MKL Classification

For our benchmark example, we have two groups which are drawn from a bivariate normal distribution where the mean of one groupis fixed and the group means shift to provide different amounts of overlap of the two groups. This example is meant to illustrate that each MKL implentation selects the correct kernel. For instance, as the hyperparameter gets smaller (using kernlab's parameterization) the resulting decision boundary is almost linear. On the other hand, as the hyperparameter gets larger then decision boundary becomes more jagged and circular. Further details of this are highlighted "Multiple-kernel learning for genomic data mining and prediction" on bioRvix doi: https://doi.org/10.1101/415950.

Loading data

```
library(RMKL)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
data(benchmark.data)
# The data sets are organized in a a list. Each entry of the list is a 100x3 matrix with each row consi
#Below is a summary of the mean of each group for each mean structure.
lapply(1:length(benchmark.data), function(a) aggregate(x = benchmark.data[[a]][,1:2], by=list(benchmark
```

```
## [[1]]
##   Group.1       V1        V2
## 1      -1 1.011548 0.9763316
## 2       1 5.127577 4.9566653
##
## [[2]]
##   Group.1       V1       V2
## 1      -1 2.182839 1.862098
## 2       1 5.073472 4.882758
##
## [[3]]
##   Group.1       V1       V2
## 1      -1 3.003946 3.048661
## 2       1 5.064277 5.036488
##
## [[4]]
##   Group.1       V1       V2
## 1      -1 3.972222 3.983406
## 2       1 4.910868 5.217608
##
## [[5]]
##   Group.1       V1       V2
## 1      -1 4.993349 5.091566
## 2       1 5.042654 5.095419
```

```
##
## [[6]]
##   Group.1       V1       V2
## 1      -1 5.914642 5.907660
## 2       1 5.062362 4.906792
##
## [[7]]
##   Group.1       V1       V2
## 1      -1 6.975286 7.072817
## 2       1 4.963323 4.906484
##
## [[8]]
##   Group.1       V1       V2
## 1      -1 8.069796 8.031453
## 2       1 4.929443 4.862344
##
## [[9]]
##   Group.1       V1       V2
## 1      -1 8.905826 9.157771
## 2       1 4.865411 5.207769
```

