pathClass: Classification with prior knowledge on feature connectivity

(Version 1.0.0)

User‘s Guide

Marc Johannes
German Cancer Research Center
Heidelberg, Germany

September 27, 2010

Contents

1 Introduction

The package `pathClass` was developed for classification tasks with the usage of *prior* knowledge about the feature connectivity. At the German Cancer Research Center we are dealing mostly with biological data. Thus, in this vignette we demonstrate the usage of the package and its functions using biologically data.

2 What data do we need

For a *standard* classification task one needs a data matrix to train on as well as class labels which tell the algorithm to what class a sample belongs to. However, we now have an additional source of knowledge, i.e. a graph structure. For the algorithm to know which feature in the data matrix corresponds to which node in the graph we need a mapping as well. In the follwing sections we will describe the structure of these data objects and give examples how to create and use them.

2.1 The data matrix

At first we need a data matrix $\mathbf{D}^{n \times p}$ with n samples of p measurements. As an example we load:

```
> library(Biobase)
> data(sample.ExpressionSet)
> sample.ExpressionSet
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 26 samples
  element names: exprs, se.exprs
protocolData: none
phenoData
  sampleNames: A, B, ..., Z  (26 total)
  varLabels and varMetadata description:
    sex: Female/Male
    type: Case/Control
    score: Testing Score
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2
```

This data set contains 500 features measured in 26 samples. However, we need the transposed version of it:

```
> x <- t(exprs(sample.ExpressionSet))
> dim(x)
```

```
[1]  26 500
```

2.2 The class labels

We create some class labels using the `phenoData` object:

```
> y <- factor(pData(sample.ExpressionSet)$sex)
> y
```

```
 [1] Female Male   Male   Male   Female Male   Male   Male   Female Male  
[11] Male   Female Male   Male   Female Female Female Male   Male   Female
[21] Male   Female Male   Male   Female Female
Levels: Female Male
```

From this output we can see that `y` has the same length as `nrow(x)`, i.e. 26.

2.3 The graph

As a next step we have to create a adjacency matrix that represents the connectivity of the features in `x`. Therefore, we download from http://www.hprd.org/download the file of binary protein-protein interactions in tab delimited format. After extracting the archive we use `pathClass` to read the tab-delimited file:

```
> library(pathClass)
```

```
> hprd <- read.hprd('BINARY_PROTEIN_PROTEIN_INTERACTIONS.txt')
```

However, for the purpose of the vignette we will use a random graph with RefSeq IDs as `rownames` and `colnames`. This graph can be loaded as follows:

```
> data(adjacency.matrix)
```

This gives us an adjacency matrix of dimensions 500×500. Since most classification algorithm can "only" use those features which that are present in both the data matrix `x` and the `adjacency.matrix` we have to match both objects to each other. Therefore, we need a mapping containing the information which protein of `adjacency.matrix` matches to which probe set in `x`.

2.4 The mapping

For most microarrays there is a annotation package available. Since we are dealing with expression data from chip `hgu95av2` we load the corresponding annotation package and create a mapping from probe set ID to protein ID:

```
> library('hgu95av2.db')
> mapped.probes <- mappedkeys(hgu95av2REFSEQ)
> refseq <- as.list(hgu95av2REFSEQ[mapped.probes])
> times <- sapply(refseq, length)
> mapping <- data.frame(probesetID=rep(names(refseq), times=times),
+                       graphID=unlist(refseq),
+                       row.names=NULL,
+                       stringsAsFactors=FALSE)
> mapping <- unique(mapping)
> head(mapping)
```

```
  probesetID      graphID
1    1000_at NM_001040056
2    1000_at NM_001109891
3    1000_at    NM_002746
4    1000_at NP_001035145
5    1000_at NP_001103361
6    1000_at    NP_002737
```

Now we have a mapping with 42865 rows. It is important that this mapping has at least two columns named `graphID` and `probesetID` since those names are needed internally when `pathClass` makes use of the mapping.
In a next step we can make use of the function `matchMatrices()` to match the data matrix `x` to the `adjacency.matrix`:

```
> matched <- matchMatrices(x=x, adjacency=adjacency.matrix, mapping=mapping)
```

The list `matched` contains copies of `x`, `adjacency.matrix` and `mapping` however with matching dimensions. Thus, these objects can now be used for classification.

3 Which classification methods are available

That far, all classification algorithms we implemented are based on the support vector machine (SVM, Vapnik and Cortes 1995). As a standard tool we provide the recursive feature elimination (SVM-RFE, Guyon et al. 2002) algorithm for the SVM. This algorithm performs a feature selection, however it makes no use of *prior* knowledge. In addition to SVM-RFE we implemented three other SVM-based algorithm that use *prior* knowledge:

1. Reweighted Recursive Feature Elimination (RRFE, Johannes et al. 2010)
2. Network-based SVM (Zhu et al., 2009)
3. Graph SVM (Rapaport et al., 2007)

The functions to train these methods are called: `fit.rfe`, `fit.rrfe`, `fit.networkBasedSVM` and `fit.graph.svm`, respectively. The user can use these functions directly to obtain a fit object of the corresponding algorithm or use the wrapper-function `crossval()` to perform a x times repeated y-fold cross-validation. Additionally the `crossval()` function is able to make use of the multicore architecture of modern PCs or a computing cluster. To use the parallel version of the method the user has to load the library `multicore` prior to calling `crossval()` and to set the parameter `parallel` to `TRUE`.

