Variational Bayesian Multinomial Probit Regression for Gaussian Process multi-class classification

Nicola Lama
Mark Girolami

October 17, 2016

Introduction

The *vbmp* package implements multinomial probit regression with Gaussian Process priors and estimates class membership posterior probability employing variational and sparse approximation to the full posterior. This software also incorporates feature weighting by means of Automatic Relevance Determination.
For more detailed description of classification using multinomial-probit regression model with Gaussian Process priors, refer to **?**.
To illustrate the potential of the method, a couple of examples are presented on a synthetic toy dataset and on literature data from the breast cancer study by **?**

The `vbmp` function

The `vbmp` function carries out multi-class classification training and testing. The `control` parameter list provides optional control settings. The default Gaussian Process covariance functions are radial basis-style functions but other choice are possible. Whether or not to infer the scale parameter of the covariance functions is an option (`bThetaEstimate=T`). The target values `t.class` and `t.class.TEST` can be factors or integer values in 1:n where n is the number of classes.

The synthetic toy dataset

In this illustrative example dataset is composed of uniformly distributed 2-D data points drawn from 3 nonlinearly separable classes which take the form of two annular rings and one zero-centered bivariate Gaussian with spherical covariance matrix (see Fig. **??**). Four additional random noise parameters were added to the dataset
Gaussian kernel functions were adopted with scale parameter inferred by the `vbmp` method using vague hyperpriors (Gamma prior over covaraince params defaults to 10^{-6}). Monitor diagnostics are depicted in figure **??** where the progress of Automatic Relevance Detection (ARD) **?** is evident with the four irrelevant features are effectively removed from the model.

```
> ## Init random number generator
> set.seed(123);
```

Define a function to generate the dataset:

```
> genSample <- function(n, noiseVar=0) {
+     KF <- 10; # magnify factor
+     nt <- round(n/3,0)#
+     ## class 1 and 2 (x ~ U(0,1))
+     u <- 4. * matrix(runif(2*KF*n), nrow=KF*n, ncol=2) - 2.;
+     i <- which(((u[, 1]^2 + u[, 2]^2) > .1) & ((u[, 1]^2 + u[, 2]^2) < .5) );
+     j <- which(((u[, 1]^2 + u[, 2]^2) > .6) & ((u[, 1]^2 + u[, 2]^2) < 1) );
+     X <- u[c(sample(i,nt), sample(j,nt)),];
+     t.class <- c(rep(1, nt),rep(2, nt));
+     ## class 3 (x ~ N(0,1))
+     x <- 0.1 * matrix(rnorm(2*KF*length(i)), ncol=2, nrow=length(i)*KF );
+     k <- which((x[, 1]^2 + x[, 2]^2) < 0.1);
+     nt <- n - 2*nt;
+     X <- rbind(X, x[sample(k,nt), ]);
+     t.class <- c(t.class, rep(3, nt));
+     ## limit number
+     #n <- min(n, nrow(X)); i <- sample(1:nrow(X),n);X <- X[i,]; t.class <- t.class[i];
+     ## add random coloumns
+     if (noiseVar>0) X <- cbind(X, matrix(rnorm(noiseVar*n), ncol=noiseVar, nrow=nrow(X)));
+     structure( list( t.class=t.class, X=X), class="MultiNoisyData");
+ }
```

Set the number of additional noisy input parameters:

```
> nNoisyInputs <- 4;
```

Set the sample sizes:

```
> Ntrain <- 100;
> Ntest  <- Ntrain * 5;
```

Generate training and test samples independently:

```
> dataXtTrain <- genSample(Ntrain, nNoisyInputs);
> dataXtTest  <- genSample(Ntest,  nNoisyInputs);
```

Initialization of scale covariance function and of the maximum number of iterations:

```
> theta <- rep(1., ncol(dataXtTrain$X));
> max.train.iter <- 12;
```

Train, test and plot monitors

```
> library(vbmp);
> res <- vbmp( dataXtTrain$X, dataXtTrain$t.class,
+              dataXtTest$X, dataXtTest$t.class, theta,
+              control=list(bThetaEstimate=T, bMonitor=T, maxIts=max.train.iter));
```