Using RMKL with benchmark data

```
data.mkl=benchmark.data[[4]]
kernels=rep('radial',2)
sigma=c(2,1/20)
train.samples=sample(1:nrow(data.mkl),floor(0.7*nrow(data.mkl)),replace=FALSE)
degree=sapply(1:length(kernels), function(a) ifelse(kernels[a]=='p',2,0))
#kernels.gen splts the data into a training and test set, and generates the desired kernel matrices.
#Here we generate two gaussisan kernel matrices with sigma hyperparameter 2 and 0.05
K=kernels.gen(data=data.mkl[,1:2],train.samples=train.samples,kernels=kernels,sigma=sigma,degree=degre
C=0.05 #Cost parameter for DALMKL
K.train=K$K.train
K.test=K$K.test

 # parameters set up
 cri_outer = 0.01 # criterion for outer cycle, 0.01 is default by author
 cri_inner = cri_outer/10000  #criterion for inner cycle, this ratio is default by author
 calpha = 10 ### Lagrangian duality constraint parameter, must be positive, 10 is default by author
 max_iter_outer = 500 # maximum number of iterations in outer cycle
 max_iter_inner = 500 # maximum number of iterations in inner cycle
 ytr=data.mkl[train.samples,3]
 k.train=simplify2array(K.train) #Converts list of kernel matrices in to an array with is appropriate
 k.test=simplify2array(K.test)

 #Implement DALMKL with the hinge loss function
 spicy_svmb1n=SpicyMKL(k.train, ytr, 'hinge',C, cri_outer, cri_inner, max_iter_outer, max_iter_inner,
 spicysvmb1n_results=predict_Spicy(spicy_svmb1n$alpha,spicy_svmb1n$b, k = k.test)
 cm.DALMKL.svm=confusionMatrix(factor(sign(spicysvmb1n_results),levels=c(-1,1)), factor(data.mkl[-trai
 cm.DALMKL.svm
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction -1  1
##         -1 27  4
##         1   9 20
##
##                Accuracy : 0.7833
##                  95% CI : (0.658, 0.8793)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.002107
##
##                   Kappa : 0.5638
##
##  Mcnemar's Test P-Value : 0.267257
##
##             Sensitivity : 0.7500
##             Specificity : 0.8333
##          Pos Pred Value : 0.8710
##          Neg Pred Value : 0.6897
##              Prevalence : 0.6000
##          Detection Rate : 0.4500
##    Detection Prevalence : 0.5167
##       Balanced Accuracy : 0.7917
##
##        'Positive' Class : -1
##
```

```
  #Implement DALMKL with a logistic loss function
  spicy_logib1n=SpicyMKL(k.train, ytr,'logistic' ,C, cri_outer, cri_inner, max_iter_outer, max_iter_inn
  spicylogib1n_results=predict_Spicy(spicy_logib1n$alpha,spicy_logib1n$b, k = k.test)
  cm.DALMKL.logi=confusionMatrix(factor(sign(spicylogib1n_results),levels=c(-1,1)), factor(data.mkl[-tra
  cm.DALMKL.logi
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 29  6
##         1   7 18
##
##                Accuracy : 0.7833
##                  95% CI : (0.658, 0.8793)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.002107
##
##                   Kappa : 0.5517
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8056
##             Specificity : 0.7500
##          Pos Pred Value : 0.8286
##          Neg Pred Value : 0.7200
##              Prevalence : 0.6000
##          Detection Rate : 0.4833
##    Detection Prevalence : 0.5833
```

```
##       Balanced Accuracy : 0.7778
##
##        'Positive' Class : -1
##
```

```
#Convert C parameter from DALMKL implenetation to SimpleMKL and SEMKL implementation to make the four
C_SEMKL=C.convert(K.train,spicy_logib1n,C)

#Implement SimpleMKL
SimpleMKL.model=SimpleMKL.classification(k=K.train,data.mkl[train.samples,3], penalty=C_SEMKL)
cm.SimpleMKL=confusionMatrix(factor(prediction.Classification(SimpleMKL.model,ktest=K.test,data.mkl[t
cm.SimpleMKL
```

