The Iterative Bayesian Model Averaging Algorithm for Survival Analysis: an Improved Method for Gene Selection and Survival Analysis on Microarray Data

Amalia Annest, Roger E. Bumgarner, Adrian E. Raftery, and Ka Yee Yeung

October 17, 2016

1 Introduction

Survival analysis is a supervised learning technique that in the context of microarray data is most frequently used to identify genes whose expression levels are correlated with patient survival prognosis. Survival analysis is generally applied to diseased samples for the purpose of analyzing time to event, where the event can be any milestone of interest (e.g., metastases, relapse, or death). Typically, the interest is in identifying genes that are predictive of a patient's chances for survival. In such cases, both the accuracy of the prediction and the number of genes necessary to obtain a given accuracy is important. In particular, methods that select a small number of relevant genes and provide accurate patient risk assessment can aid in the development of simpler diagnostic tests. In addition, methods that adopt a weighted average approach over multiple models have the potential to provide more accurate predictions than methods that do not take model uncertainty into consideration. To this end, we developed the iterative Bayesian Model Averaging (BMA) method for gene selection and survival analysis on microarray data (**?**). Typical gene selection and survival analysis procedures ignore model uncertainty and use a single set of relevant genes (model) to predict patient risk. BMA is a multivariate technique that takes the interaction of variables (typically genes) and model uncertainty into account. In addition, the output of BMA contains posterior probabilities for each prediction, which can be useful in assessing the correctness of a given prognosis.

1.1 Bayesian Model Averaging (BMA)

Bayesian Model Averaging (BMA) takes model uncertainty into consideration by averaging over the posterior distributions of a quantity of interest based on multiple models, weighted by their posterior model probabilities (**?**). The posterior probability that a test sample is at risk for the given event is the posterior probability that the test sample is at risk for the given even computed using the set of relevant genes in model M_k multiplied by the posterior probability of model M_k, summed over a set of 'good' models M_k.

1.2 Iterative Bayesian Model Averaging (BMA) Algorithm for Survival Analysis

The BMA algorithm we have described is limited to data in which the number of variables is greater than the number of responses. In the case of performing survival analysis on microarray data, there are typically thousands or tens of thousands of genes (variables) and only a few dozens samples (responses).

In this package, the iterative BMA algorithm for survival analysis is implemented. In the iterative BMA algorithm for survival analysis, we start by ranking the genes in descending order of their log likelihood using a univariate measure such as the Cox Proportional Hazards Model (**?**). In this initial preprocessing step, genes with a larger log likelihood are given a higher ranking. Once the dataset is sorted, we apply the traditional BMA algorithm to the $maxNvar$ top log-ranked genes. We use a default of $maxNvar = 25$, because the traditional BMA algorithm employs the leaps and bounds algorithm that is inefficient for numbers of genes (variables) greater than 30. In the next step, genes to which the BMA algorithm assigns low posterior probabilities of being in the predictive model are removed. In our study, we used 1% as the threshold and eliminated genes with posterior probabilities $< 1\%$. Suppose m genes are removed. The next m genes from the rank ordered log likelihood scores are added back to the set of genes so that we maintain a window of $maxNvar$ genes and apply the traditional BMA algorithm again. These steps of gene swaps and iterative applications of BMA are continued until all genes are considered.

2 Some examples

The R package `BMA` is required to run the key commands in this package.

```
> library(BMA)
> library(iterativeBMAsurv)
```

An adapted diffuse large B-Cell lymphoma (DLBCL) dataset (**?**) is included for illustration purposes. The adapted DLBCL dataset consists of the top 100 genes selected using the Cox Proportional Hazards Model. The training set consists of 65 samples, while the test set consists of 36 samples. In the following examples, we chose parameters to reduce computational time for illustrative purposes. Please refer to our manuscript ((**?**)) for recommended input parameters.

```
> ## Use the sample training data. The data matrix is called trainData.
> data(trainData)
> ## The survival time vector for the training set is called trainSurv, where survival times
> data(trainSurv)
> ## The censor vector for the training set is called trainCens, where 0 = censored and 1 = u
> data(trainCens)
```

The function `iterateBMAsurv.train` selects relevant variables by iteratively applying the `bic.surv` function from the `BMA` package until all variables are exhausted. The function `iterateBMAsurv.train.wrapper` acts as a wrapper for `iterateBMAsurv.train`, initializing the `bic.surv` parameters and calling `iterateBMAsurv.train` to launch the `bic.surv` iterations.

