Iterative Bayesian Model Averaging: A Method for the Application of Survival Analysis to High-Dimensional Microarray Data

Amalia Annest

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2008

Program Authorized to Offer Degree:
Institute of Technology - Tacoma

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Amalia Annest

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Isabelle Bichindaritz

_____
Donald Chinn

Date:_____

In presenting this thesis in partial fulfillment of the requirements for a master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature_____

Date_____

University of Washington

Abstract

Iterative Bayesian Model Averaging: A Method for the Application of Survival Analysis to High-Dimensional Microarray Data

Amalia Annest

Chair of the Supervisory Committee:
Professor Isabelle Bichindaritz
Computing and Software Systems

The traditional cancer prognostic tools of tumor stage and morphology are inadequate benchmarks for the accurate determination of patient risk. The emergence of microarray technology has enabled the simultaneous measurement of thousands of gene expression levels, allowing researchers to apply sophisticated data mining and statistical techniques in the search for a superior prognostic methodology. This paper extends an existing procedure called iterative Bayesian Model Averaging (BMA) for application to survival analysis. Iterative BMA is a method for predicting survival prognosis by isolating a small group of relevant predictor genes from a high-dimensional microarray dataset. In this project, the iterative BMA algorithm for survival analysis is applied to two real cancer datasets: diffuse large B-cell lymphoma and breast cancer. The selected genes are used to divide patients into high- and low-risk categories, and the p-values and Kaplan-Meier survival curves representing the difference between risk groups are provided as measures of predictive accuracy. Results show that the iterative BMA algorithm for survival analysis consistently selects a small number of relevant genes while providing a higher degree of predictive accuracy than other feature selection methods. The procedure shows promise as a powerful and cost-effective prognostic tool in future cancer research.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author wishes to express her sincere appreciation to the Institute of Technology at the University of Washington, Tacoma, and a double dose of gratitude goes out to committee chair Dr. Isabelle Bichindaritz and committee member Dr. Donald Chinn for their tireless guidance and input. Thanks also to Dr. Steven Hanks for his assistance in getting this project approved as a thesis. In addition, this project would not have been possible without the collaboration of the Bumgarner Lab at the University of Washington, Seattle. Thank you to Roger Bumgarner and Adrian Raftery for providing insightful comments during the various iterations of the paper. An extra special thanks goes out to Ka Yee Yeung, whose patience and infallible guidance were invaluable during every step of this endeavor.

INTRODUCTION

In the fight against cancer, oncologic researchers are constantly striving for a more accurate method by which to assess a patient's chances for survival. The relatively recent emergence of microarray technology has allowed for new advances in the determination of survival prognosis through the study of gene expression levels. This paper presents a survival analysis extension to the iterative Bayesian Model Averaging (BMA) algorithm (Yeung et al., 2005), which is a method for predicting survival prognosis by isolating a small group of relevant predictor genes from a high-dimensional microarray dataset. The purpose of this project is to show that the iterative BMA algorithm for survival analysis consistently selects a small and cost-effective number of predictor genes while outperforming other feature selection and supervised learning procedures in the accuracy of patient risk assessment.

Until recently, oncologists relied primarily on tumor stage and morphology to help outline an appropriate course of treatment for their cancer patients. Malignant tumors were generally resected in operable cases, and follow-up radiation therapy was provided to victims exhibiting advanced-stage diseases. This methodology proved problematic in that a number of low-risk patients experienced cancer recurrence or death within a short time frame, while a contingent of high-risk patients went into permanent remission despite the bleak nature of their original prognoses. This trend indicated a need to explore other benchmarks by which doctors could understand the underlying prognosis of a given disease and decide on a treatment plan that would optimize the patient's chances for survival.

Microarray technology provides a promising avenue. The availability of thousands of gene expression levels has enabled the pursuit of a new direction in cancer research. In particular, gene expression patterns can be thought of as multidimensional quantitative "expression phenotypes" which can in turn be

correlated with clinical outcome. Because a single microarray can measure the expression levels of tens of thousands of genes simultaneously, the challenge lies in the development of data mining methods and tools to extract biological meaning from this immense amount of data. More specifically, the aim is to filter the expression dataset down to the smallest possible subset of accurate predictor genes. Reducing the number of predictor genes both decreases clinical costs and mitigates the possibility of overfitting due to high inter-variable correlations (Jiangeng et al., 2007).

The most common approach to identifying a manageable group of predictor genes is called feature selection, in which a subset of relevant “features” (or variables) is selected from the larger dataset in order to produce a robust learning model (Liu & Motoda, 1998; Liu & Motoda, 2008). A well-designed feature selection algorithm will choose a small set of variables that is highly predictive of clinical outcome. Univariate feature selection methods evaluate the usefulness of each variable on an individual basis. Examples of univariate techniques include the t-test (Nguyen & Rocke, 2002), the signal-to-noise ratio (Golub et al., 1999), the Cox proportional hazards model (Cox, 1972), threshold number of misclassification (TNoM) score (Ben-Dor et al., 2000), the between-groups to within-groups sum of squares (BSS/WSS) ratio (Dudoit et al., 2002), and mean aggregate relevance (Chow et al., 2001). Multivariate methods are more sophisticated in that they perform combinatorial searches within the feature subspace to evaluate the effectiveness of groups of genes. Examples include Recursive Feature Elimination (RFE) (Guyon et al., 2002), genetic algorithms (Li et al., 2001; Silva et al., 2005; Yu et al., 2007), floating search (Pudil et al., 1994), and top-scoring pair methods (Geman et al., 2004; Xu et al., 2005). Despite some evidence to the contrary (e.g., Lai et al., 2006), multivariate selection algorithms are generally preferable to univariate because they cut down on inter-variable dependencies and often lead to models with fewer predictive variables (Chen et al., 2003; Yeung et al., 2005).

Subsequent to or concurrent with the feature selection process, a supervised machine learning technique can be applied to generate a predictive function using the selected variables from a set of training data (Huang et al., 2006; Witten & Frank, 2005). In a supervised learning algorithm, the input is a set of training samples paired with the corresponding labels of those samples. The labels can be any predictable quantity of interest, such as a tumor subtype or a length of survival time. If the labels are exhaustive discrete classes to which the samples belong (e.g. “survived beyond five years” and “died before five years”), then the learning model is a *classifier* (for a review of classification techniques in supervised machine learning, see Kotsiantis, 2007). With microarray data, the most common approach is to apply a classification algorithm in which the patient data is split into subcategories corresponding to different prognoses or diagnoses. The algorithm then identifies a subset of genes from which a mathematical model is built, and the model produces an output that correlates with prognosis or diagnosis. In general, the subcategories are static and based on thresholds associated with some clinical variable (e.g., time to metastases). Classification studies on microarray data have used gene expression levels to distinguish diseased tissue samples from normal ones (Xu et al., 2007; Jiang et al., 2004), identify cancer subtypes (Golub et al., 1999; Tan et al., 2005), and assign discrete risk groups for survival prognosis (Sotiriou et al., 2003; van’t Veer et al., 2002, Yu et al., 2007; Yeung et al., 2005; Raponi et al., 2006). See Hu et al. (2006) for a comparative analysis of classification methods on microarray data.

In cancer research, gene expression data is often reported in tandem with time to event information (such as time to metastasis, death, or relapse). In order to take advantage of these continuous clinical variables under a supervised framework, survival analysis can be applied. Survival analysis on microarray data differs from classification in that the sample labels are continuous rather than discrete. The overall goal in survival analysis research is to create the

strongest predictive model of patient survival, and the most important components of this process are feature selection and model construction. In the context of survival analysis, a *model* refers to a set of selected genes whose regression coefficients have been calculated for use in predicting survival prognosis (Volinsky et al., 1997). In the application of survival analysis to high-dimensional microarray data, a feature selection algorithm identifies this subset of genes from the gene expression training dataset. These genes are then used to build a mathematical model that evaluates the continuous time to event data (Hosmer et al., 2008). The choice of feature selection algorithm determines which genes are chosen and the number of predictor genes deemed to be relevant, whereas the choice of mathematical framework used in model construction dictates the ultimate success of the model in predicting the time to event on an independent validation dataset. See Figure (1) for a process-flow diagram delineating the application of feature selection and model construction to survival analysis on microarray data.

