)) sigmasqs<-sigmasqs+rep(alpha,P)
60
61 ##### Now start iterative estimation of the  $\ell_1$  penalised means
62 ##### Note "sqig" is used for the ML estimates
63 G<-matrix(0,nrow=K,ncol=K)
64 deltatal<-1e-10
65 deltaMu<-1
66 maxIter<-500
67 itcount<-0
68 while(deltaMu>(1e-5) && itcount<maxIter)
69 {
70     deltaMuNumer<-0
71     deltaMuDenom<-0
72     itcount<-itcount+1
73
74     ### For each of the features
75     for(j in 1:P)
76     {
77         beta.t.j<-MuIter[,j]
78         musqig.j<-MuMat[,j]
79         sqigY<-X[,j]
80         sqigX<-matrix(0,nrow=N,ncol=K)
81         G<-matrix(0,nrow=K,ncol=K)
82
83         ###  $\sum_{k=1}^{K-1} \sum_{k_{dash}=k+1}^K$ 
84         for(k in 1:(K-1))
85         {
86             for(kdash in (k+1):K)
87             {
88                 PFweight<-1/abs(musqig.j[k]-musqig.j[kdash])
89                 ### assign updated iterations, or tol value if "zero"
90                 muDiffIter<-max(abs(beta.t.j[k]-beta.t.j[kdash]),deltatal)
91                 G[k,kdash]<-G[kdash,k]<- -PFweight/muDiffIter
92             }
93         }
94         for(k in 1:K)
95         {
96             sqigX[which(Xclassint==k),k]<-1
97             ### note the diag elements of G can be calculated as the sum of the column
98             G[k,k]<- -sum(G[,k])
99         }
100         #####  $\hat{M} = (B^T B + \lambda \sigma_j^2 G)^{-1} B^T J$ 
101         MuIter[,j]<-solve(t(sqigX)%*%sqigX+lambdar*sigmasqs[j]*G)%*%(t(sqigX)%*%sqigY)

```

```

102     deltaMuNumer<-deltaMuNumer+sum(abs(MuIter[,j]-beta.t.j))
103     deltaMuDenom<-deltaMuDenom+sum(abs(beta.t.j))
104   }
105   ### our break loop value
106   deltaMu<-deltaMuNumer/deltaMuDenom
107
108 }
109 cat("Iterations performed to aquire a solution:",itcount
110     , "| Final tol val:",deltaMu," \n")
111
112 MuIter[which(MuIter<deltatol)]<-0
113
114 ###  $\frac{1}{2} \sum_{j=1}^p \hat{\mu}_{k,j}^2$  constant term
115 consts<-rep(0,K)
116 for(k in 1:K) consts[k]<-0.5*sum(MuIter[k,]^2/sigmasqs)
117 ###  $\hat{\mu}_{k,j}^2/\sigma_p^2$  term
118 lins<-vector(mode="list",length=K)
119 for(k in 1:K) lins[[k]]<-MuIter[k,]/sigmasqs
120
121 ### return calculated information as list object
122 return(list(classes=classnames,consts=consts,lins=lins,prior=priors
123            ,meanadj=transMeans,MuIter=MuIter,InitMu=MuMat))
124 }

```

A.5.2 Predict class for new data and a PFDA object