3.1 Reweighted Recursive Feature Elimination

The RRFE method can be run without using the mapping created above. The reason for this is, that the method can use all features if the user sets the paramter `useAllFeatures` to TRUE. Therefore, this method has its own, internal mapping routine. RRFE has an tuning parameter d $\in (0, 1)$ that controls the influence of the graph structure on the ranking of the genes. A value of d $\rightarrow 1$ puts more weight on the connectivity infromation whereas d $\rightarrow 0$ relies more on the expression data. To use the RRFE method one can use:

```
> res.rrfe <- crossval(x,
+                      y,
+                      DEBUG=TRUE,
+                      theta.fit=fit.rrfe,
+                      folds=3,
+                      repeats=1,
+                      parallel=TRUE,
+                      Cs=10^(-3:3),
+                      mapping=mapping,
+                      Gsub=adjacency.matrix,
+                      d=1/2)
```

or, to use all features:

```
> res.rrfe <- crossval(x,
+                      y,
+                      DEBUG=TRUE,
+                      theta.fit=fit.rrfe,
+                      folds=3,
+                      repeats=1,
+                      parallel=TRUE,
+                      Cs=10^(-3:3),
+                      useAllFeatures=TRUE,
+                      mapping=mapping,
+                      Gsub=adjacency.matrix,
+                      d=1/2)
```

Please, have a look into the help files or the paper (Johannes et al., 2010) for more information on the useAllFeatures option.

3.2 network-based SVM

The network-based support vector machine (Zhu et al., 2009) needs the mapping from above, since the dimensions of the data objects have to match exactely. However, instead of an adjacency matrix it needs an adjacency list which we have to create before:

```
> ad.list <- as.adjacencyList(matched$adjacency)
> res.nBSVM <- crossval(matched$x,
+                       y,
+                       theta.fit=fit.networkBasedSVM,
+                       folds=3,
+                              repeats=1,
+                              DEBUG=TRUE,
+                              parallel=FALSE,
+                              adjacencyList=ad.list,
+                              lambdas=10^(-1:2),
+                              sd.cutoff=50)
```

Since, the algorithm internally uses `lpSolve`, it has to calculate a constraints-matrix. Thus, when having lots of features this matrix can become very big. Therefore, we added the parameter `sd.cutoff` which only keeps genes with standard deviation $\geq$ `sd.cutoff`.

3.3 graph SVM

Rapaport et al. (2007) developed a supervised classification framework which we refer to as "graph SVM". This methods makes use of a so-called diffusion kernel (**REF**), which has to be calculated before using this method:

```
> dk <- calc.diffusionKernel(L=matched$adjacency,
+                            is.adjacency=TRUE,
+                            beta=0)
> res.gSVM <- crossval(matched$x,
+                      y,
+                      theta.fit=fit.graph.svm,
+                      folds=5,
+                      repeats=2,
+                      DEBUG=TRUE,
+                      parallel=FALSE,
+                      Cs=10^(-3:3),
+                      mapping=matched$mapping,
+                      diffusionKernel=dk)
```

Were `beta` is a tuning parameter that controls the extent of diffusion. This parameter should be optimized.

4 Showing the results

We can have a look on the results by typing:

```
> plot(res.rrfe, toFile=F)
```

We get a boxplot for each repeat of the cross-validation showing the distribution of AUC's obtained by the classifiers trained in the repeat as well as a receiver operator characteristic (ROC) curve showing the overall performance.
Additionally we can extract the features which have been chosen by the classifier by using the following function:

```
> extractFeatures(res.rrfe, toFile=T, fName='OurFeatures.csv')

> switcher <- function(res){
+   for(j in 1:ncol(res$auc)){
+     for(i in 1:nrow(res$auc)){
+       if(res$auc[i,j] < 0.5){
+         ind <- res$fits[[j]][[i]]$tst.indices
+         res$cv[ind,j] = res$cv[ind,j] * -1
+         res$auc[i,j] = 1 - res$auc[i,j]
+       }
+     }
+   }
+   return(res)
+ }
```

References

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002. URL http://www.springerlink.com/index/W68424066825VR3L.pdf.

M. Johannes, J. C. Brase, H. Fröhlich, S. Gade, M. Gehrmann, M. Fälth, H. Sültmann, and T. Beißbarth. Integration of pathway knowledge into a reweighted recursive feature elimination approach for risk stratification of cancer patients. *Bioinformatics*, 26(17):2136–2144, Jun 2010. doi: 10.1093/bioinformatics/btq345. URL http://dx.doi.org/10.1093/bioinformatics/btq345.

F. Rapaport, A. Zinovyev, M. Dutreix, E. Barillot, and J.-P. Vert. Classification of microarray data using gene networks. *BMC Bioinformatics*, 8:35, 2007. doi: 10.1186/1471-2105-8-35. URL http://dx.doi.org/10.1186/1471-2105-8-35.

V. Vapnik and C. Cortes. Support-vector networks. *Machine Learning*, Jan 1995. URL http://www.springerlink.com/index/K238JX04HM87J80G.pdf.

Y. Zhu, X. Shen, and W. Pan. Network-based support vector machine for classification of microarray samples. *BMC Bioinformatics*, 10 Suppl 1:S21, 2009. doi: 10.1186/1471-2105-10-S1-S21. URL http://dx.doi.org/10.1186/1471-2105-10-S1-S21.