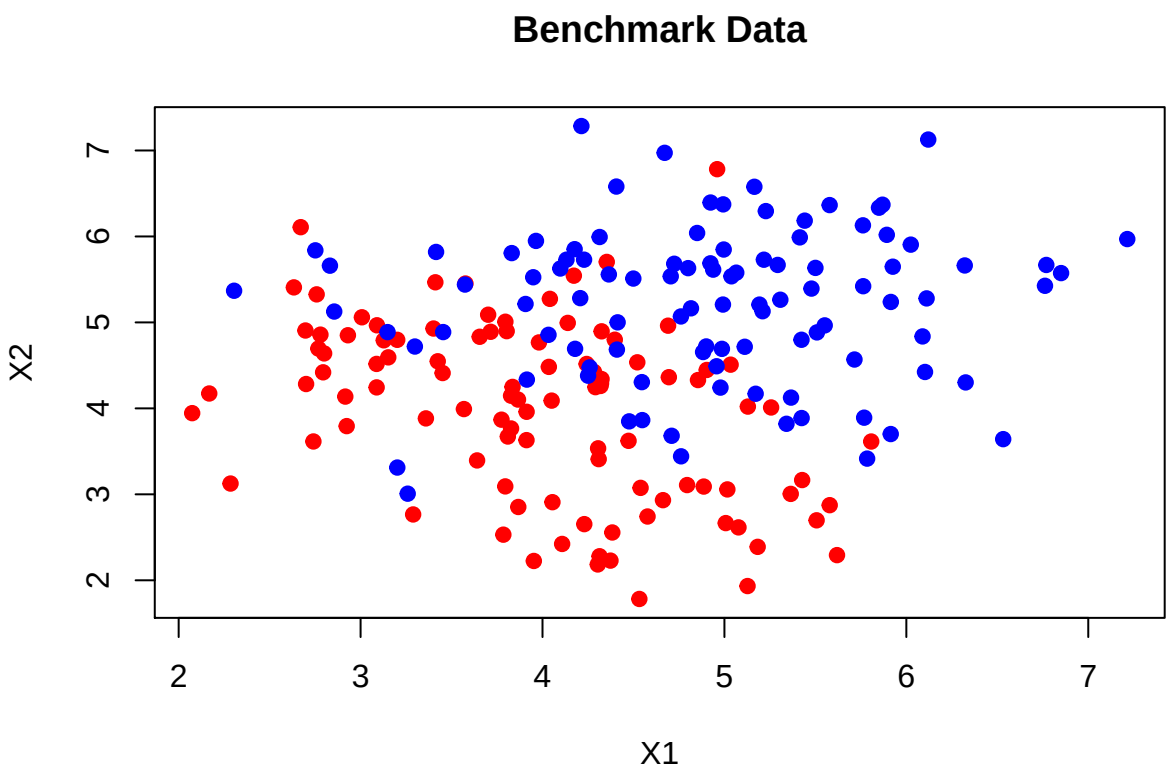
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 21  4
##         1  15 20
##
##                Accuracy : 0.6833
##                  95% CI : (0.5504, 0.7974)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.11696
##
##                   Kappa : 0.3871
##
##  Mcnemar's Test P-Value : 0.02178
##
##             Sensitivity : 0.5833
##             Specificity : 0.8333
##          Pos Pred Value : 0.8400
##          Neg Pred Value : 0.5714
##              Prevalence : 0.6000
##          Detection Rate : 0.3500
##    Detection Prevalence : 0.4167
##       Balanced Accuracy : 0.7083
##
##        'Positive' Class : -1
##
```

```
#Implement SEMKL
SEMKL.model=SEMKL.classification(k=K.train,data.mkl[train.samples,3], penalty=C_SEMKL)
cm.SEMKL=confusionMatrix(factor(prediction.Classification(SEMKL.model,ktest=K.test,data.mkl[train.sam
levels=c(-1,1)),factor(data.mkl[-train.samples,3],levels=c(-1,1)))
cm.SEMKL
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 36 22
##         1   0  2
##
##                Accuracy : 0.6333
##                  95% CI : (0.499, 0.7541)
```

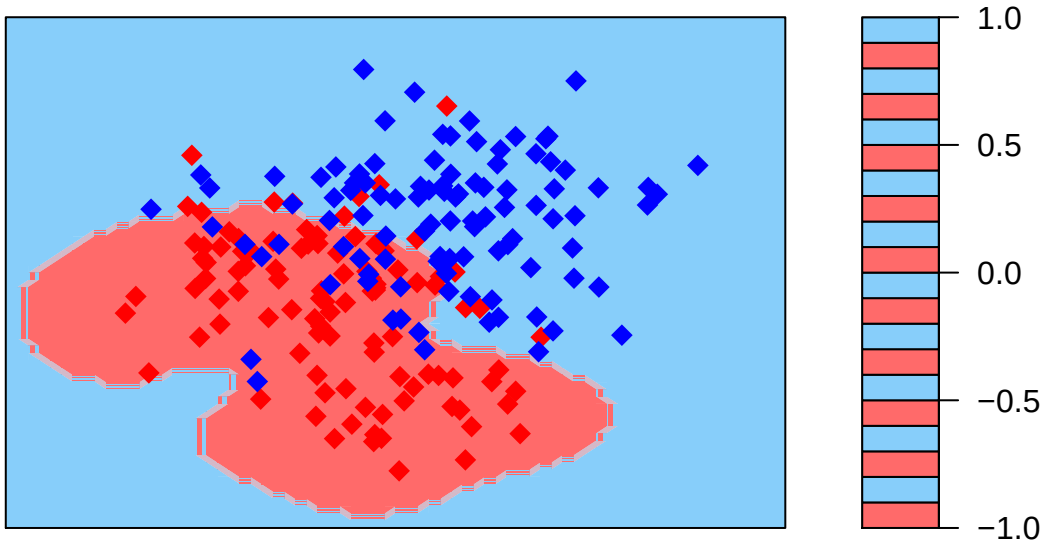
```
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.3493
##
##                   Kappa : 0.0984
##
##  Mcnemar's Test P-Value : 7.562e-06
##
##             Sensitivity : 1.00000
##             Specificity : 0.08333
##          Pos Pred Value : 0.62069
##          Neg Pred Value : 1.00000
##              Prevalence : 0.60000
##          Detection Rate : 0.60000
##    Detection Prevalence : 0.96667
##       Balanced Accuracy : 0.54167
##
##        'Positive' Class : -1
##
```

