

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	27
A.6.2	Pareto Front wrapper functions	28

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

Consider $x \in \{1, 2, \dots, n\} = X$ and $f(x) \in \mathbb{R} \quad \forall x \in X$. 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.

```

1 erode<-function(f,sesize) #f are the intensities
2 {
3   if(!(sesize %% 2)) return(NULL) #SE must be of odd length
4   nx<-length(f); erode<-rep(0,nx); halfse<-(sesize-1)/2;
5   for(i in 1:nx) #for each m/z point across the spectrum
6     erode[i]<-min(f[max(1,i-halfse):min(nx,i+halfse)])
7   return(erode)
8 }
9 ### dilate is the same as erode except use max instead of min ...OR...
10 dilate<-function(f,sesize) return(-erode(-f,sesize))
11 ### as defined  $\tau_B(f) = f - (f \ominus B) \oplus B$ 
12 tophat<-function(f,sesize) return(f-dilate(erode(f,sesize),sesize))

```

Note, the code checks the SE provided is an odd integer as a centred, symmetric SE is required. The maximum and minimum statements on line 6 of the code segment above, namely $\max(1, i - \text{halfse})$ and $\min(\text{nx}, i + \text{halfse})$, 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.

A.1.2 Line segment erosion

```

1  erode.quick<-function(f,se.size){
2    nx<-length(f)
3    k<-se.size
4    t1<-proc.time()[3] ### get start time
5    if(k>=nx){
6      cat("Warning: structuring element is >= in length as the input\n")
7      cat("The input vector has been output \n")
8      return(f)
9    }else{
10     if((k%2) != 1){
11       k<-k-1
12       cat("Structring Element not symmetric, using SE length -1 =",k,"\n")
13     }
14     k.left<-(k-1)/2
15     add.pix<-k-(nx%k)
16     isMultiple<-(add.pix==k)
17     if(!isMultiple){
18       f<-c(f,rep(+Inf,add.pix))
19       rem.indxs<-(nx+1):(nx+add.pix)
20       nx<-nx+add.pix
21     }
22     g<-rep(0,nx)
23     h<-rep(0,nx)
24     r<-rep(0,nx)
25     j<-nx
26     for(i in 1:nx){
27       if(i%k==1){
28         g[i]<-f[i]
29       }else{
30         g[i]<-min(g[i-1],f[i])
31       }
32       if(j%k==0){
33         h[j]<-f[j]
34       }else{
35         h[j]<-min(h[j+1],f[j])
36       }
37       j<-j-1
38     }
39     r[1:(k.left+1)]<-g[(k.left+1):k]
40     r[(k.left+2):(nx-k.left-1)]<-pmin(g[(k+1):(nx-1)],h[2:(nx-k)])
41     r[(nx-k.left):nx]<-h[(nx-k+1):(nx-k.left)]
42     if(!isMultiple) r<-r[-rem.indxs]
43
44     delta.t<-sprintf("%.2f",proc.time()[3]-t1) ### time elapsed
45     cat("Completed morphological erosion in",delta.t,"seconds\n")
46     return(r)
47   }
48 }

```

A.1.3 Naive erosion for unequally spaced values

A vector of all the lower bounds (LB) x_l in Algorithm ?? for each corresponding x_i can be created using a $O(n)$ algorithm using two pointers along the input vector X . One pointer is the current position, the other is a lagging pointer that moves along the vector when required. A simple algorithm to do this is described in Algorithm ?. To find the upper bound the same algorithm would be employed, but the pointers will start from the right and move down the vector with the second point lagging to the right.

```

1  get.lo.bounds<-function(x,k){
2    nx <- length(x)
3    k0 <- k/2
4    L0 <- rep(0,nx)
5    i <- 2
6    i_lo <- 1
7    L0[i_lo] <- i_lo # x[1] has lower bound 1
8    while(i<=nx)
9    {
10     if((x[i]-x[i_lo])<=k0){
11       L0[i] <- i_lo
12       i <- i+1
13     }else{
14       i_lo <- i_lo+1
15     }
16   }
17   return(L0)
18 }
19 get.hi.bounds<-function(x,k){
20   nx <- length(x)
21   k0 <- k/2
22   HI <- rep(0,nx)
23   i <- nx-1
24   i_hi <- nx
25   HI[i_hi] <- i_hi # x[nx] has upper bound nx
26   while(i>0)
27   {
28     if((x[i_hi]-x[i])<=k0){
29       HI[i] <- i_hi
30       i <- i-1
31     }else{
32       i_hi <- i_hi-1
33     }
34   }
35   return(HI)
36 }
37 erode.cts.slow<-function(x,f,k){
38   nx<-length(x)