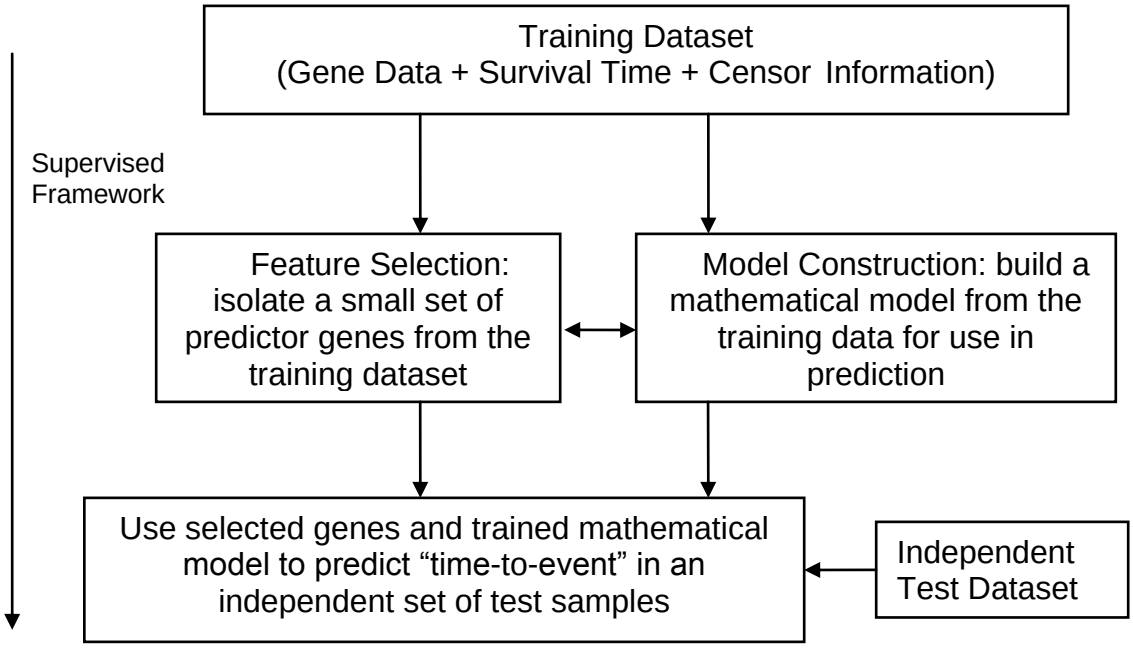

Figure (1). Process-flow diagram illustrating the application of feature selection and model construction to survival analysis on microarray data.

The problem with most feature selection algorithms used to produce continuous predictors of patient survival is that they fail to account for model uncertainty. With thousands of genes and only tens to hundreds of samples, there is a relatively high likelihood that a number of different models could describe the data with equal predictive power. In this paper, the Bayesian Model Averaging (BMA) methods (Raftery, 1995; Hoeting et al., 1999) are applied to select a subset of genes for survival analysis on microarray data. Instead of choosing a single model and proceeding as if the data was actually generated from it, BMA combines the effectiveness of multiple models by taking the weighted average of their posterior distributions. In addition, BMA consistently identifies a small number of predictive genes (Volinsky et al., 1997; Yeung et al., 2005), and the posterior probabilities of the selected genes and models are available to facilitate an easily interpretable summary of the output. Yeung et al. (2005) extended the applicability of the traditional BMA algorithm to high-dimensional microarray data by embedding the BMA framework within an iterative approach. Here their iterative BMA method is further extended to survival analysis. In particular, the iterative BMA method for survival analysis has been developed and implemented as a Bioconductor package called *iterativeBMAsurv*, and the algorithm is demonstrated on two real cancer datasets. The results reveal that iterative BMA consistently selects a small number of predictor genes while providing greater predictive accuracy than other algorithms, and the models themselves are simple and highly amenable to biological interpretation.

BACKGROUND

The science of biology has proven to be one of the most important areas of knowledge in human history. The study of biology has led to such discoveries as cell differentiation, the Krebs cycle, and the complete mapping of the human genome. Medical breakthroughs are a natural result of these biological advances, but other disciplines are necessary for the full realization of the

potential benefits to human health. The close relationship between chemistry and biology yielded the field of biochemistry, and the need to describe biological processes on an atomic scale led to the emergence of biophysics. A tremendous amount of data is required to explore biological phenomena, and computer science techniques have become pivotal in managing this mountain of information. Bioinformatics is the merger of biology and computer science (Cohen, 2004).

Bioinformatics spans the complete range of biological topics, but the study of gene expression is one of the most prominent and important subjects to pursue in the quest for medical advances. Gene expression is defined as the process by which a gene's DNA sequence is converted into a functional protein. The DNA sequence is first encoded as messenger RNA (mRNA) through the process of transcription, and the mRNA is subsequently translated into a protein (Piatetsky-Shapiro & Tamayo, 2003). A higher expression score denotes greater amounts of gene activity. Gene expression profiles are created and studied to determine which genes are expressed in particular cell types, at what times these genes are expressed, and under what conditions expression occurs. Medical researchers often use these profiles to compare the expression levels of normal cells with those of cells in a special condition. This condition could involve cancer, other diseases, starvation, extreme temperature, or any other unique cellular state that scientists are interested in understanding. By studying the differences between the expression levels of these cells, researchers are able to determine the effects of the given condition on the process of gene expression in a particular cell type.

Gene expression profiles are most commonly extracted through the use of microarray technologies. A single silicon microarray chip can measure the expression levels of tens of thousands of genes at once, and this number changes rapidly as the technology grows more robust. To put this in

perspective, the human genome is believed to contain around 20,000 to 25,000 genes, so one microarray chip could ostensibly produce expression scores for more genes than exist in the entire human genome (Piatetsky-Shapiro & Tamayo, 2003). Microarrays come in several different types, including short oligonucleotide arrays, cDNA or spotted arrays, long oligonucleotide arrays, and fiber-optic arrays. Short oligonucleotide arrays, manufactured by the company Affymetrix, are the most popular commercial variety on the market today (Piatetsky-Shapiro et al., 2003). See Figure (2) for a pictorial representation of microarray data from van't Veer et al. (2002).

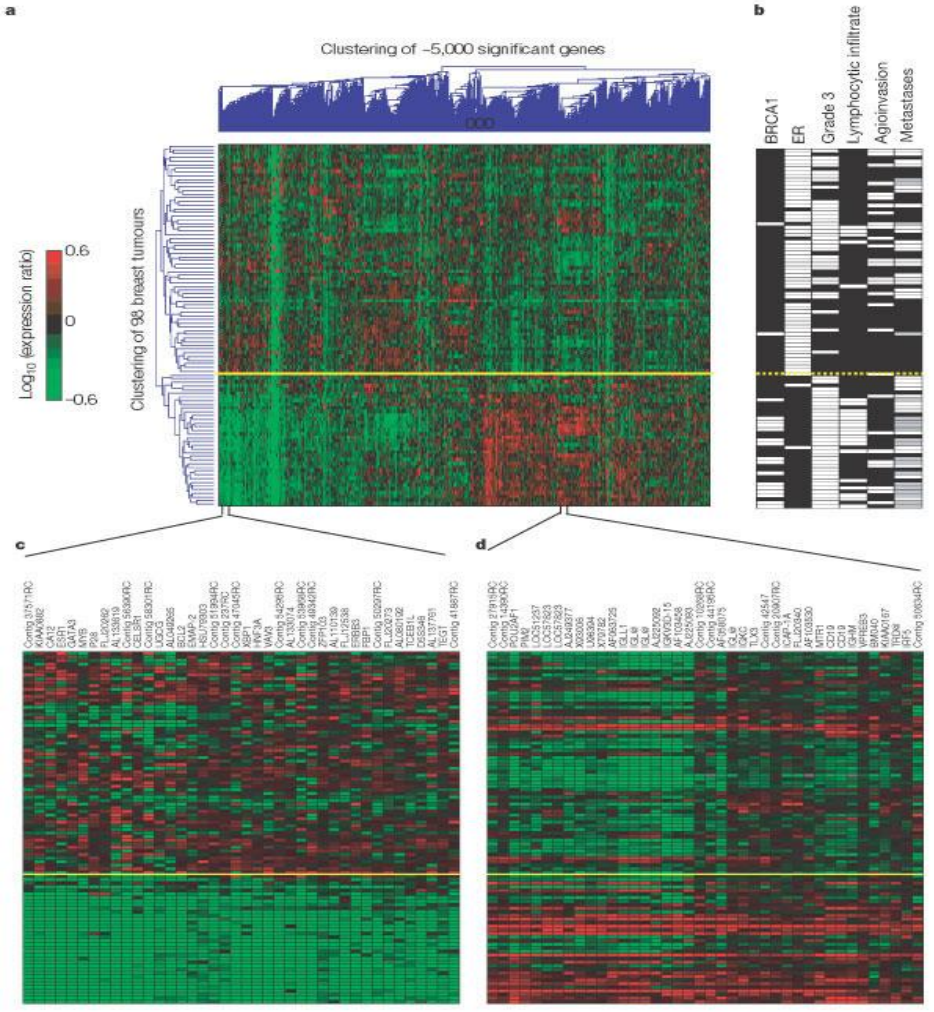

Figure (2). A heatmap of microarray data from van't Veer et al. (2002).

The conduction of a microarray experiment consists of three distinct phases. In the first, the scientist places thousands of single-strand RNA pieces into tiny wells on the surface of the chip. These pieces are just normal, standard-issue RNA strands that are known to be produced by a specific gene in a particular type of cell (the same type of cell that the researchers are interested in studying). Each well has a unique 2D coordinate, so the biologists know exactly which well corresponds to which gene in the cell. In phase two, genetic material from the cell under study is smeared onto the top of the chip. This cell is usually diseased or in some special condition (e.g., starvation, extreme temperature) as mentioned previously. The RNA from the two cells will combine on the chip, and scientists can ascertain how much of the RNA from the cell under study is being expressed through an analysis of the combined genetic material. In the last phase, a laser scanner connected to a computer measures the combination results in each individual chip well. The program then outputs a real number that corresponds to the degree of genetic expression for each gene in the cell under study (Cohen, 2004).