When `iterateBMAsurv.train.wrapper` is called, the data is assumed to be pre-sorted by rank and assumed to contain the desired number of variables. In the training phase, only the sorted training dataset and the corresponding survival times and censor data are required as input.

```
> ## Training phase: select relevant genes
> ## In this example training set, the top 100 genes have already been sorted in decreasing o
> ret.list <- iterateBMAsurv.train.wrapper (x=trainData, surv.time=trainSurv, cens.vec=trainC
```

```
17: Explored up to variable # 100
Iterate bic.surv is done!
Selected genes:
 [1] "X31687" "X33840" "X31242" "X16948" "X31471" "X17154" "X28531" "X19241"
 [9] "X26146" "X17804" "X27332" "X17241" "X32212" "X29911" "X33558" "X33013"
[17] "X27884" "X33706" "X16817" "X31968" "X30209" "X29650" "X25054" "X16988"
[25] "X32904"
Posterior probabilities of selected genes:
 [1] 100.0  47.5  47.3   2.4  38.5  28.5  40.1  96.7   2.8   1.7   0.0  59.9
[13]   0.0   0.0  10.0   0.0   2.5  58.3   2.1  98.8  28.4   7.1  95.1   0.0
[25] 100.0
```

```
> ## Extract the {\tt bic.surv} object
> ret.bic.surv <- ret.list$obj
> ## Extract the names of the genes from the last iteration of {\tt bic.surv}
> gene.names <- ret.list$curr.names
> ## Get the selected genes with probne0 > 0
> top.gene.names <- gene.names[ret.bic.surv$probne0 > 0]
> top.gene.names
```

```
 [1] "X31687" "X33840" "X31242" "X16948" "X31471" "X17154" "X28531" "X19241"
 [9] "X26146" "X17804" "X17241" "X33558" "X27884" "X33706" "X16817" "X31968"
[17] "X30209" "X29650" "X25054" "X32904"
```

```
> ## Get the posterior probabilities for the selected models
> ret.bic.surv$postprob
```

```
 [1] 0.075782322 0.068183539 0.062240254 0.056227073 0.045761712 0.044794588
 [7] 0.043328132 0.042831731 0.039567629 0.039285627 0.038997242 0.034867824
[13] 0.032225236 0.030210326 0.026904418 0.025508701 0.025052995 0.024869256
[19] 0.021711946 0.021061750 0.020689119 0.020114454 0.017345536 0.017179713
[25] 0.017104052 0.015294500 0.014059561 0.014050900 0.012658966 0.010182444
[31] 0.008768581 0.007844758 0.007014883 0.006609877 0.006555310 0.005115046
```

If all the variables are exhausted in the `bic.surv` iterations, the `iterateBMAsurv.train.wrapper` function returns `curr.names`, or a vector containing the names of the variables in the last iteration of `bic.surv`. It also returns an object of class `bic.surv` from the last iteration of `bic.surv`. This object is a list consisting of many components. Here are some of the relevant components:

- `namesx`: The names of the variables in the last iteration of `bic.surv`.
- `postprob`: The posterior probabilities of the models selected. The length of this vector indicates the number of models selected by BMA.
- `which`: A logical matrix with one row per model and one column per variable indicating whether that variable is in the model.
- `probne0`: The posterior probability that each variable is non-zero (in percent) in the last iteration of `bic.surv`. The length of this vector should be identical to that of `curr.mat`.
- `mle`: Matrix with one row per model and one column per variable giving the maximum likelihood estimate of each coefficient for each model.