```
  #Selecting a plot in the middle to show the benefit of MKL over SVM
plot(benchmark.data[[4]][,-3],col=benchmark.data[[4]][,3]+3,main='Benchmark Data',pch=19,xlab='X1', ylab
```



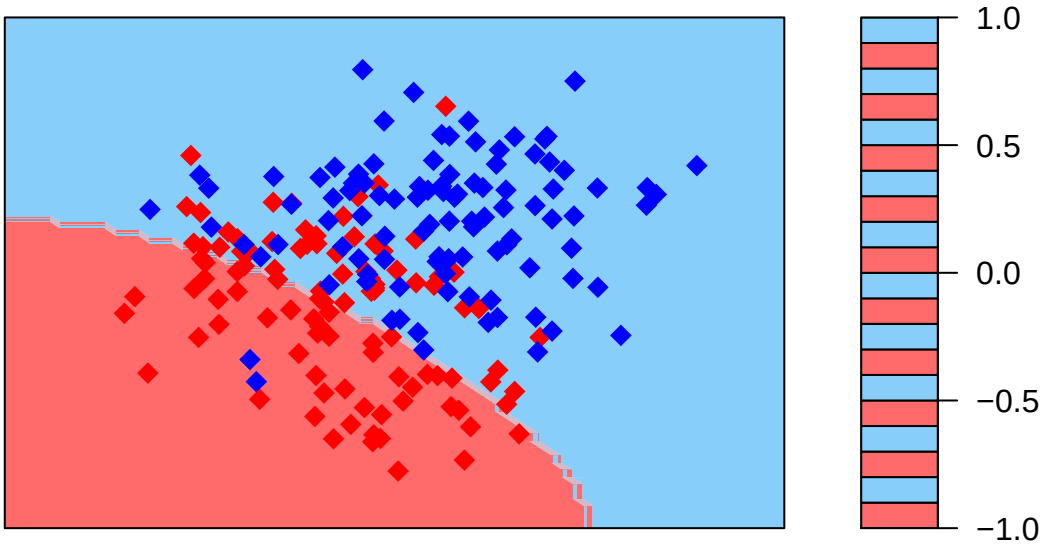

```
#Using the radial kernel with both hyperparameter individually and then in a combined analysis
C=100
K=kernels.gen(data=benchmark.data[[4]][,1:2],train.samples=train.samples,kernels=kernels,
              sigma=sigma,degree=degree,scale=rep(0,length(kernels)))
K.train=K$K.train
K.test=K$K.test
#MKL with only one candidate kernel is equivalent to SVM
#SVM with radial hyperparameter 2
rbf2=SEMKL.classification(k = list(K.train[[1]]),outcome = benchmark.data[[4]][train.samples,3],penalty
```