Microarray technologies have a number of potential applications. These include disease diagnosis through an analysis of gene expression levels, predicting the outcome of a particular course of treatment, choosing drugs appropriate for a given genetic profile, drug and toxicology studies, and general biological knowledge acquisition (Piatetsky-Shapiro et al., 2003). The downside of microarray analysis is the propensity of the technology to produce “false positives”, and the reason for this problem lies in the structure of the data fields. Microarray datasets typically contain relatively few rows or entries (often fewer than 100 patients), but the number of columns, where each column represents a singe gene, can reach into the tens of thousands. Most other common datasets (e.g., bank or retail customer data) are built to exhibit the opposite scenario; they are comprised of thousands to millions of records with no more than several hundred attributes. In microarray data, the low ratio of patient

entries to genetic expression scores can lead to results that appear to be meaningful when in fact they are simply due to chance. For this reason, especially robust analysis techniques are required to mine gene expression data collected through microarray technologies (Piatetsky-Shapiro & Tamayo, 2003).

One of the most exciting applications of gene expression profiling is the ability to predict prognoses for cancer patients. Recent studies have shown that the intensity with which certain genes are expressed in cancerous tissue is an important predictor of the patient's chances for survival (Beer et al., 2002; Lu et al., 2006; Glinsky et al., 2004; van't Veer et al., 2002; van di Vijver et al., 2002; Bair & Tibshirani, 2004; Raponi et al., 2006; Korkola et al., 2007; Rosenwald et al., 2002). This finding is beneficial to health care because it means that doctors can potentially use gene expression profiles to more accurately determine the best course of treatment for their patients. This area of research has been burgeoning in the past decade, and a number of papers have examined genetic expression in several types of tumors including lung cancer (Beer et al., 2002; Lu et al., 2006; Raponi et al., 2006), breast cancer (van't Veer et al., 2002; van di Vijver et al., 2002; Korkola et al., 2007; Bair & Tibshirani, 2004), prostate cancer (Glinsky et al., 2004), and lymphoma (Rosenwald et al., 2002; Bair & Tibshirani, 2004). Findings indicate that genetic expression levels are significantly predictive of both cancer recurrence and survival prognosis.

PREVIOUS WORK IN SURVIVAL ANALYSIS ON MICROARRAY DATA

A growing number of studies have used a variety of statistical methodologies to perform survival analysis on high-dimensional microarray data. Beer et al. (2002) developed a risk index based on 50 genes that classified lung cancer patients into either low- or high-risk groups with a considerable degree of accuracy. The two groups differed significantly from one another in terms of survival rates (p-value=0.0006), and these results still held among patients with

stage-1 tumors (p-value=0.028). Beer et al. tested their risk index through leave-one-out cross validation on the original data set. The index was then applied to an independent dataset with the same genetic expression information. Again, the high- and low-risk groups were found to differ significantly from one another, both overall (p-value=0.003) and among patients with stage-1 tumors (p-value=0.006).

Lu et al. (2006) applied a similar methodology based on 64 genes, with improved results. The study used multivariate Cox Proportional Hazards regression with bootstrap resampling and forward selection to identify a 64-gene signature, whereas Beer et al. employed univariate Cox regression. In addition, Lu et al. gathered seven independent data sets for validation purposes. Once they created a risk index by training on two of the seven datasets, they divided patients from the other five into high- and low-risk groups. In all cases, the two groups were significantly different from one another ($p < 0.001$).

In 2002, van't Veer et al. published a survival classification study involving primary invasive breast carcinomas. They attempted to classify patients into two groups: those whose cancer recurred within five years of diagnosis and tumor resection (poor prognosis group), and those who remained disease-free beyond five years (good prognosis group). They used a three-step supervised classification method to develop a 70-gene predictive signature, which they applied to an independent test set of 19 patients. Their classifier correctly labeled 17 of the 19 samples, an accuracy of 89.5% (p-value=0.0018).

More recently, Korkola et al. (2007) used a combination of three different feature selection methods to identify a set of predictive genes: Prediction Analysis for Microarrays (PAM), Significance Analysis for Microarrays (SAM), and a correlation-based technique similar to that of van't Veer et al. (2002). Of

the genes selected in each method, 21 were present in all three models. This 21-gene signature was applied to two independent breast cancer datasets and successfully separated patients into good and poor outcome groups ($p < 0.0005$).

Bair and Tibshirani (2004) proposed a semi-supervised version of principal components analysis that is capable of generating a continuous predictor of patient survival. Their algorithm consistently selected fewer than 20 genes and successfully divided patients into high- and low-risk groups in four different datasets: lymphoma (p-value=0.00124), breast cancer (p-value=2.06e-05), lung cancer (p-value=1.47e-07), and acute myeloid leukemia (p-value=0.00136).

METHODS AND MATERIALS

This section presents the cancer datasets selected for use in this study, along with a complete description of Bayesian Model Averaging (Raftery, 1995; Hoeting et al., 1999). A brief synopsis of BMA and survival analysis will be provided, along with the modifications necessary to extend the applicability of BMA to high-dimensional microarray data. Finally, methods of predictive assessment will be explained.

DATA

Lymphoma. The first dataset in this study consists of tumor samples from 240 patients diagnosed with diffuse large B-Cell lymphoma (DLBCL) (Rosenwald et al., 2002). Roughly 60% of DLBCL victims who are treated with chemotherapy do not survive, and the disease comprises a majority 30-40% of all non-Hodgkin lymphomas (NHLCP, 1997; Shipp et al., 2002). This DLBCL dataset was generated and first analyzed by Rosenwald et al. (2002), and the expression profiles from 7399 genes along with corresponding patient information can be downloaded from their supplemental website (http://llmpp.nih.gov/DLBCL/). The raw data were processed with “lymphochip” cDNA microarrays (Alizadeh et al., 1999), which are specialized to include genes that are known to be preferentially expressed within the germinal centers of lymphoid organs. Survival times ranged from 0 to 21.8 years, with a median of 2.8 years across all samples. Of the 240 patients, only 102 (42.5%) were still alive at the final follow-up visit. Rosenwald et al. randomly divided the dataset into 160 training samples and 80 validation samples, and their division has been preserved for this project in order to facilitate a direct comparison of results. See Table (1) for a summary of the DLBCL data.

Breast Cancer. The second dataset consists of patient samples from primary invasive breast carcinomas (van’t Veer et al., 2002; van de Vijver et al., 2002). The breast cancer dataset from van’t Veer et al. was comprised of 78 training

samples and 19 test samples, and van't Veer et al. proposed a 70-gene predictive signature which classified patient samples into good versus poor prognosis groups. Subsequently, van de Vijver et al. acquired a test set of 295 patient samples with clinical data on which to validate the 70-gene predictive signature. Of these 295 patient samples, 61 samples overlapped with the 78 training samples from van't Veer et al. Since different clinical data and survival information were made available from these two publications, the 61 overlapping samples will make up the training set in this project and the remaining 234 samples will comprise the test set. The training and test sets used in this paper are available at the supplemental website for this project: http://expression.washington.edu/ibmasurv/protected (username: amanu, password: ibmasurv). The samples in both breast cancer datasets were hybridized to two-color microarrays containing approximately 25,000 genes. Previously, Yeung et al. (2005) filtered the van't Veer et al. dataset down to 4919 significantly regulated genes (at least a 2-fold difference and p-value < 0.01 in at least three samples), and these 4919 genes have been retained for the analysis in this project. Of the 295 total samples in the training and validation datasets, the survival times ranged from 0.05 to 18.3 years, with a median of 7.2 years. 216 patients (73%) were still alive at the final follow-up visit. See Table (1) for a summary of the breast cancer data.

Table (1). Summary of DLBCL and Breast Cancer Datasets.

Dataset	Total Number of Samples	# Training Samples	# Validation Samples	Number of Genes
DLBCL	240	160	80	7399
Breast Cancer	295	61	234	4919

BAYESIAN MODEL AVERAGING (BMA)

The strength of BMA lies in its ability to account for model uncertainty, an aspect of analysis that is largely ignored by traditional stepwise selection procedures (Raftery, 1995). These traditional methods tend to overestimate the goodness-of-fit between model and data, and the model is subsequently unable to retain its predictive power when applied to independent datasets (Derksen & Keselman, 1992; Volinsky et al., 1997). BMA attempts to solve this problem by selecting a subset of all possible models and making statistical inferences using the weighted average of these models' posterior distributions.