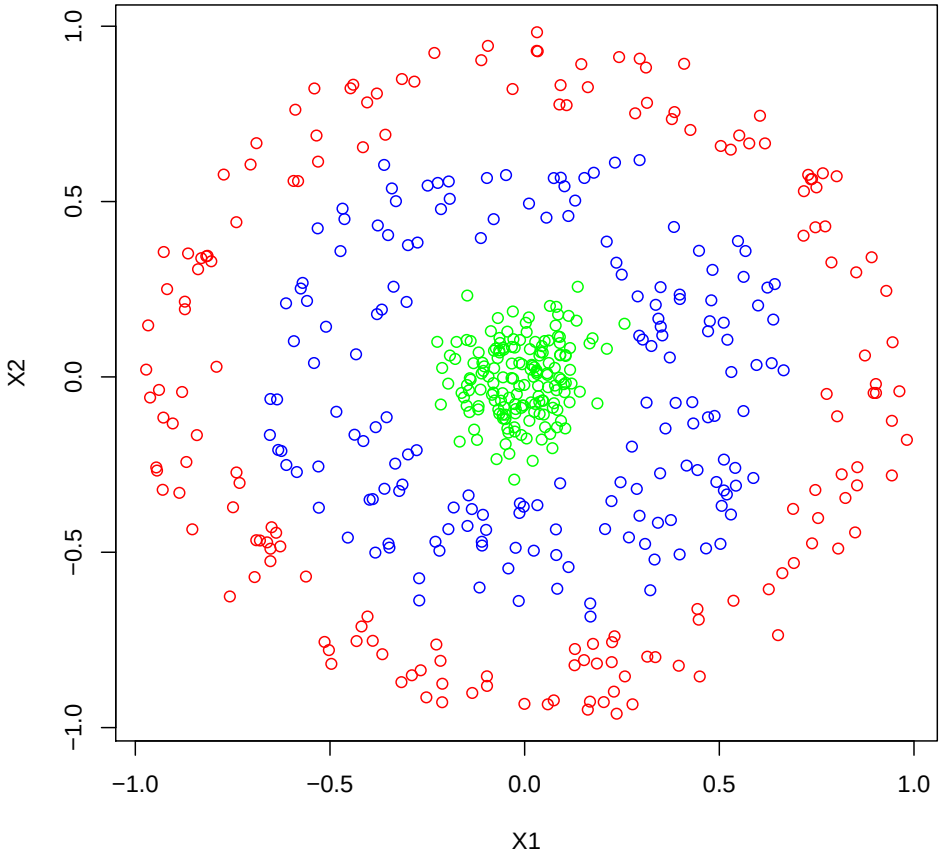

Figure 1: Scatter plot of the first two dimensions of the toy dataset. Classes are identified by different colors

Print the out-of-sample prediction error

```
> predError(res);
```

```
[1] 0.028
```

```
> ## plot convergence diagnostics (same as setting bPlotFitting=T)
> plotDiagnostics(res);
```

