The mpda package vignette

Lars Snipen

1 Introduction

This package is an implementation of simple methods for classification based on PLS and LDA. It also includes an algorithm for variable selection in this setting, based on the principle algorithm that we published in `http://www.ncbi.nlm.nih.gov/pubmed/22142365`. It deviates from the publiscation by using the `pda()` instead of the `cppls()` from the `pls`-package, and by running the selection 'to the end' in order to see the entire selection process before deciding on the final model. Below we describe how to use the functions in this package.

2 Example data

A variable selection problem means we search systematically for some subset of variables who are capable of predicting some response variable as good as possible. Thus, we are in the setting of a pattern recognition problem, using some kind of supervised learning algorithm. There are endless examples of such problems. We focus here on an example data set in this package.

The `microbiome` data set consists of bacterial samples collected from various body-fluids or sites for 100 persons. The samples have been subject to targeted sequencing of the 16S rRNA gene. All 16S marker reads have been given taxonomic classification, i.e. each read has been assigned to some prokaryotic genus. For each genus is counted the number of reads, and for each sample we have a vector of proportions (normalized read-counts to 1.0). We load the data set:

```
library(mpda)

## Loading required package:  pls
##
## Attaching package:  'pls'
## The following object is masked from 'package:stats':
##
##    loadings
## Loading required package:  MASS

data(microbiome)
```

The first column of this `data.frame` hold the class labels, and the ramining columns the taxonomic profiles we will use as predictors.

```
y.all <- microbiome[,1]              # Class labels (body fluid)
X.all <- as.matrix(microbiome[,-1]) # Taxonomic profiles
```

We can make some brief inspection of the data:

```
table(y.all)

## y.all
##   Fecal   Nasal    Oral    Skin Vaginal
##      20      20      20      20      20
```

which means we have equal number of samples of each class. We also verify that we have 100 microbiome profiles, each having 1005 values (transformed genus read-count) in `microbiome.X`:

```
dim(X.all)

## [1]  100 1005
```

Before we proceed, we only consider the sub-problem with 2 categorical class-labels. This formas the basis for the following methods. Let us consider only the first 40 samples from above, those from either `Oral` or `Fecal` body.fluids:

```
y <- y.all[1:40]
X <- X.all[1:40,]
```

The pattern recognition problem is as follows: Find a rule that discriminates `Oral` from `Fecal` samples based on the (patterns in the) taxonomic profiles. For this we need a supervised learning algorithm.

3 The `pda()` function

This is a simple implementation that combines two commonly used supervised learning methods, working in sequence.

First, the PLS method is used to reduce the dimension of the variable-space from 1005 to something much smaller. PLS will search for linear combinations of the original variables that explains the response. Since PLS can only work with numerical responses, the factor response is dummy-coded as 0's and 1's. This step uses the `plsr()` function from the `pls`-package, with the `oscorespls` algorithm.

Next, the scores from the PLS-step are used as the alternative explanatory variables in a standard LDA, using the `lda()` function in the `MASS`-package. One LDA-model is fitted for each possible choice of dimension.

Here is an example of how to use `pda`:

```
trained.mod <- pda(y,X,prior=c(1,1),max.dim=5)
summary(trained.mod$PLS)

## Data:  X dimension: 40 1005
```

```
##  Y dimension: 40 1
## Fit method: oscorespls
## Number of components considered: 5
## TRAINING: % variance explained
##        1 comps  2 comps  3 comps  4 comps  5 comps
## X        63.24    68.25    82.45    86.82    89.48
## y.dum    81.91    96.08    96.93    97.87    98.28
```

```
summary(trained.mod$LDA)
```

```
##      Length Class Mode
## [1,] 8      lda   list
## [2,] 8      lda   list
## [3,] 8      lda   list
## [4,] 8      lda   list
## [5,] 8      lda   list
```

The `prior` indicates our prior belief in observing the two classes. Usually this is flat (like here), i.e. both classes are equally likely. Since we used `max.dim=5` we get trained PLS-models for 1 to 5 dimensions, and corresponding trained LDA models in a list of length 5.

If we have collected new taxonomix profiles, we can now classify these using the trained model from above. Since we have no such new data, we use the same data again:

```
lst <- predict(trained.mod,newdata=X)
y.hat <- lst[[2]]$Classifications
confusion.matrix <- table(y,y.hat)
print(confusion.matrix)
```

```
##        y.hat
## y       Fecal Oral
##   Fecal    20    0
##   Oral      0   20
```

Here we decided to use the 2-dimension model (`lst[[2]]`) and compared its `Classifications` to the actual response. We can see the classification was perfect, however, this is not a proper prediction since we used the same data for training.

4 Regularization - the `pdaDim()` function

Deciding on the optimal dimension is crucial in any PLS-based method, like `pda`. The function `pdaDim` can be used to estimate the number of dimensions to use. It will perform a cross-validation, and compute the classification accuracy for all dimensions from this.