The core of the BMA algorithm is depicted in Equation (1) below (Raftery, 1995). Let Ψ denote the quantity of interest, and let $S = \{M_1, M_2, \ldots, M_n\}$ represent the subset of models selected for inclusion in the analysis. Then the posterior probability of Ψ given the training data *TD* is the weighted average of the posterior probability of Ψ given *TD* and model M_i, multiplied by the posterior probability of model M_i given *TD*. Summing over all the models in set *S*, we get:

$$\Pr(\Psi \mid TD) = \sum_{i \in S} \Pr(\Psi \mid TD, M_i) \bullet \Pr(M_i \mid TD). \qquad (1)$$

There are three issues to consider before Equation (1) can be applied: obtaining the subset *S* of models to be included, estimating the value of $\Pr(\Psi \mid TD, M_i)$, and estimating the value of $\Pr(M_i \mid TD)$. The remainder of this subsection will address these issues.

One challenge with BMA is the sheer number of models that could potentially be explored by the algorithm, especially when dealing with microarray data. If there are *G* candidate explanatory genes in the expression set, then there are 2^G possible models to consider. When working with tens of thousands of genes, such an undertaking is computationally intractable. In order to discard the noncontributory models and obtain a subset that approximates an average over

all 2^G possibilities, Raftery (1995) proposed to use the regression by leaps and bounds algorithm from Furnival and Wilson (1974). This algorithm takes a user-specified input "*nbest"* and efficiently returns the top *nbest* models of each size (maximum 30 variables). Following application of the leaps and bounds algorithm, the Occam's window method of Madigan and Raftery (1994) can be used to reduce the set of models. After identifying the strongest model returned by the leaps and bounds algorithm, the procedure can eliminate any model whose posterior probability is below the cutoff point in relation to the best model. The cutoff point can be varied, but the default is 20; that is, a model must be at least 1/20 as likely as the strongest model in order to be retained. Once this step is complete, the remaining group of models constitutes the set *S* to be used in Equation (1).

An exact calculation of the predictive distribution $\Pr(\Psi \mid TD, M_i)$ requires an integration over the vector of regression parameters θ_i:

$$\Pr(\Psi \mid TD, M_i) = \int \Pr(\Psi \mid \theta_i, TD, M_i) \Pr(\theta_i \mid TD, M_i) \, d\theta_i. \qquad (2)$$

Because this integral has no closed form solution for most censored survival models, the maximum likelihood estimate (MLE) can be used as an approximation:

$$\Pr(\Psi \mid TD, M_i) \approx \Pr(\Psi \mid \hat{\theta}_i, TD, M_i). \qquad (3)$$

While certain techniques such as the Markov Chain Monte Carlo (MCMC) methods have been used in survival analysis to obtain a more exact predictive distribution (Kuo & Smith, 1992), the MLE requires fewer computational resources and has been deemed sufficient for the purpose of averaging over contending models (Draper, 1995; Taplin, 1993; Taplin & Raftery; 1994, Volinsky et al., 1997).

Finally, a calculation of the posterior probability of model M_i given the training data *TD* involves an integral whose solution is impossible to evaluate exactly. Bayes' theorem yields Equation (4), which represents the posterior probability of model M_i given *TD*:

$$\Pr(M_i \mid TD) \propto \Pr(TD \mid M_i) \Pr(M_i), \tag{4}$$

where

$$\Pr(TD \mid M_i) = \int \Pr(TD \mid \theta_i, M_i) \Pr(\theta_i \mid M_i) \, d\theta_i. \tag{5}$$

$\Pr(TD \mid M_i)$ is the integrated likelihood of model M_i, and θ_i is the vector of regression parameters (b_0, b_1, … , b_p) of model M_i. The Bayesian Information Criterion (BIC) first derived by Schwarz (1978) can be used to approximate the integral in equation (5). The Laplace method (Raftery, 1996) is sufficient to accomplish this task when dealing with regular statistical models:

$$\log \Pr(TD \mid M_i) = \log \Pr(TD \mid \hat{\theta}_i, M_i) - (k_i/2) \log n + O(1). \tag{6}$$

In equation (6), *n* represents the number of records in the data, k_i is the number of regression parameters in model M_i, and $O(1)$ is the error term. The Laplace method approximation is generally far more accurate than this final term suggests; see Kass and Wasserman (1995) and Raftery (1996) for discussion. The source code for the BMA implementation described above can be downloaded for both R and S-Plus software at http://www.research.att.com/~volinsky/bma.html.

While this section has focused on the posterior probabilities of the models included in the BMA analysis, it is also beneficial to obtain the posterior probabilities for each of the individual variables (genes) involved. This information is helpful in facilitating biological discussion as it reveals which of the genes are relevant predictors. Let the expression ($b_i \neq 0$) indicate that the regression parameter for gene x_i exists in the vector of regression parameters θ_i for at least one model *M*. In other words, at least one model in the subset *S*

includes gene x_i. Then the posterior probability that gene x_i is a relevant predictor can be written as:

$$\Pr(b_i \neq 0 \mid TD) = \sum_{M_S \text{ where gene i is relevant}} \Pr(M_S \mid TD). \quad (7)$$

In equation (7), M_s refers to the set of all models within the subset *S* that include gene x_i. The posterior probability of gene x_i is a summation of the posterior probabilities of all models in M_s. This computation ensures that all statistically relevant predictor genes will be a part of at least one model in the subset.

BMA FOR SURVIVAL ANALYSIS

To the knowledge of this author, only one previous study has applied the Bayesian Model Averaging methods (Raftery, 1995; Hoeting et al., 1999) to survival analysis. Volinsky et al. (1997) assessed a patient's risk of stroke by using BMA to select variables in Cox Proportional Hazard Models (Cox, 1972). The data was made available by the Cardiovascular Health Study and included 23 variables (e.g., age, smoking history, and blood pressure) that may contribute to a patient's chances of experiencing a stroke. BMA selected a total of 5 models and 11 predictive variables, including diuretic, aspirin use, diabetes, stenosis, and timed walk. Patient risk scores were calculated by taking the weighted average of the risk scores for each of the top five contending models. The patients were then assigned to either the high-risk, medium-risk, or low-risk group based on the empirical 33rd and 66th percentile cutoff points in the risk scores of the training set. To assess performance, Volinsky et al. created an analogue to the log-score called the partial predictive score (PPS). The PPS for BMA was compared against the PPS for the top BMA model (that is, the single model of the top five BMA models with the highest posterior probability) and against the PPS of the model returned by stepwise backward elimination. BMA exhibited the highest PPS, with a prediction mechanism 15% more effective than the top model alone and 3.5% more effective than the stepwise procedure. Furthermore, the patients assigned to a risk group using BMA experienced

fewer strokes in the low-risk group and more strokes in the high-risk group when compared with the other two methods.

EXTENDING BMA FOR HIGH-DIMENSIONAL MICROARRAY DATA

Iterative BMA for Classification. The BMA implementation described above is incompatible with microarray data. This is because the typical microarray dataset contains thousands or even tens of thousands of genes, but the leaps and bounds algorithm from Furnival and Wilson (1974) can only consider a maximum of 30 variables when selecting the top *nbest* models to return to the user. The usual practice of employing stepwise backward elimination to reduce the number of genes down to 30 is not applicable in a situation where the number of predictive variables is greater than the number of samples. Yeung et al. (2005) developed an iterative BMA algorithm that takes a rank-ordered list of genes and successively applies the traditional BMA algorithm until all genes up to a user-specified value p (G_1, G_2, …, G_p) have been processed. The authors begin by using the ratio of between-group to within-group sum of squares (BSS/WSS) (Dudoit et al., 2002) to rank-order the genes from the microarray dataset. As the algorithm iterates, genes with a high posterior probability (equation (7)) are retained while genes with a low posterior probability are eliminated. The default threshold for inclusion is set to 1%; genes whose posterior probabilities are less than 1% are discarded.

Iterative BMA for Survival Analysis. In order to extend the iterative BMA method to survival analysis, a number of algorithmic modifications were implemented. First, instead of applying the BSS/WSS technique (Dudoit et al., 2002) to rank-order the genes in the preprocessing step, the Cox Proportional Hazards Model (Cox, 1972) is used to rank each individual gene. Cox regression is a popular choice in the realm of survival analysis due to its broad applicability and capacity for handling censored data. It is a semi-parametric method that quantifies the hazard rate for a subject *s* at time T as follows:

$$\lambda(T \mid p_s) = \lambda_0(T)\exp(p_s\theta). \quad (8)$$

In this equation, $\lambda_0(T)$ is the baseline hazard function at time T, p_s is the vector of effect parameters (predictors) for subject *s*, and θ is the vector of unknown predictor coefficients. Cox observed that the baseline hazard function in equation (8) could be left unspecified if the effect of a covariate on one individual remains the same for all times T (e.g., if an environmental variable doubles your personal risk of dying at time 5, it also doubles your risk at time 8). Therefore, an estimation of θ is all that is needed. This approximation can be calculated using the partial likelihood:

$$PL(\theta) = \prod_{s=1}^{n}\left(\frac{\exp(p_s\theta)}{\sum_{\ell\in R_s}\exp(p_\ell\theta)}\right)^{\delta_i}. \quad (9)$$

In equation (9), R_s is the risk set at time t_s (where the risk set consists of individuals who have not yet experienced the event of interest). Once the regression parameters are estimated using the Cox model, the genes can be ranked in descending order of their log likelihood.