```
#SVM with radial hyperparameter 1/20
rbf.05=SEMKL.classification(k=list(K.train[[2]]),outcome = benchmark.data[[4]][train.samples,3],penalty
domain=seq(1,8,0.1)
grid=cbind(c(replicate(length(domain), domain)),c(t(replicate(length(domain), domain))))
predict.data=rbind(benchmark.data[[4]][train.samples,1:2],grid)
kernels.predict=kernels.gen(data=predict.data,train.samples=1:length(train.samples),kernels=kernels,
            sigma=sigma,degree=degree,scale=rep(0,length(kernels)))

predict2=prediction.Classification(rbf2, ktest = list(kernels.predict$K.test[[1]]),
                           train.outcome = benchmark.data[[4]][train.samples,3])

predict.05=prediction.Classification(rbf.05, ktest = list(kernels.predict$K.test[[2]]),
                                   train.outcome = benchmark.data[[4]][train.samples,3])

#Contour plot of the predicted values using the model where a single kernel was used
filled.contour(domain,domain, matrix(predict2$predict,length(domain),length(domain)),
               col = colorRampPalette(c('indianred1','lightskyblue'))(2),
               main='Classication Rule Hyperparameter=2',
               plot.axes={points(benchmark.data[[4]][,-3],col=benchmark.data[[4]][,3]+3,pch=18,cex=1.5)
```



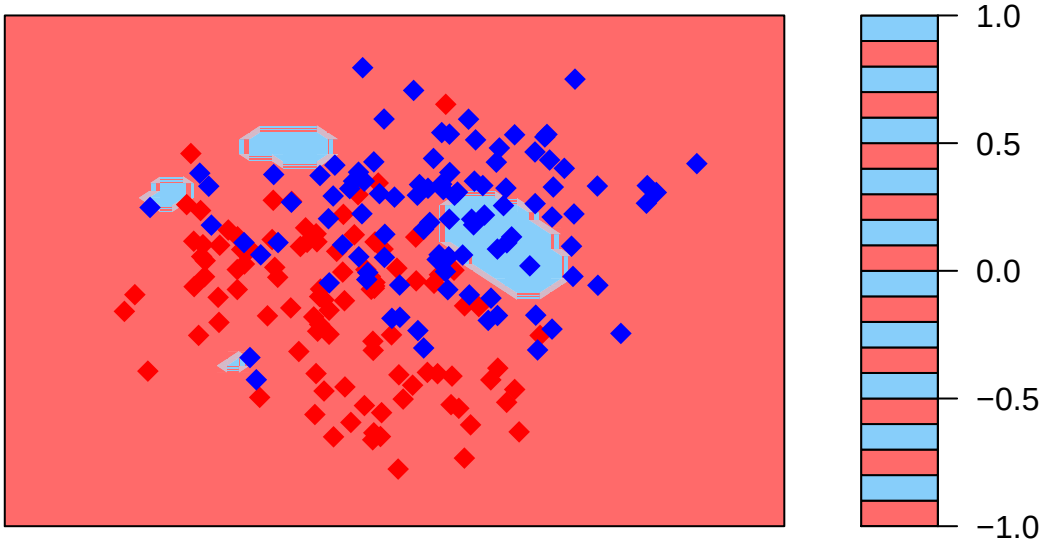

```
filled.contour(domain,domain, matrix(predict.05$predict,length(domain),length(domain)),
               col = colorRampPalette(c('indianred1','lightskyblue'))(2),
               main='Classication Rule Hyperparameter=0.05',
               plot.axes={points(benchmark.data[[4]][,-3],col=benchmark.data[[4]][,3]+3,pch=18,cex=1.5)
```

Classication Rule Hyperparameter=0.05