When searching systematically the dimensions 1,2,... there is always a maximum accuracy obtained somewhere, and a too small or too large dimension will be sub-optimal. However, often many different choices of dimension will give more or less the same accuracy, and it is quite random which of these will

produce the maximum value in any given data set. In our regularization we seek the *smallest* dimension which gives not significantly poorer accuracy than the maximum. This means we get a much more stable selection of dimension.

The regularization is adjusted by the rejection level of the McNemar-test. This test is used to test if a smaller dimension gives significantly poorer accuracy than the maximum. The smaller the rejection level, the harder regularization. Here is an example with the example data:

```
lst <- pdaDim(y,X,reg=0.1,prior=c(1,1),max.dim=5)

## pdaDim:
##   cross-validation...
##    maximum accuracy 1 at 2 dimensions...
##    selected accuracy 0.975 at 1 dimensions
```

Here we used the default value 0.1 for the regularization parameter `reg`. This resulted in dimension 1 being the optimal, since it has an accuracy of 0.97. The dimension giving maximum accuracy in the present data set can also be selected, by setting `reg=1.0`, which means any reductions in accuracy are significantly poorer than the maximum:

```
lst <- pdaDim(y,X,reg=1.0,prior=c(1,1),max.dim=5)

## pdaDim:
##   cross-validation...
##    maximum accuracy 1 at 2 dimensions...
##    selected accuracy 1 at 2 dimensions
```

5 The Eliminator

The function `eliminator` will use `pda` and `pdaDim` functions repeatedly in a search for a subset of the variables that gives stable and good classification.

In the example data we have 1005 variables in `X`, but it is more than likely that many of these are of little value for discriminating between the body fluids. Which bacteria are the important ones? This is the question that the `eliminator` deals with.

The basic idea is to start out with all variables, rank them according to some criterion for importance, and eliminate a fraction of the unimportant ones. This is repeated until there are no unimportant variables left. The results of this elimination is then returned, and from this we decide what is the best subset of variables for our purpose, as shown below.

Here is an example, where we use default regularization, setting `reg=0.1`:

```
lst <- eliminator(y,X,reg=0.1,prior=c(1,1),max.dim=5)

## The Eliminator:
##    full model has 1005 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 758 variables, accuracy = 0.975 using 1 dimensions
```

```
##    eliminated to 572 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 432 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 327 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 248 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 189 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 144 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 109 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 83 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 63 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 48 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 36 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 27 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 20 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 15 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 11 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 8 variables, accuracy = 0.975 using 1 dimensions
##    eliminated to 6 variables, accuracy = 0.95 using 1 dimensions
##    eliminated to 5 variables, accuracy = 0.95 using 1 dimensions
##    eliminated to 4 variables, accuracy = 0.95 using 1 dimensions
##    eliminated to 3 variables, accuracy = 0.95 using 1 dimensions
##    eliminated to 2 variables, accuracy = 0.95 using 1 dimensions
##    eliminated to 1 variables, accuracy = 0.875 using 1 dimensions
```

We notice the elimination runs all the way to only 1 variable left. But, in order to decide what is optimal we need to inspect the resulting `lst`. This is a list with 2 matrices inside it. The first matrix, named `Elimination` gives us the results we need:

```
print(lst$Elimination)
```

```
##              N.variables Accuracy   P.value
## Iteration 1         1005    0.975 1.0000000
## Iteration 2          758    0.975 1.0000000
## Iteration 3          572    0.975 1.0000000
## Iteration 4          432    0.975 1.0000000
## Iteration 5          327    0.975 1.0000000
## Iteration 6          248    0.975 1.0000000
## Iteration 7          189    0.975 1.0000000
## Iteration 8          144    0.975 1.0000000
## Iteration 9          109    0.975 1.0000000
## Iteration 10          83    0.975 1.0000000
## Iteration 11          63    0.975 1.0000000
## Iteration 12          48    0.975 1.0000000
## Iteration 13          36    0.975 1.0000000
## Iteration 14          27    0.975 1.0000000
## Iteration 15          20    0.975 1.0000000
## Iteration 16          15    0.975 1.0000000
## Iteration 17          11    0.975 1.0000000
## Iteration 18           8    0.975 1.0000000
## Iteration 19           6    0.950 1.0000000
```

```
## Iteration 20           5    0.950 1.0000000
## Iteration 21           4    0.950 1.0000000
## Iteration 22           3    0.950 1.0000000
## Iteration 23           2    0.950 1.0000000
## Iteration 24           1    0.875 0.1336144
```

For each iteration we get listed the number of variables left, the accuracy achieved in the cross-validation, and the p-value of the McNemar-test. The latter tells us if a drop in performance is significant (small p-value) or not. We notice that at Iteration 23 there is a drop in accuracy from 0.95 to 0.875. The corresponding p-value is around 0.13. It seems like a sensible choice to stop at Iteration 23.

At iteration 23 there are 2 selected variables. The matrix `Selected` tells us which 2 variables:

```
print(which(lst$Selected[23,]))

##   Bacteroides Streptococcus
##           133           879
```

It turns out that only the two genera *Bacteroides* and *Streptococcus* is enough to distinguish `Oral` from `Fecal` samples in the majority of cases.