```

1 ##### FUNCTION: pfda_predict()
2 ### estimate probabilities and class of new inputs
3
4 ##### input:
5 ### ldaobj: an object created by create_pfda_obj()
6 ### Xnew: a n x p matrix, of n obs and p variables.
7 ###      May also be a single numeric vector (n=1) of length p
8 ### priors: a vector of length K (#classes) with elements in (0,1)
9
10 pfda_predict<-function(ldaobj,Xnew,priors=NULL)
11 {
12   if(is.vector(Xnew,mode="numeric")){
13     Xnew<-matrix(Xnew,nrow=1)
14   }else if(!is.matrix(Xnew)){
15     cat("Xnew must be numeric and either a vector or matrix \n ")
16   }
17
18   Nnew<-nrow(Xnew)
19   ### centre each feature at 0, as done on the training data
20   Xnew<-Xnew-matrix(rep(ldaobj$meanadj,Nnew),nrow=Nnew,byrow=TRUE)
21   Xclasses<-ldaobj$classes
22   K<-length(Xclasses)
23   if(is.null(priors)) priors<-ldaobj$prior
24
25   outposteriors<-matrix(0,nrow=Nnew,ncol=K)
26   colnames(outposteriors)<-Xclasses
27
28   ### discrim function:  $\delta_k(x_{new})$ 
29   for(i in 1:Nnew){
30     for(k in 1:K){
31       outposteriors[i,k]<-log(priors[k]) - ldaobj$consts[k]
32                                     + sum(Xnew[i,]*ldaobj$lins[[k]])
33     }
34   }
35
36   ### predicted class is arg max
37   ###  $p(G_K|x_{new}) = \frac{P(x_{new}|G_k)P(G_k)}{\sum_{i=1}^K P(x_{new}|G_i)P(G_i)}$ 
38   predclasses<-rep(0,Nnew)
39   for(i in 1:Nnew){
40     outposteriors[i,<-exp(outposteriors[i,])
41     predclasses[i]<-which.max(outposteriors[i,])
42     outposteriors[i,<-outposteriors[i,]/sum(outposteriors[i,])
43   }
44   outpred<-factor(Xclasses[predclasses])
45
46   return(list(pred=outpred,posteriors=outposteriors))
47 }

```

A.6 Pareto Fronts for variable ranking

Description	File	Functions
Calculate dominating features:	/11_dom_feat.c	<code>dom_feat()</code>
Pareto Front wrapper functions:	/12_pareto_fronts.R	<code>pareto_ranking()</code>

A.6.1 Calculate dominating features

Below is the core of the Pareto Front code, finding features that are the dominated as per the definition. Written in C to be compiled to a .so file (or .dll on Windows operating systems) that in turn can be loaded into R.

```
1  #include <R.h>
2
3  void dom_feat(int *n, double *obja, double *objb, int *domvec)
4  {
5      int i, j, nonDomI;
6      for (i=0; i<*n; i++)
7      {
8          j=0;
9          nonDomI=1;
10         while(nonDomI && j<*n)
11         {
12             if((obja[i]<obja[j]) && (objb[i]<=objb[j])) nonDomI=0;
13             else if((objb[i]<objb[j]) && (obja[i]<=obja[j])) nonDomI=0;
14             j+=1;
15         }
16         domvec[i]=-nonDomI+1;
17     }
18 }
```

A.6.2 Pareto Front wrapper functions

```

1  # require animation package for kfcv() function
2  library(animation)
3  # load compiled C code (shared object)
4  dyn.load("dom_feat.so")
5
6  #####
7  #####
8  #
9  # Pairwise case of Pareto Fronts
10 #   obj is the 2 × n matrix of the two vectors of length n for
11 #   the features/observations of the 2 criteria/objective functions
12 #   istomin is a boolean vector of whether the criteria obj1, obj2
13 #   are to be minimised (=TRUE), respectively
14 #
15 #####
16 #####
17
18 #
19 # This function returns a vector of 'dominated' observations (Boolean,
20 # length n vector) FALSE=Pareto front, TRUE=dominated observation
21 #
22
23 dom_features_pw<-function(obj,istomin)
24 {
25   #if to be minimised then just make negative and maximise
26   obj[,istomin]<- -obj[,istomin]
27   n<-as.integer(nrow(obj))
28   obj1<-as.double(obj[,1])
29   obj2<-as.double(obj[,2])
30   domvec<-as.integer(rep(0,n)) #output vector
31
32   return(as.logical(.C("dom_feat",n,obj1,obj2,domvec)[[4]]))
33 }
34
35 #####
36 #####
37 #
38 # General case
39 #   objmatrix is n × m matrix. n features/observations and
40 #   m criteria/objective functions istominvec is a boolean vec
41 #   of length m to say whether the criteria are to be minimised
42 #
43 #####
44 #####
45
46 #
47 # This function returns a vector of 'dominated' observations (Boolean,
48 # length n vector) FALSE=Pareto front, TRUE=dominated observation
49 # same as pairwise but the input can take more than two objective functions

```