In the training phase, the relevant variables (genes) are selected using the training data, the survival times, and the censor vector. In the test phase, we call the function `predictBicSurv` with the selected variables (genes), the selected models, and the corresponding posterior probabilities to predict the risk scores for the patient samples in the test set. The predicted risk score of a test sample is equal to the weighted average of the risk score of the test sample under each selected model, multiplied by the predicted posterior probability of each model. Note that in this case, a model consists of a set of genes, and different models can potentially have overlapping genes. The posterior probability of a gene is equal to the sum of the posterior probabilities of all the models that the gene belongs to. Finally, the function `predictiveAssessCategory` assigns each test sample to a risk group (either high-risk or low-risk) based on the predicted risk score of the sample.

```
> ## The test data matrix is called testData.
> data(testData)
> ## The survival time vector for the test set is called testSurv, where survival times are r
> data(testSurv)
> ## The censor vector for the test set is called testCens, where 0 = censored and 1 = uncens
> data(testCens)
> ## Get the subset of test data with the genes from the last iteration of bic.surv
> curr.test.dat <- testData[, top.gene.names]
> ## Compute the predicted risk scores for the test samples
> y.pred.test <- apply (curr.test.dat, 1, predictBicSurv, postprob.vec=ret.bic.surv$postprob,
> ## Compute the risk scores for the training samples
> y.pred.train <- apply (trainData[, top.gene.names], 1, predictBicSurv, postprob.vec=ret.bic
> ## Assign risk categories for test samples
> ## Argument {\tt cutPoint} is the percentage cutoff for separating the high-risk group from
> ret.table <- predictiveAssessCategory (y.pred.test, y.pred.train, testCens, cutPoint=50)
> ## Extract risk group vector and risk group table
> risk.vector <- ret.table$groups
> risk.table <- ret.table$assign.risk
> risk.table

           cens.vec.test
             0  1
```

```
  High Risk  8 10
  Low Risk   7 11

> ## Create a survival object from the test set
> mySurv.obj <- Surv(testSurv, testCens)
> ## Extract statistics including p-value, chi-square, and variance matrix
> stats <- survdiff(mySurv.obj ~ unlist(risk.vector))
> stats

Call:
survdiff(formula = mySurv.obj ~ unlist(risk.vector))

                                N Observed Expected (O-E)^2/E (O-E)^2/V
unlist(risk.vector)=High Risk 18       10     10.5    0.0267    0.0538
unlist(risk.vector)=Low Risk  18       11     10.5    0.0268    0.0538

 Chisq= 0.1  on 1 degrees of freedom, p= 0.817
```

The p-value is calculated using the central chi-square distribution.

The function `iterateBMAsurv.train.predict.assess` combines the training, prediction, and test phases. The function begins by calling `singleGeneCoxph`, which sorts the genes in descending order of their log likelihood. After calling `iterateBMAsurv.train.wrapper` to conduct the `bic.surv` iterations, the algorithm predicts the risk scores for the test samples and assigns them to a risk group. Predictive accuracy is evaluated through the p-value and chi-square statistic, along with a Kaplan-Meier survival analysis curve to serve as a pictorial nonparametric estimator of the difference between risk groups. If the Cox Proportional Hazards Model is the desired univariate ranking measure, then calling the function `iterateBMAsurv.train.predict.assess` is all that is necessary for a complete survival analysis run. The parameter p represents the number of top univariate sorted genes to be used in the iterative calls to the `bic.surv` algorithm. Our studies showed that a relatively large p typically yields good results (**?**). For simplicity, there are 100 genes in the sample training set, and we used p= 100 in the iterative BMA algorithm for survival analysis. Experimenting with greater p values, higher numbers of *nbest* models, and different percentage cutoffs for *cutPoint* will likely yield more significant results than the following examples illustrate. The function returns a list consisting of the following components:

- `nvar`: The number of variables selected by the last iteration of `bic.surv`.
- `nmodel`: The number of models selected by the last iteration of `bic.surv`.
- `ypred`: The predicted risk scores on the test samples.
- `result.table`: A 2 x 2 table indicating the number of test samples in each category (high-risk/censored, high-risk/uncensored, low-risk/censored, low-risk/uncensored).
- `statistics`: An object of class `survdiff` that contains the statistics from survival analysis, including the variance matrix, chi-square statistic, and p-value.

- success: A boolean variable returned as TRUE if both risk groups are present in the patient test samples.