```
#############################################################################################
#Use the optimal model with the combination of kernels

predict.combined=prediction.Classification(SEMKL.model, ktest = kernels.predict$K.test,
                                    train.outcome = benchmark.data[[4]][train.samples,3])
filled.contour(domain,domain, matrix(predict.combined$predict,length(domain),length(domain)),
               col = colorRampPalette(c('indianred1','lightskyblue'))(2),
               main='Classication Rule MKL',
               plot.axes={points(benchmark.data[[4]][,-3],col=benchmark.data[[4]][,3]+3,pch=18,cex=1.5)
```

Classication Rule MKL



Realizations that fall in the light blue region will be classified as 1, while the points that fall in the light red region will be classified as -1. The points are the original observations. Notice that the two groups do overlap, and that a radial kernel with a large hyperparameter is able to classify in areas with overlap, while a radial kernel with a small hyperparameter can not. The kernel wieghts for this example are 0.9997 for a radial kernel 2 as a hyperparameter, and 0.0002 for radial kernel with 1/20 as a hyper parameter.

TCGA small example

These date are described in man/tcga.small.Rd, these data are log2(miRNA expression value+1).

```
rm(list=ls())
library(RMKL)
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(caret)

data(tcga.small)

normalized=apply(tcga.small,2,function(a) a/sqrt(sum(a^2)))


kernels=c('linear', rep('radial',8))
sigma=c(0,10^(-5:2))
training.samples=sample(1:nrow(normalized), 200, replace=FALSE)
K=kernels.gen(data=normalized[,-ncol(normalized)], sigma=sigma, degree=0, scale=0,kernels=kernels,
              train.samples=training.samples)


K.train=K$K.train
K.test=K$K.test
K.train.dal=simplify2array(K.train)
K.test.dal=simplify2array(K.test)


outcome=tcga.small[,ncol(tcga.small)]
y.train=outcome[training.samples]
cri_out = .01
cri_in = .000001
maxiter_out = maxiter_in = 500
C = 0.5*10^c(-2:0)
calpha = 10

mod.hinge=lapply(C, function(a){
mod.hinge = SpicyMKL(K.train.dal, y.train,'hinge' , a , cri_out, cri_in,
                     maxiter_out, maxiter_in, calpha)
prediction.hinge = predict_Spicy(mod.hinge$alpha,mod.hinge$b, K.test.dal)
cm=confusionMatrix(factor(sign(prediction.hinge),levels=c(-1,1)),
                   factor(outcome[-training.samples],levels=c(-1,1)))
return(list(model=mod.hinge,cm=cm))
})
```

```
## Does not converge in inner cycle.
```

```
mod.logistic=lapply(C, function(a){
  mod.logistic = SpicyMKL(K.train.dal, y.train,'logistic' , a , cri_out, cri_in,
```

```
                        maxiter_out, maxiter_in, calpha)
  prediction.logistic = predict_Spicy(mod.logistic$alpha,mod.logistic$b, K.test.dal)
  cm=confusionMatrix(factor(sign(prediction.logistic),levels=c(-1,1)),
                     factor(outcome[-training.samples],levels=c(-1,1)))
  return(list(model=mod.logistic,cm=cm))
})

C.SEMKL.logisic=sapply(1:length(mod.logistic), function(b) C.convert(K.train, mod.logistic[[b]]$model,C

C.SEMKL.hinge=sapply(1:length(mod.hinge), function(b) C.convert(K.train, mod.hinge[[b]]$model,C[b]))

SimpleMKL.results=lapply(C.SEMKL.hinge, function(b){
  SimpleMKL=SimpleMKL.classification(k=K.train, outcome=as.numeric(as.character(outcome[training.sample
  cm.SimpleMKL=confusionMatrix(factor(prediction.Classification(SimpleMKL,ktest=K.test,as.numeric(as.cha
                               factor(outcome[-training.samples]))
  return(list(cm=cm.SimpleMKL,model=SimpleMKL))})


SEMKL.results=lapply(C.SEMKL.hinge, function(b){
  SEMKL=SEMKL.classification(k=K.train, outcome=as.numeric(as.character(outcome[training.samples])), per
  cm.SEMKL=confusionMatrix(factor(prediction.Classification(SEMKL,ktest=K.test,as.numeric(as.character(
                               factor(outcome[-training.samples]))
  return(list(cm=cm.SEMKL,model=SEMKL))})
```