Following this step, the algorithm iterates through the user-specified *p* top-ranked genes, applying the traditional BMA algorithm for survival analysis (Volinsky et al., 1997) to each group of variables in the current BMA window (where the window size is denoted by *maxNvar*). This part of the procedure is similar to the classification method described previously; genes with high posterior probabilities are retained while genes with low posterior probabilities are eliminated. Following Yeung et al. (2005), the 1% default threshold for inclusion has been adopted for this project. The algorithm relies on the elimination of at least one gene per iteration from the current BMA window, so the method cannot proceed if all genes in the window have a posterior probability ≥ 1%. Yeung et al. proposed an “adapted threshold” heuristic to account for this possibility, whereby the genes with the lowest posterior

probabilities are removed to make room for subsequent variables. This heuristic has been incorporated into the iterative BMA method for survival analysis because the authors reported that its inclusion boosts predictive accuracy. See Figure (3) for an outline of the iterative BMA algorithm for survival analysis developed for this study.

Input: training set *TD* with *G* genes and *n* samples
Pre-processing step: Rank-order all *G* genes by applying Cox Proportional Hazards Regression to each individual gene. Let $x_1, x_2, \ldots, x_G$ be the ordered list of genes, sorted in descending order of log likelihood. Let *maxNvar* denote the user-specified size of the BMA window (maximum 30).
Parameters: *nbest* and *p*, where *p* is the total number of genes to be processed by the iterative BMA algorithm and *G* is the total number of genes in the training dataset. Note that *maxNvar* < *p* ≤ *G*.

1. Initially, start with the *maxNvar* top ranked genes ($x_1, x_2, \ldots, x_{maxNvar}$), and apply the traditional BMA algorithm for survival analysis (Volinsky et al., 1997). Let *toBeProcessed* be an ordered list of genes with ranks (*maxNvar* + 1) to *p*. Initially, *toBeProcessed* ← ($x_{maxNvar+1}, x_{32}, \ldots, x_p$).
2. Repeat until all *p* genes are processed
    a. Remove all genes *i* with $Pr(b_i \neq 0 \mid TD) < 1\%$.
    b. *Adaptive threshold step*: If all genes have $Pr(b_i \neq 0 \mid TD) \geq 1\%$, determine the minimum $Pr(b_i \neq 0 \mid TD)$, *minProbne0*, among the *maxNvar* genes in the current BMA window. Remove all genes with $Pr(b_i \neq 0 \mid TD) <$ (*minProbne0* + 1)%.
    c. Let *removedGenes* be the set of genes removed, and suppose *q* genes are removed.
    d. Replace the *q* removed genes with the *q-next-up* genes from *toBeProcessed*. Update *toBeProcessed* ← *toBeProcessed* – *q-next-up*.
    e. Apply the traditional BMA algorithm for survival analysis.

Output: selected models and their posterior probabilities, selected genes and their corresponding posterior probabilities ($Pr(b_i \neq 0 \mid TD)$), maximum-likelihood estimates of the regression parameters in each model.

Figure (3). Outline of the iterative BMA algorithm for survival analysis on microarray data.

ASSESSMENT

To evaluate the performance of this method, the continuous risk scores of the patients were discretized into risk groups. The overall risk score for a single patient is the weighted average of the risk scores calculated for each model M_i in the set *S* of contending models. The equation is as follows (Volinsky et al, 1997):

$$\sum_{i \in S} ({x_j}^{v} \hat{\theta}_i) \Pr(M_i \mid TD) \qquad (10)$$

In equation (10), $\hat{\theta}_i$ represents the vector of regression parameters for model M_i, and ${x_j}^{v}$ refers to the expression score of each gene x_j within model M_i for a patient in the validation dataset. Therefore, the risk score is computed by multiplying the expression scores of all genes included in model M_i by their corresponding predictor coefficients, adding these $x_j b_j$ terms together, weighing this number by the posterior probability of each model M_i, and summing over all contending models in the set *S*. Note that the predictor coefficients and the model posterior probabilities are all determined from the training data. The implementation employs the user-specified "*cutPoint*" for defining high- versus low-risk groups (e.g., a *cutPoint* of 60 means the lower 60% of scores will be deemed low-risk, and the upper 40% will comprise the high-risk group) using the risk scores of patients in the training data.

The Kaplan-Meier survival curves (Kaplan & Meier, 1958), in which the proportions of surviving patients in each risk group are plotted against successive time intervals, are used to illustrate the results. An advantage of the Kaplan-Meier curve is that it takes censored data into consideration: small vertical tick-marks represent losses where patient data were censored. In addition, predictive performance was measured with the p-value calculated from the log-rank test using the central chi-square distribution. The log-rank test

calculates a p-value testing the null hypothesis that the survival curves from the high- and low-risk groups are identical. Therefore, a significant p-value means that the two risk groups are highly distinct.

RESULTS

This section presents the results from the application of the iterative BMA algorithm for survival analysis to the DLBCL and breast cancer datasets. The p-values and chi-square statistics are reported in each case, and the Kaplan-Meier survival analysis curves are provided as pictorial nonparametric estimators of the difference between risk groups.

DLBCL DATA

The main user-specified parameters to the iterative BMA algorithm for survival analysis include the number of top-ranked *p* genes to be included in the iterations, the *nbest* strongest models to be returned by the leaps and bounds algorithm from Furnival and Wilson (1974), the desired *cutPoint* for separating high- from low-risk patient samples, and the size of the active BMA window (*maxNvar*). In order to determine the best combination of these input parameters, a series of 10-fold cross validation runs were performed on the DLBCL training dataset. Preliminary analyses showed a *cutPoint* of 60 yielded better results than either 40 or 50 (data not shown), and furthermore, a threshold of 60% has precedence in the literature (e.g., Beer et al., 2002). As noted previously, the leaps and bounds algorithm from Furnival and Wilson (1974) becomes inefficient for BMA windows larger than 30 variables. On training sets with relatively small numbers of samples, *maxNvar* may need to be reduced below the 30-variable limit in order to avoid convergence errors caused by matrix singularity and instability in fitting the data. For this reason, a conservative default value of 25 has been chosen for *maxNvar*. A window size of 25 provides a good balance between approximating the maximum and avoiding convergence errors. The initial cross validation runs also showed that $p < 500$ performed poorly, while $p > 1000$ did not add significant predictive value beyond that of the first 1000 genes. Table (2) presents the results from 10 runs of 10-fold cross validation with *nbest*=10, 20, 50, and 100 for both *p*=500 and *p*=1000 genes on the DLBCL dataset. The means and standard deviations

of the p-values and chi-square statistics are calculated across all folds and all runs for each line in the table.

Table (2). 10-run/10-fold cross validation results on the DLBCL dataset for *cutPoint*=60 and *maxNvar*=25.

p (# genes)	*nbest*	Average p-value	p-value stdev	Average chi-square value	chi-square stdev
500	10	0.385	0.308	2.048	2.831
500	20	0.398	0.313	1.853	2.349
500	50	0.329	0.291	2.211	2.842
500	100	0.320	0.294	2.414	2.735
1000	10	0.313	0.303	2.648	3.139
1000	20	0.369	0.308	2.107	2.588
1000	50	0.307	0.303	2.958	3.251
1000	100	0.310	0.271	2.493	3.040

The outcome of the 10-fold cross validation runs on the DLBCL dataset suggested that the strongest results would be obtained from using the input parameters *p*=1000 and *nbest*=50. This conclusion is based primarily on the p-value average. The p-value indicates the probability that a result completely due to chance would be at least as extreme as the observed result. In other words, a p-value of 0.05 (the standard “significance” cutoff point) means that there is a 5% probability that the exact same or better results would be obtained if the iterative BMA procedure had no predictive effect whatsoever. Therefore, the claim that an algorithm can successfully predict survival prognosis grows stronger as the p-value gets smaller. Since the cross validation runs with *p*=1000 and *nbest*=50 yielded the lowest p-value, these parameters were chosen for survival analysis (since the p-value is calculated using the central

chi-square distribution, a lower p-value should be accompanied by a higher chi-square statistic). With a *cutPoint* of 60 and the default *maxNvar* value of 25, the iterative BMA algorithm for survival analysis selected a total of 25 predictive genes contained within 3 contending models. The models were comprised of 24, 23, and 25 genes respectively. Of the 25 genes selected, 23 had a posterior probability of 100.0%, which means that these 23 genes were included in all three models. Table (3) lists these 25 selected genes, along with their descriptions, posterior probabilities, and univariate log likelihood rankings. This table demonstrates the power of iterative Bayesian Model Averaging in gene selection; even genes with poor univariate rankings may be meaningful predictors when used in combination with other genes. For example, both the highest-ranked gene (BC012161) and the lowest-ranked gene (U70981) in the top p=1000 genes were included in the final set of predictive variables. In addition, several genes with rankings between 500 and 1000 were returned with calculated posterior probabilities of 100.0% (e.g., D83492, AK025754, and NM_005347). This evidence further supports the observation that univariate feature selection methods are overly restricted in their capacity to detect the strongest sets of predictor variables.