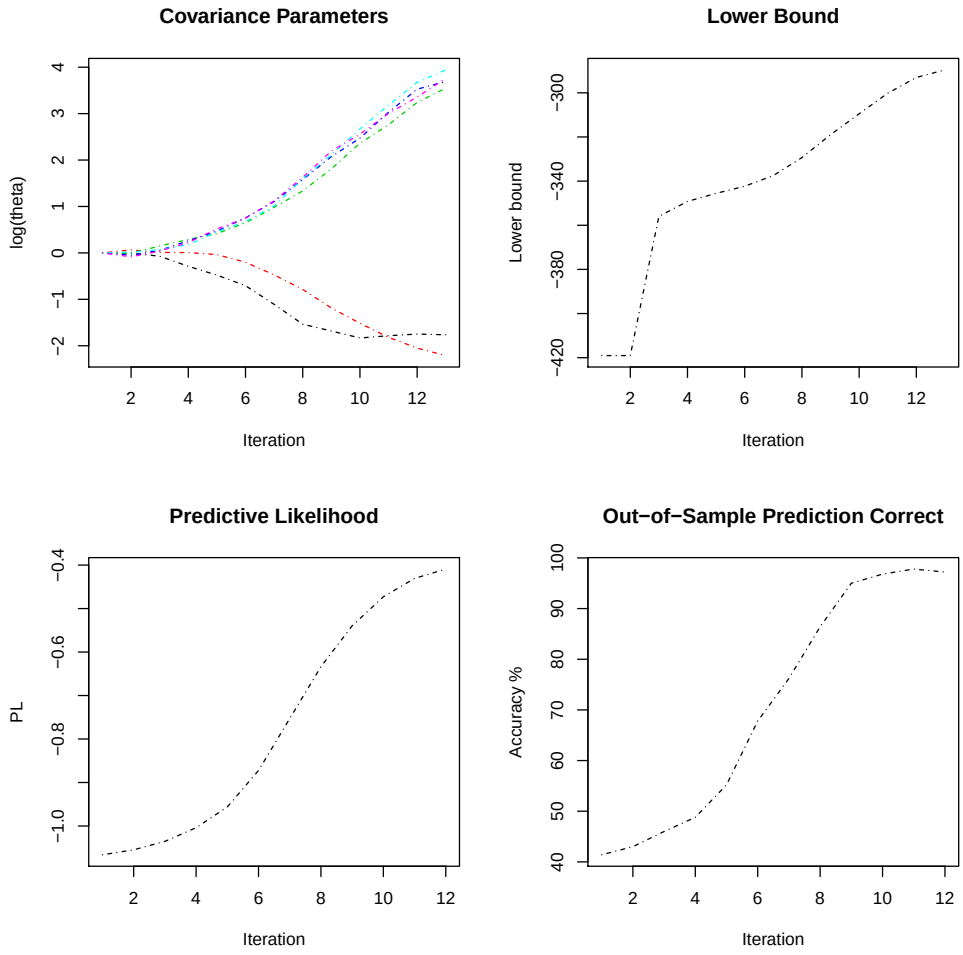

Figure 2: Monitor diagnostics for the evolution of: posterior means for the Gaussian kernel scale parameters (top-left), lower bound on the marginal likelihood (top-right), predictive likelihood (bottom-left) and out-of-sample accuracy (0/1-error loss)

Breast cancer dataset

The dataset consist of 8080 genes and 30 samples from three classes of individual with specific gene mutations. As reported in **?** using a linear inner product covariance function the Leave-One-Out error is zero which improves on an SVM multiclass classifier adopted by **?**.

```
>   library("Biobase");
>   data("BRCA12");
>   brca.y <- BRCA12$Target.class;
>   brca.x <- t(exprs(BRCA12));
```

Define a function to set up the target class predictor during cross-validation. It uses linear inner product covariance function and 24 training iterations and sets iteration threshold to 10^{-8}. Initialize all theta values to 1 and, in this case, it is not necessarily to run the importance sampler to obtain the weightings in the features.

```
>   predictVBMP <- function(x) {
+       sKernelType <- "iprod";
+       Thresh      <- 1e-8;
+       theta       <- rep(1.0, ncol(brca.x));
+       max.train.iter <- 24;
+       resX <- vbmp( brca.x[!x,], brca.y[!x],
+                     brca.x[ x,], brca.y[ x],
+                     theta,  control=list(bThetaEstimate=F,
+                     bPlotFitting=F, maxIts=max.train.iter,
+                     sKernelType=sKernelType, Thresh=Thresh));
+       predClass(resX);
+   }
```

In the following code, `Kfold` is the number of cross validation folds and it is set equal to `n`, the number of pts in the dataset, in order to obtain the Leave-One-Out approach. `res` collects the predicted targets.

```
>   n     <- nrow(brca.x);
>   Kfold <- n; # number of folds , if equal to n then LOO
>   samps <- sample(rep(1:Kfold, length=n), n, replace=FALSE);
>   res   <- rep(NA, n);
>   print(paste("Crossvalidation started...... (",n,"steps )"));
```

```
[1] "Crossvalidation started...... ( 30 steps )"
```

```
>   for (x in 1:Kfold) {
+       cat(paste(x,", ",ifelse(x%%10==0,"\n",""),sep=""));
+       flush.console();
+       res[samps==x] <- predictVBMP(samps==x);
+   }
```

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
```

The Cross-validated error rate can be finally evaluated:

```
>     CVerrorRate <- round(sum(res!=brca.y)/n,2);
>   # don't print out the results, owing to the fake predictVBMP
```