```

```
39   r_min<-rep(0,nx)
40   L0<-get.lo.bounds(x,k)
41   HI<-get.hi.bounds(x,k)
42   for(i in 1:nx) r_min[i]<-min(f[L0[i]:HI[i]])
43   return(r_min)
44 }
```

A.1.4 Continuous line segment erosion

```

1  erode.cts.quick<-function(x,f,se.span){
2
3    out.vector<-NULL
4    nx<-length(x)
5    x.span<-x[nx]-x[1]
6    k<-se.span
7    t1<-proc.time()[3]
8    isAppend<-FALSE
9
10   if(k>=x.span){
11     cat("Warning: structuring element spans the entire input set \n")
12     cat("The input f vector has been output \n")
13     return(f)
14   }else{
15     m<-ceiling(x.span/k)
16     mk<-m*k
17     if(!((x[1]+mk) == x[nx])){
18       x<-c(x,x[1]+mk)
19       f<-c(f,+Inf)
20       isAppend<-TRUE
21       nx<-nx+1
22       x.span<-x[nx]-x[1]
23     }
24     k.blocks<-c(0,findInterval(x,seq(x[1],x[1]+(m-1)*k,by=k)),m+1)
25     p<-k.blocks[1]
26     q<-k.blocks[nx+2]
27     g<-rep(0,nx)
28     h<-rep(0,nx)
29     r<-rep(0,nx)
30     i<-1
31     j<-nx
32     while(i<=nx){
33       this.p<-k.blocks[i+1]
34       this.q<-k.blocks[j+1]
35       if(p==this.p){
36         g[i]<-min(g[i-1],f[i])
37       }else{
38         g[i]<-f[i]
39       }
40       if(q==this.q){
41         h[j]<-min(h[j+1],f[j])
42       }else{
43         h[j]<-f[j]
44       }
45       p<-this.p
46       q<-this.q
47       i<-i+1
48       j<-j-1
49     }

```



```
50     k.left<-k/2
51     low.bound.index<-nx-rev(findInterval(rev(-x),rev(-(x+k.left))))+1
52     upp.bound.index<-findInterval(x+k.left,x)
53     out.vector<-pmin(h[low.bound.index],g[upp.bound.index])
54     which.low<-which(k.blocks[low.bound.index]==k.blocks[upp.bound.index+1])
55     out.vector[which.low]<-h[low.bound.index[which.low]]
56     which.hi<-which(k.blocks[low.bound.index+1]==k.blocks[upp.bound.index+2])
57     out.vector[which.hi]<-g[upp.bound.index[which.hi]]
58     if(isAppend) out.vector<-out.vector[-nx]
59     cat("Completed morphological erosion (cts scale) in"
60         ,sprintf("%.2f",proc.time()[3]-t1),"seconds \n")
61     return(out.vector)
62 }
63 }
```

A.2 Spectra normalisation

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

A.2.1 Empirical quantile normalisation

```
1  ### Input: msData - the spectra intensities in matrix
2  ###      where columns are spectra 1,2,...,n
3  msEmpiricalQuantNorm<-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  ### adjM(): Used by intensAdj(), performs LOESS regression on ordered MA-vals
2  ### Input: ordered dependent variable A with corresponding M
3  ### Returns: adjusted M values,  $M_t^*$ 
4  adjM<-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   MAlloess<-loess(finites.M~finites.A
11     ,span=0.40,degree=2,family="symmetric",normalize=FALSE)
12
13   finites.M<-finites.M-MAlloess$fitted # make adjustments
14   ordered.M[MA.finites]<-finites.M # and return adjusted values
15   return(ordered.M)
16 }
17
18 ### intensAdj(): 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 intensAdj<-function(F1,F2)
22 {
23   t1<-proc.time()[3] ### get start time
24   V1<-log2(F1) # Will produce -∞ for  $\log_2(0)$ 
25   V2<-log2(F2)
26   M<-V1-V2
27   A<-(V1+V2)/2
28   ### A is the dependent regression variable,
29   ### ordering required for adjM function
30   ordered.indxs<-order(A)
31   ordered.A<-A[ordered.indxs]
32   ordered.M<-M[ordered.indxs]
33   ordered.M<-adjM(ordered.M,ordered.A)
34   ### get indexes of original ordering
35   orig.order<-order(ordered.indxs)
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>Wmatrix()</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 # Wmatrix(): 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 Wmatrix<-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 # guideTreePeakAlign(): 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 guideTreePeakAlign<-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	doSVA()

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: `RESn×P`, residual matrix after `Y` has been regressed; `BETAd×P`, matrix of the regression coefficients, `P` columns for each regression; `TVALSd×P`, the corresponding *t*-statistics; `PVALSd×P` the corresponding *p*-values of `TVALS`; `FPVALSP×1`, *p*-value for each linear regression corresponding to the null model *F*-statistic.