Table (3). Genes selected by the iterative BMA algorithm and their corresponding posterior probabilities, univariate log likelihood rankings, and descriptions. The analysis was conducted on the DLBCL dataset with *p*=1000, *nbest*=50, *maxNvar*=25, and *cutPoint*=60. The genes are sorted in descending order of their posterior probabilities and ascending order of their univariate rankings.

Selected genes	Posterior Probability (%)	Univariate Cox ranking	Gene description
BC012161	100.0	1	septin 1
D42043	100.0	4	KIAA0084 protein
X53505	100.0	41	ribosomal protein S12
BF129543	100.0	49	ESTs, weakly similar to A47224 thyroxine-binding globulin precursor
D13666	100.0	73	osteoblast specific factor 2 (fasciclin I-like)
M83664	100.0	93	MHC, class II, DP beta I
AK000978	100.0	101	hypothetical protein FLJ10116
AF009615	100.0	116	a disintegrin and metallo-proteinase domain 10
AK027711	100.0	123	hypothetical protein MGC3234
LC_24015	100.0	129	no description available
K01144	100.0	140	CD74 antigen (invariant polypeptide of MHC, class II antigen associated)
U68418	100.0	181	branched chain aminotransferase 2
D88532	100.0	213	phosphoinositide-3-kinase, regulatory subunit, polypeptide 3 (p55, gamma)
NM_022551	100.0	223	ribosomal protein S18
U18259	100.0	242	MHC, class II transactivator
X64707	100.0	243	ribosomal protein L13
NM_006312	100.0	278	nuclear receptor co-repressor 2
M58297	100.0	385	zinc finger protein 42 (myeloid-specific retinoic acid-responsive)
LC_26524	100.0	473	no description available
AK022743	100.0	499	hypothetical protein FLJ12681
NM_005347	100.0	518	heat shock 70kDa protein 5 (glucose-regulated protein 78kDa)
D83492	100.0	632	EphB6
AK025754	100.0	652	HP1-BP74
AA747694	81.7	885	ESTs, weakly similar to ALU SUBF J
U70981	9.2	1000	interleukin 13 receptor, alpha 2

The algorithm computed the predicted risk scores of the test samples using the maximum likelihood estimate coefficients of the selected genes and the posterior probabilities of the selected models. The risk groups were assigned using the 60% cutoff point of the calculated risk scores in the training set. Of the 80 samples in the validation dataset, 24 were assigned to the high-risk category while 56 were deemed low-risk. Only 3 patients in the high-risk group were still alive at the final follow-up visit, while 27 low-risk patients survived to the study's conclusion. Table (4) provides a summary of the patient samples in each category, along with the margin totals (in the context of survival analysis, the term "censored" refers to a patient who either left the study early or survived until the study's conclusion, while an "uncensored" patient is one who died during the course of the study). The majority of patients in the DLBCL validation dataset did not survive (50/80=62.5%), which explains the relatively large number of uncensored samples assigned to the low-risk group. Of the 30 patients who were alive at the final visit, only 3 (10%) were placed in the high-risk category.

Table (4). The number of censored and uncensored DLBCL patient samples in each risk group, along with the total numbers of censored and uncensored patients and the total number of patients in the high- and low-risk categories.

	Censored	Uncensored	Total
High risk	3	21	24
Low risk	27	29	56
Total	30	50	

To assess the difference between high- and low-risk patient categories, the p-value was calculated using the central chi-square distribution. The two risk groups showed a significant difference in survival probability at p-value=0.00139 and chi-square=10.221. Figure (4) shows the Kaplan-Meier survival analysis curve, where survival time is given in years. These results are

comparable with previous studies using the exact same division between the training and testing sets on the DLBCL data. Rosenwald et al. (2002) identified four separate gene-expression signatures within the patient samples, and the number of microarray features within each signature ranged from 37 to 1333. The p-values illustrating the difference between high-, medium-, and low-risk validation samples in each of the four signatures ranged from 0.009 to 0.11. Bair and Tibshirani (2004) used a semi-supervised principal components method to separate the validation set of 80 DLBCL patients into high- and low-risk groups. They used 17 genes in their analysis and reported a p-value of 0.00124. These results are summarized in Table (5).

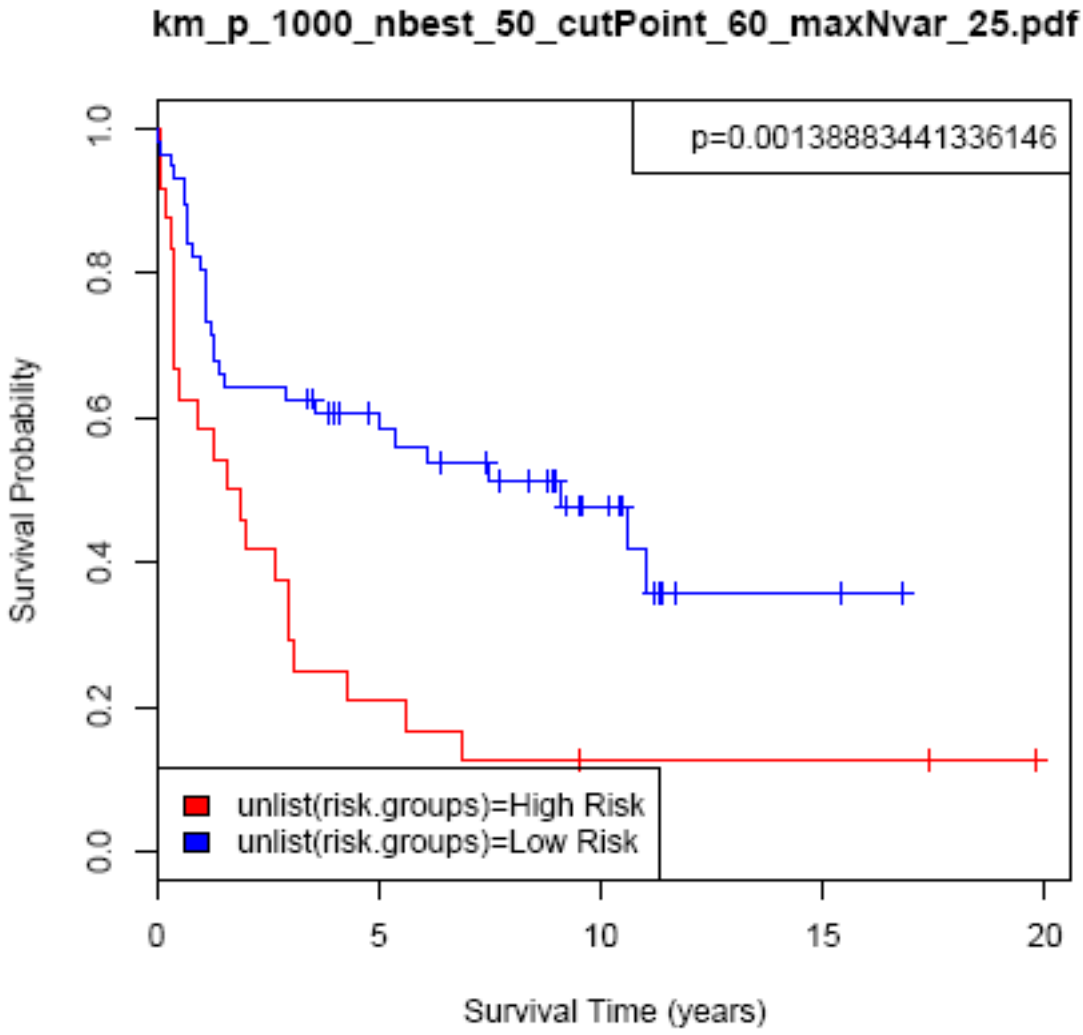

Figure (4). DLBCL: Kaplan-Meier survival analysis curve as a nonparametric estimator of the difference between risk groups. Analysis conducted on the DLBCL dataset with *p*=1000, *nbest*=50, *maxNvar*=25, and *cutPoint*=60. Survival time is given in years, p-value=0.00139, and chi-square=10.221.

Table (5). A comparison among three studies of the number of genes selected and the corresponding p-values in survival analysis on the DLBCL dataset.

	Number of genes	p-value
iterative BMA	25	0.00139
Bair & Tibshirani (2004)	17	0.00124
Rosenwald et al. (2002)	37 – 1333	0.009 – 0.11

BREAST CANCER DATA

In the application of the iterative BMA algorithm to the breast cancer dataset of van't Veer et al. (2002), the optimal input parameters as determined from the cross validation study on the DLBCL dataset were used. The only difference is the size of the active BMA window (represented by the input parameter *maxNvar*). During preliminary analyses (data not shown), training sets with fewer than 100 samples tended to result in fatal errors at higher values of *maxNvar*. These errors occur because smaller matrices lead to instability in fitting the data. Fortunately, these singularity errors can be largely mitigated by reducing the number of variables in the active BMA window. Since the breast cancer training set is so much smaller than the DLBCL training set (61 vs. 160 samples), the value of *maxNvar* was decreased from 25 to 15 variables in order to avoid convergence errors caused by matrix singularity. The algorithm was then applied to the breast cancer dataset with input parameters p=1000, *nbest*=50, *cutPoint*=60, and *maxNvar*=15.