```

50 #
51
52 dom_features<-function(objmatrix,istominvec)
53 {
54   n<-as.integer(nrow(objmatrix))
55   m<-ncol(objmatrix)
56   objmatrix[,istominvec]<- -objmatrix[,istominvec]
57   vecdomvec<-rep(1,n)
58   indxs<-combn(m,2)
59   nm<-ncol(indxs)
60   i<-0
61   while(i<nm)
62   {
63     i<-i+1
64     # call pairwise function, take the intersection of previous
65     # dominated observations remembering the intersection(s)
66     # of dominated in the same as unions(s) of Pareto fronts
67     vecdomvec<-vecdomvec * .C("dom_feat"
68                               ,n
69                               ,as.double(objmatrix[,indxs[1,i]])
70                               ,as.double(objmatrix[,indxs[2,i]])
71                               ,as.integer(rep(0,n)))[[4]]
72   }
73   return(as.logical(vecdomvec))
74 }
75
76 #####
77 #####
78 #
79 # Successive Pareto Fronts
80 #   noFronts is the # of pareto fronts required
81 #
82 #   fn returns a vector of length n
83 #       each element is labelled the pareto front #,
84 #       0 is dominated even after noFronts found
85 #
86 #####
87 #####
88
89 pareto_fronts<-function(noFronts,objmatrix,istominvec)
90 {
91   objmatrix[,istominvec]<- -objmatrix[,istominvec]
92   n<-as.integer(nrow(objmatrix))
93   m<-ncol(objmatrix)
94   pfvec<-rep(0,n) #output vector
95   #once a front is found we need to set the corresponding values to  $\infty$  or
96   #  $-\infty$  so they won't be chosen again
97   #try: as.numeric(c(TRUE,FALSE,TRUE))*2-1 to see what the next line is doing
98   #if Min then set 1, ifMax then set -1 (the sign of the Inf if we find front
99   # and have to put to a value)
100   ourInfs<-min(objmatrix)-1
101   allFrontsFound<-FALSE

```

```

102
103   i<-0
104   while(i<noFronts && !allFrontsFound) #go thru all fronts required
105   {
106     i<-i+1
107     df<-dom_features(objmatrix,rep(FALSE,m)) #general m obj vectors function
108     # pf.i are the indexes of the output vector that need to be updated
109     # with the pareto front number
110     pf.i<-(!df) & (pfvec<1)
111     pfvec[pf.i]<-i
112     # re-assign values were pareto front found
113     objmatrix[pf.i,]<-ourInfs
114     if(all(pfvec>0)) allFrontsFound<-TRUE
115   }
116   return(pfvec)
117 }
118
119 #####
120 #####
121 #
122 # Leave-one-out/k-fold feature ranking
123 #
124 # returns a vector of length n with values  $\in (0,1]$  for feature importance
125 #
126 #####
127 #####
128
129 #
130 # Same inputs of previous functions, with folds (aka k-fold cross
131 # validation) and reps is the number of times we re-do the cross
132 # validation fold=1 or the length of the input (i.e. n) creates
133 # leave-one-out cross validation
134
135 pareto_ranking<-function(objmatrix,istominvec,noFronts=20,folds=1,reps=5)
136 {
137   objmatrix[,istominvec]<- -objmatrix[,istominvec]
138   m<-ncol(objmatrix)
139   n<-nrow(objmatrix)
140   pfmetric<-rep(0,n) #output vector
141   nfolds<-n
142   if(folds>1) nfolds<-folds
143   if(nfolds==n) reps<-1
144
145   blocks<-kfcv(nfolds,n)
146   block.nos<-rep(1:nfolds,blocks)
147
148   for(r in 1:reps)
149   {
150     indxs<-sample(1:n) #fresh randomisation each repetition
151     k.f.mat<-cbind(indxs,block.nos) # create the fold 'blocks' of data
152     for(i in 1:nfolds)
153     {

```