If all test samples are assigned to a single risk group or all samples are in the same censor category, a boolean variable success is returned as FALSE.

```
> ## Use p=10 genes and nbest=5 for fast computation
> ret.bma <- iterateBMAsurv.train.predict.assess (train.dat=trainData, test.dat=testData, sur

1: Explored up to variable # 10
Iterate bic.surv is done!
Selected genes:
 [1] "X33310" "X28197" "X19373" "X16527" "X27415" "X24394" "X28531" "X27585"
 [9] "X27766" "X26940"
Posterior probabilities of selected genes:
 [1] 100.0  18.9  16.9   0.0  19.6   0.0  47.0  12.6   0.0   4.3
# selected genes = 7
# selected models = 11
Risk Table:
           cens.vec.test
             0  1
  High Risk  7  9
  Low Risk   8 12
Call:
survdiff(formula = mySurv.obj ~ unlist(risk.groups))

                               N Observed Expected (O-E)^2/E (O-E)^2/V
unlist(risk.groups)=High Risk 16        9     8.21    0.0752     0.126
unlist(risk.groups)=Low Risk  20       12    12.79    0.0483     0.126

 Chisq= 0.1  on 1 degrees of freedom, p= 0.722
          [,1]      [,2]
[1,]  4.891446 -4.891446
[2,] -4.891446  4.891446
[1] 0.1262768

> ## Extract the statistics from this survival analysis run
> number.genes <- ret.bma$nvar
> number.models <- ret.bma$nmodel
> evaluate.success <- ret.bma$statistics
> evaluate.success

Call:
survdiff(formula = mySurv.obj ~ unlist(risk.groups))

                               N Observed Expected (O-E)^2/E (O-E)^2/V
```

```
unlist(risk.groups)=High Risk 16        9     8.21    0.0752     0.126
unlist(risk.groups)=Low Risk  20       12    12.79    0.0483     0.126

 Chisq= 0.1  on 1 degrees of freedom, p= 0.722
```

The function `crossVal` performs k runs of n-fold cross validation on the training set, where k and n are specified by the user through the `noRuns` and `noFolds` arguments respectively. The `crossVal` function in this package can be used to evaluate the selected mathematical models and determine the optimal input parameters for a given dataset. For each run of cross validation, the training set, survival times, and censor data are re-ordered according to a random permutation. For each fold of cross validation, 1/nth of the data is set aside to act as the validation set. In each fold, the `iterateBMAsurv.train.predict.assess` function is called in order to carry out a complete run of survival analysis. This means the univariate ranking measure for this cross validation function is Cox Proportional Hazards Regression; see `iterateBMAsurv.train.wrapper` to experiment with alternate univariate ranking methods. With each run of cross validation, the survival analysis statistics are saved and written to file. The output of this function is a series of files written to the working R directory which give the fold results, run results, per-fold statistics, and average statistics across all runs and all folds.

```
> ## Perform 1 run of 2-fold cross validation on the training set, using p=10 genes and nbest
> ## Argument {\tt diseaseType} specifies the type of disease present the training samples, u
> cv <- crossVal (exset=trainData, survTime=trainSurv, censor=trainCens, diseaseType="DLBCL",

******************** BEGINNING CV RUN 1 ********************
---------- BEGINNING FOLD 1 ----------
1: Explored up to variable # 10
Iterate bic.surv is done!
Selected genes:
 [1] "X27766" "X19373" "X27415" "X17280" "X27587" "X27585" "X27573" "X16988"
 [9] "X27764" "X28531"
Posterior probabilities of selected genes:
 [1] 65.3 17.5 22.5  3.8  8.7  9.0  8.7  4.6 15.4  0.0
# selected genes = 9
# selected models = 15
Risk Table:
           cens.vec.test
             0  1
  High Risk  2  6
  Low Risk  10 14
Call:
survdiff(formula = mySurv.obj ~ unlist(risk.groups))

                             N Observed Expected (O-E)^2/E (O-E)^2/V
unlist(risk.groups)=High Risk  8        6     3.66     1.501      1.93
unlist(risk.groups)=Low Risk  24       14    16.34     0.336      1.93
```

```
 Chisq= 1.9  on 1 degrees of freedom, p= 0.165
          [,1]      [,2]
[1,]  2.843887 -2.843887
[2,] -2.843887  2.843887
[1] 1.929997
---------- BEGINNING FOLD 2 ----------
1: Explored up to variable # 10
Iterate bic.surv is done!
Selected genes:
 [1] "X33850" "X31453" "X34500" "X31806" "X25054" "X32042" "X19241" "X34574"
 [9] "X26627" "X26922"
Posterior probabilities of selected genes:
 [1] 45.4 20.4  0.0 55.4 50.3  0.0 79.0 38.3  5.1  0.0
# selected genes = 7
# selected models = 21
Risk Table:
           cens.vec.test
             0  1
  High Risk 13  9
  Low Risk   5  6
Call:
survdiff(formula = mySurv.obj ~ unlist(risk.groups))

                               N Observed Expected (O-E)^2/E (O-E)^2/V
unlist(risk.groups)=High Risk 22        9    10.72     0.275      1.01
unlist(risk.groups)=Low Risk  11        6     4.28     0.687      1.01

 Chisq= 1  on 1 degrees of freedom, p= 0.315
          [,1]      [,2]
[1,]  2.916762 -2.916762
[2,] -2.916762  2.916762
[1] 1.008635
******************** END CV RUN 1 ********************
Overall results from this run:
          Censored Uncensored Percent Uncensored
Low Risk        15         20           57.14286
High Risk       15         15           50.00000
Overall average result matrix over all runs:
          Censored Uncensored Percent Uncensored
Low Risk        15         20           57.14286
High Risk       15         15           50.00000
Average p-value across all folds and all runs:
[1] 0.2399944
Standard deviation of p-values across all folds and all runs:
[1] 0.1063993
```

```
Average chi-square value across all folds and all runs:
[1] 1.469316
Standard deviation of chi-square across all folds and all runs:
[1] 0.6515016
```