The iterative BMA algorithm for survival analysis selected a total of 15 genes across 84 contending models. The number of variables per model ranged from 5 to 10, with an average of 8.37 genes per model. Table (6) shows the posterior probabilities, univariate log likelihood rankings, and descriptions of the 15 selected genes. This table on the breast cancer data demonstrates an even more striking example of the limitations of univariate feature selection: most of the selected genes have poor univariate rankings. The highest-ranked gene in this group is number 437 out of 1000, and genes ranked 993 through 1000 are all included in this set of selected predictive variables. Of the 15 genes selected by BMA, only 4 of them (27%) were assigned a univariate ranking above 900 by the Cox Proportional Hazards Model. Furthermore, the average ranking of the five genes with a posterior probability of 100.0% is 646.8. Clearly, the more poorly-ranked Cox genes wield a substantial amount of predictive power when considered in combination with other variables. The multivariate iterative BMA

algorithm is able to detect these patterns by analyzing groups of genes as opposed to treating each gene as an individual predictive entity.

Table (6). Genes selected by the iterative BMA algorithm and their corresponding posterior probabilities, univariate log likelihood rankings, and descriptions. Analysis was conducted on the breast cancer dataset with *p*=1000, *nbest*=50, *maxNvar*=15, and *cutPoint*=60. The genes are sorted first in descending order of their posterior probabilities and second in ascending order of their univariate rankings.

Selected genes	Posterior Probability (%)	Univariate Cox ranking	Gene description
NM_000767	100.0	437	cytochrome P450, subfamily IIB (phenobarbital-inducible)
Contig47102_RC	100.0	564	no description available
NM_018965	100.0	935	triggering receptor expressed on myeloid cells 2
NM_013989	100.0	765	deiodinase, iodothyronine, type II (DIO2), transcript variant 1, mRNA
NM_002019	100.0	533	fms-related tyrosine kinase 1 (vascular endothelial growth factor/vascular permeability factor receptor)
NM_021151	99.0	956	carnitine O-octanoyltransferase
AF063936	43.9	984	putative neuronal cell adhesion molecule
NM_004911	40.6	998	protein disulfide isomerase related protein (calcium-binding protein, intestinal-related)
NM_014862	29.5	994	KIAA0307 gene product
Contig40146	19.0	996	wi84e 12.x1 NCI_CGAP_Kid12 Homo sapiens cDNA clone IMAGE : 2400046 3' similar to SW: RASD_DICDI P03967 RAS- LIKE PROTEIN RASD;, mRNA sequence.
NM_012319	17.6	993	LIV-1 protein, estrogen regulated
NM_002411	13.1	995	secretoglobin, family 2A, member 2 (SCGB2A2), mRNA
NM_003645	10.7	997	fatty-acid-Coenzyme A ligase, very long-chain 1
NM_012415	10.3	1000	RAD54 homolog B (S. cerevisiae), transcript variant 1, mRNA (cDNA Clone, ORF Clone)
NM_015972	9.4	999	polymerase (RNA) I polypeptide D, 16kDa

As before, the maximum likelihood estimate coefficients of the 15 selected genes and the posterior probabilities of the 84 selected models were used to compute the predicted risk scores of the 234 patients in the validation dataset. After calculating the risk scores of the patients in the training set to find the 60% *cutPoint*, the test samples with predicted risk scores under the cutoff point were placed in the low-risk group. The patients whose risk scores exceeded the *cutPoint* were designated as high-risk. Since the overall prognosis for breast cancer is more promising than that of diffuse large B-cell lymphoma, 179 patients (76.5%) were still alive at the conclusion of the study. The iterative BMA algorithm assigned 127 patients to the low-risk category and 107 to the high-risk category. Of the patients that survived, 110 (61%) were placed in the low risk group, while 69% (38/55) of the patients that succumbed to their disease were high-risk members. Table (7) shows the number of patients in each group: high-risk/censored, high-risk/uncensored, low-risk/censored, and low-risk/uncensored, in addition to the corresponding margin totals.

Table (7). The number of censored and uncensored breast cancer patient samples in each risk group, along with the total numbers of censored and uncensored patients and the total number of patients in the high- and low-risk categories.

	Censored	Uncensored	Total
High risk	69	38	107
Low risk	110	17	127
Total	179	55	

Figure (5) shows the Kaplan-Meier survival analysis curve in which the proportion of surviving patients from each risk group is plotted against time. For the breast cancer validation samples, the algorithm achieved a p-value of 7.26e-05 and a chi-square statistic of 15.741 from the log-rank test. Figure (5) and the associated p-value show that the iterative BMA method assigned the validation patient samples to relatively distinct risk groups. Furthermore, these

results compare favorably with previous work. For example, Bair and Tibshirani (2004) applied their semi-supervised principal components method to the full dataset of van de Vijver et al. (2002). Recall that this dataset is comprised of 78 training samples and 295 test samples, 61 of which overlap. When testing the difference between risk groups on the full validation set of 295 patients, Bair & Tibshirani used only five predictor genes and reported a p-value of 3.12e-05. They subsequently removed the 61 overlapping samples and applied their method to the validation set of 234 independent samples used in this work. With the exclusion of the overlapping records, their p-value increased to 0.00328. For comparison purposes, Bair & Tibshirani also analyzed the difference between risk groups for both n=234 and n=295 as calculated through the discrete predictor method described in van't Veer et al. (2002). They reported p-values of 0.0105 and 0.00934 respectively.

In order to provide a more direct performance comparison between the iterative BMA method and these alternative procedures, a couple of modifications were required. First, the previously selected 15 genes and 84 models were applied to the full van de Vijver et al. validation set of 295 samples. Figure (6) displays the resulting Kaplan-Meier survival analysis curve (p-value=3.38e-10, chi-square=39.441). Second, the risk scores for the validation set were predicted and the difference between risk groups was calculated using only the top 5 genes with posterior probabilities of 100% from Table (6). Figure (7) shows the Kaplan-Meier survival analysis curve using these 5 genes for n=234 (p-value=9.06e-06, chi-square=19.69936), and Figure (8) provides the same information for n=295 (p-value=1.14e-10, chi-square=41.55947). The exclusion of the bottom-ranked 10 genes did not undermine predictive accuracy; in fact, the results are slightly better than those obtained from using all 15 genes originally selected by the algorithm. Table (8) summarizes the comparison in terms of the p-values and numbers of predictor genes across all aforementioned methodologies for the two different breast cancer validation

sets. As shown in Table (8), the iterative BMA algorithm produced significantly lower p-values than the other two studies using only 5 predictor genes.

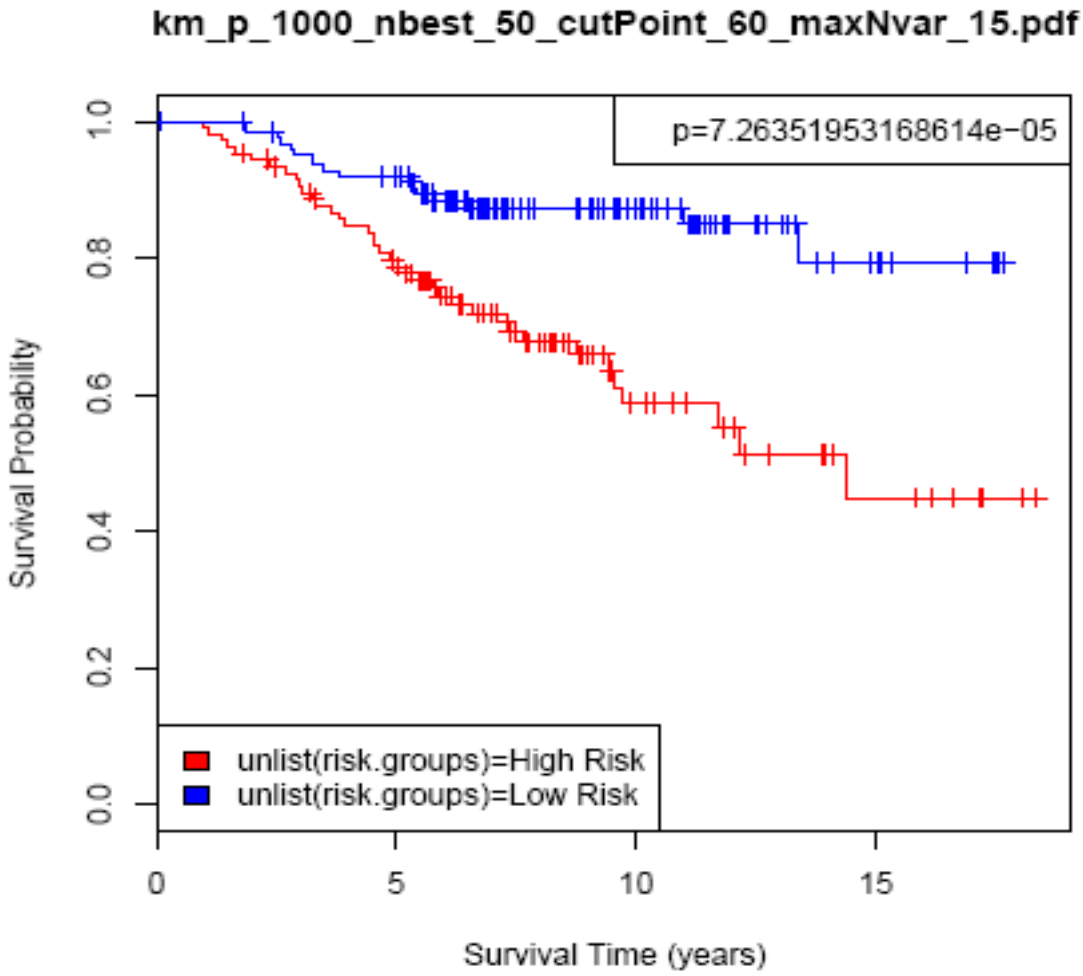

Figure (5). Breast Cancer (n=234): Kaplan-Meier survival analysis curve as a nonparametric estimator of the difference between risk groups. Analysis conducted on the breast cancer dataset with *p*=1000, *nbest*=50, *maxNvar*=15, and *cutPoint*=60. Survival time is given in years, p-value=7.26e-05, and chi-square=15.741.