```

1 ##### FUNCTION #####
2 #### doSVA: 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/ $\beta$ -vals
21 #### W: the eigen vectors matrix
22 #### H: the number of columns of  $W$  (used eigen-vectors)
23 #####
24 doSVA<-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: createPFldaobj()
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 createPFldaobj<-function(X,Xclass,lambdar=1,priors=NULL,alph=NULL,wt=TRUE)
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(alph) & (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: PFldapredict()
2  ### estimate probabilities and class of new inputs
3
4  ##### input:
5  ### ldaobj: an object created by createPFlldaobj()
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 PFldapredict<-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>domfeat()</code>
Pareto Front wrapper functions:		
	/12_pareto_fronts.R	<code>paretoRanking()</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 domfeat(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  library(animation)
2  dyn.load("domfeat.so")
3
4  #####
5  #####
6  #
7  # Pairwise case of Pareto Fronts
8  #   obj is the 2 × n matrix of the two vectors of length n for
9  #   the features/observations of the 2 criteria/objective functions
10 #   istomin is a boolean vector of whether the criteria obj1, obj2
11 #   are to be minimised (=TRUE), respectively
12 #
13 #####
14 #####
15
16 #
17 # This function returns a vector of 'dominated' observations (Boolean,
18 # length n vector) FALSE=Pareto front, TRUE=dominated observation
19 #
20
21 domFeaturesPW<-function(obj,istomin)
22 {
23   #if to be minimised then just make negative and maximise
24   obj[,istomin]<- -obj[,istomin]
25   n<-as.integer(nrow(obj))
26   obj1<-as.double(obj[,1])
27   obj2<-as.double(obj[,2])
28   domvec<-as.integer(rep(0,n)) #output vector
29
30   return(as.logical(.C("domfeat",n,obj1,obj2,domvec)[[4]]))
31 }
32
33 #####
34 #####
35 #
36 # General case
37 #   objmatrix is n × m matrix. n features/observations and
38 #   m criteria/objective functions istominvec is a boolean vec
39 #   of length m to say whether the criteria are to be minimised
40 #
41 #####
42 #####
43
44 #
45 # This function returns a vector of 'dominated' observations (Boolean,
46 # length n vector) FALSE=Pareto front, TRUE=dominated observation
47 # same as pairwise but the input can take more than two objective functions
48 #
49

```

```

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

```

```

102 while(i<noFronts && !allFrontsFound) #go thru all fronts required
103 {
104   i<-i+1
105   df<-domFeatures(objmatrix,rep(FALSE,m)) #general m obj vectors function
106   # pf.i are the indexes of the output vector that need to be updated
107   # with the pareto front number
108   pf.i<-(!df) & (pfvec<1)
109   pfvec[pf.i]<-i
110   # re-assign values were pareto front found
111   objmatrix[pf.i,]<-ourInfs
112   if(all(pfvec>0)) allFrontsFound<-TRUE
113 }
114 return(pfvec)
115 }
116
117 #####
118 #####
119 #
120 # Leave-one-out/k-fold feature ranking
121 #
122 # returns a vector of length n with values  $\in (0,1]$  for feature importance
123 #
124 #####
125 #####
126
127 #
128 # Same inputs of previous functions, with folds (aka k-fold cross
129 # validation) and reps is the number of times we re-do the cross
130 # validation fold=1 or the length of the input (i.e. n) creates
131 # leave-one-out cross validation
132
133 paretoRanking<-function(objmatrix,istominvec,noFronts=20,folds=1,reps=5)
134 {
135   objmatrix[,istominvec]<- -objmatrix[,istominvec]
136   m<-ncol(objmatrix)
137   n<-nrow(objmatrix)
138   pfmetric<-rep(0,n) #output vector
139   nfolds<-n
140   if(folds>1) nfolds<-folds
141   if(nfolds==n) reps<-1
142
143   blocks<-kfcv(nfolds,n)
144   block.nos<-rep(1:nfolds,blocks)
145
146   for(r in 1:reps)
147   {
148     indxs<-sample(1:n) #fresh randomisation each repetition
149     k.f.mat<-cbind(indxs,block.nos) # create the fold 'blocks' of data
150     for(i in 1:nfolds)
151     {
152       rows<-k.f.mat[k.f.mat[,2]==i,1] # find the ith fold to leave out
153       # call general function with ith fold removed

```



```

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

```

```

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

```