This package also contains the `imageplot.iterate.bma.surv` function, which allows for the creation of a heatmap-style image to visualize the selected genes and models (see Figure **??**).

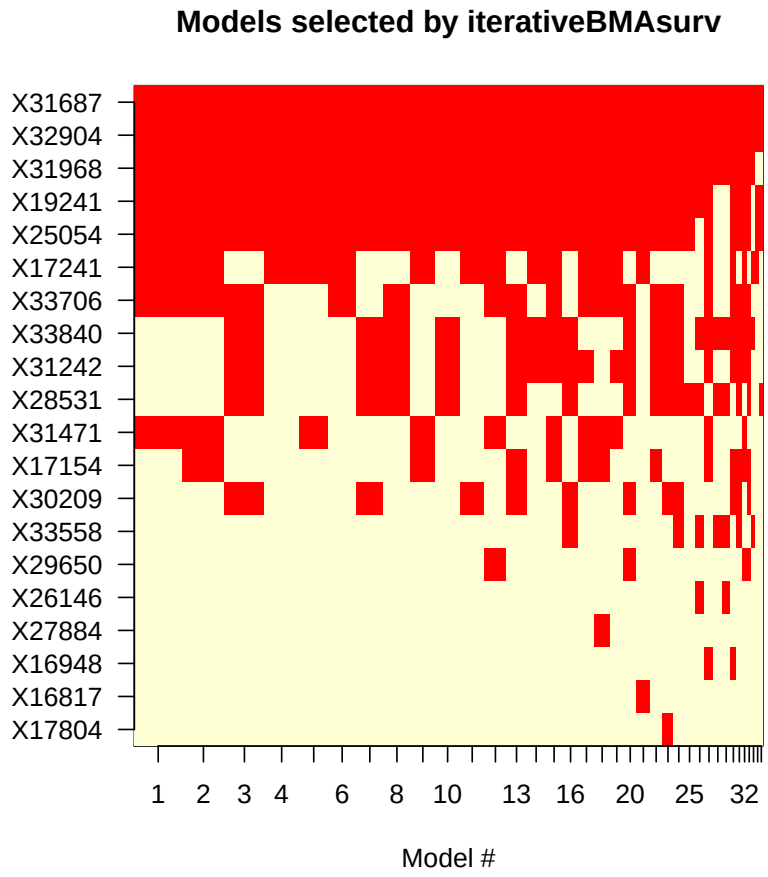

Figure 1: An image plot showing the selected genes and models.

In Figure **??**, the BMA selected variables are shown on the vertical axis, and the BMA selected models are shown on the horizontal axis. The variables (genes) are sorted in decreasing order of the posterior probability that the variable is not equal to 0 (`probne0`) from top to bottom. The models are sorted in decreasing order of the model posterior probability (`postprob`) from left to right.

Acknowledgements

We would like to thank Isabelle Bichindaritz, Donald Chinn, Steve Hanks, Ian Painter, Deanna Petrochilos, and Chris Volinsky. Yeung is supported by NIH-NCI 1K25CA106988. Bumgarner is funded by NIH-NHLBI P50 HL073996, NIH-NIAID U54 AI057141, NIH-NCRR R24 RR021863-01A1, NIH-NIDCR R01 DE012212-06, NIH-NCRR 1 UL1 RR 025014-01, and a generous basic research grant from Merck. Raftery is supported by NIH-NICHD 1R01HDO54511-01A1, NSF llS0534094, NSF ATM0724721, and Office of Naval Research grant N00014-01-1-0745.