```

154     rows<-k.f.mat[k.f.mat[,2]==i,1] # find the ith fold to leave out
155     # call general function with ith fold removed
156     calcfronts<-pareto_fronts(noFronts,objmatrix[-rows,],rep(FALSE,m))
157     # which are non-dominated
158     whichnondom<-calcfronts>0
159     # if you are on the first front you get 1, second=1/2, third=1/3,
160     # ..., jth=1/j, else 0
161     pfmetric[-rows][whichnondom]<-
162         pfmetric[-rows][whichnondom]+1/calcfronts[whichnondom]
163     }
164 }
165 #now divide by maximum possible value i.e. (nfolds-1)*reps so output in [0,1]
166 pfmetric<-pfmetric/((nfolds-1)*reps)
167 return(pfmetric)
168 }
169
170 #####
171 # Below are three metrics that can possibly describe the value of
172 # variables/features to discriminate between classes
173 #####
174
175 # minIntraClassVar(): find the minimum WITHIN class variance of the K groups
176 # interClassVar(): find the variance of means/centroids of the K groups
177 # maxInterClassDist(): possibly correlated with interClassVar, find the dist
178 # MAXIMUM between the K group's centroids/means
179
180 # The rationale of the last metric is that a variable/feature that only
181 # separates two of the K classes is undervalued by the Fisher score because
182 # it may not separate the K-2 classes remaining well.
183 # ... And a separation of two classes (in conjunction with other variables)
184 # is important information for the discriminant model
185
186 #####
187 #### ds: a data.frame or matrix (numeric values only/factors will be dealt
188 #### with as integers) class vec: a vector corresponding to the class of
189 #### the rows of ds
190 #####
191
192 minIntraClassVar<-function(ds,class.vec){
193
194     dsfs<-ds
195     if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)
196
197     p<-ncol(dsfs)
198     K<-length(levels(class.vec))
199     n.all<-length(class.vec)
200     n.i<-0
201
202     intraClassVar<-Inf
203     mean.j<-colMeans(dsfs)
204
205     for(i in 1:K){

```

```

206     true.vec<- (as.integer(class.vec)==i)
207     n.i<-length(which(true.vec))
208     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
209     var.class<-colSums(as.matrix((dsfs[true.vec,]-rep(mean.class,each=n.i))^2,ncol=p))
210     intraClassVar<-pmin(intraClassVar,var.class/(n.i-1))
211   }
212   return(intraClassVar)
213 }
214
215 interClassVar<-function(ds,class.vec){
216
217   dsfs<-ds
218   if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)
219
220   p<-ncol(dsfs)
221   K<-length(levels(class.vec))
222
223   interClassVar<-0
224   mean.j<-colMeans(dsfs)
225
226   for(i in 1:K){
227     true.vec<- (as.integer(class.vec)==i)
228     n.i<-length(which(true.vec))
229     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
230     var.class<-((mean.j-mean.class)^2)
231     interClassVar<-interClassVar+var.class
232   }
233   return(interClassVar/(K-1))
234 }
235
236 maxInterClassDist<-function(ds,class.vec){
237
238   dsfs<-ds
239   if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)
240
241   p<-ncol(dsfs)
242   K<-length(levels(class.vec))
243
244   interClassDist<-Inf
245   mean.j<-colMeans(dsfs)
246
247   for(i in 1:K){
248     true.vec<- (as.integer(class.vec)==i)
249     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
250     interClassDist<-pmax(interClassDist,abs(mean.j-mean.class))
251   }
252   return(interClassDist)
253 }

```