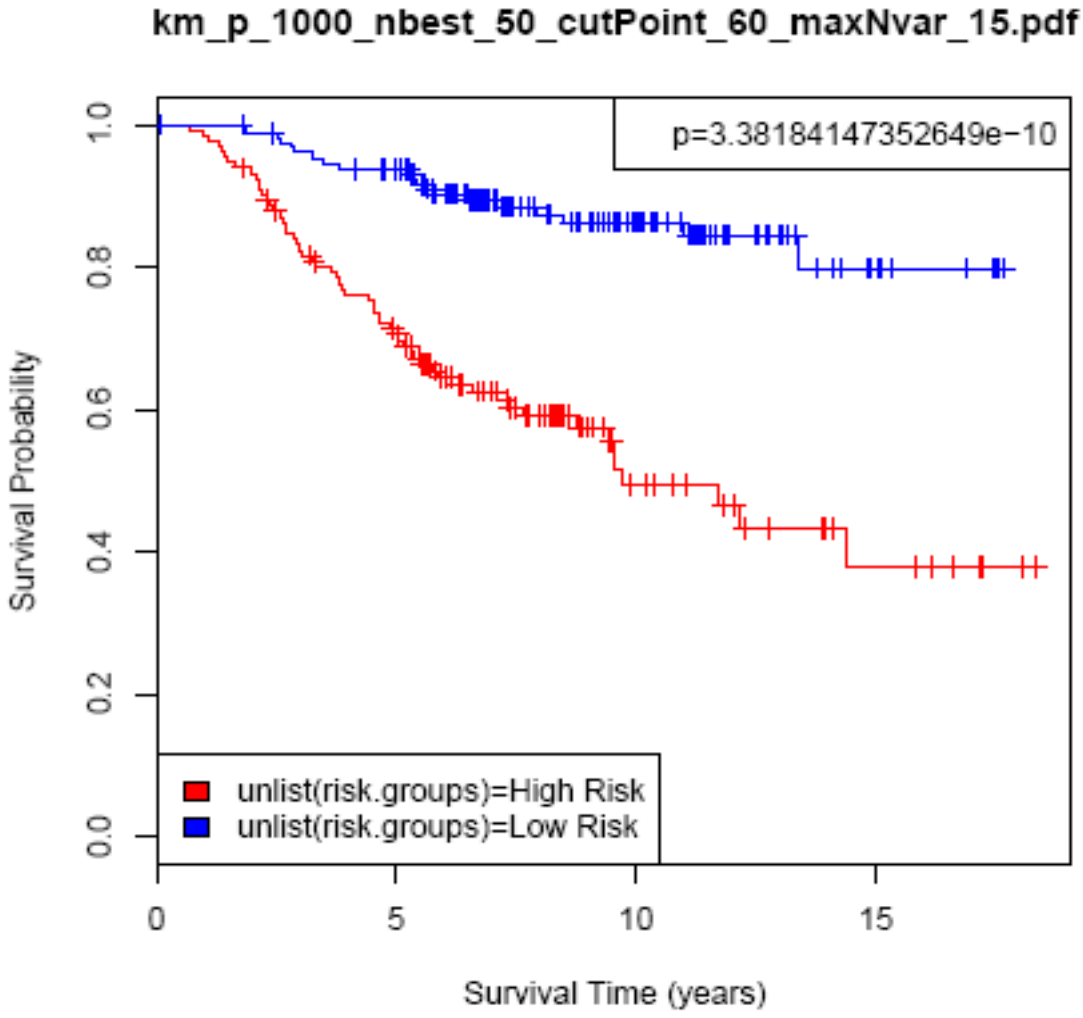

Figure (6). Breast Cancer (n=295): Kaplan-Meier survival analysis curve calculated on the full 295-sample breast cancer validation set of van di Vijver et al. (2002). In this analysis, p=1000, nbest=50, maxNvar=15, and cutPoint=60. Survival time is given in years; p-value=3.38e-10 and chi-square=39.441.

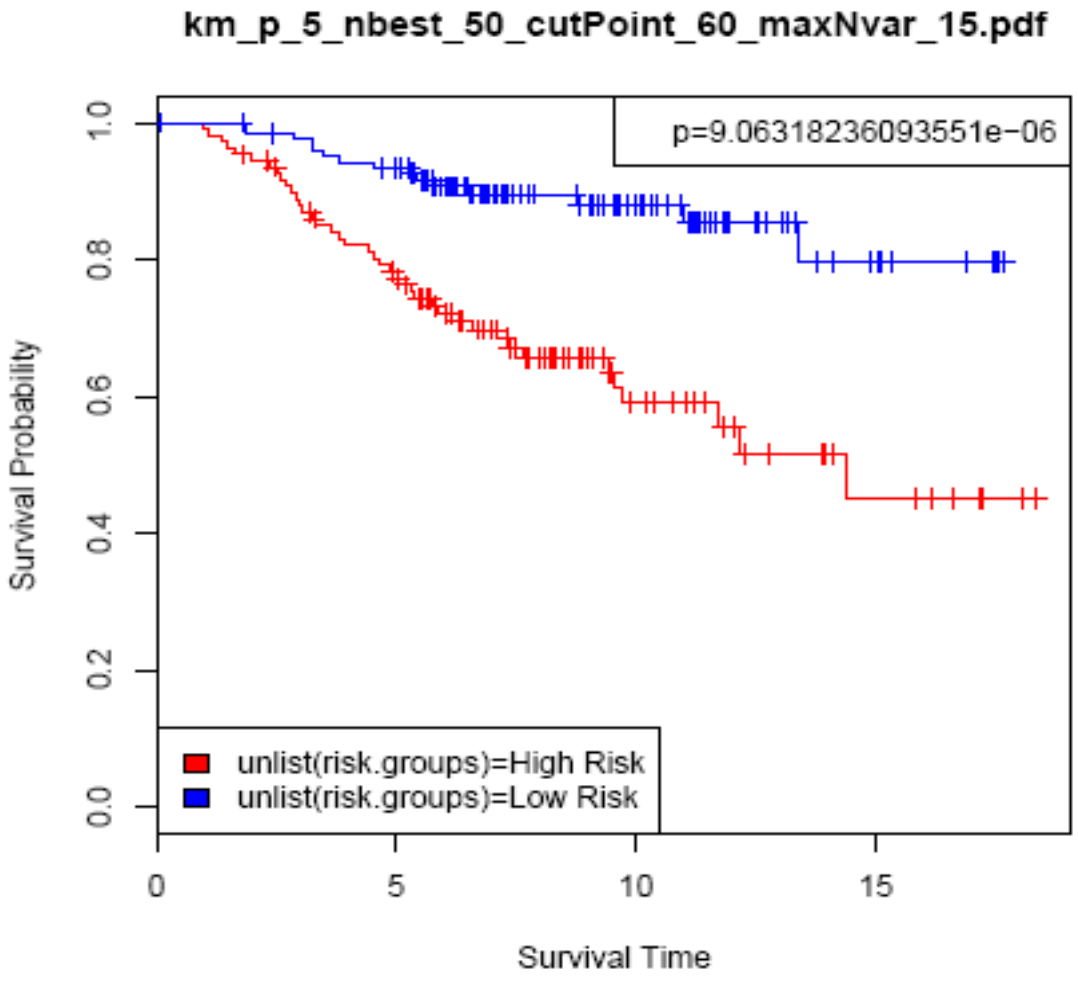

Figure (7). 5-gene Breast Cancer, n=234: Kaplan-Meier survival analysis curve as a nonparametric estimator of the difference between risk groups. Analysis conducted on the breast cancer dataset with *p*=5, *nbest*=50, *maxNvar*=15, and *cutPoint*=60. Validation set risk scores were predicted using 5 top-ranked genes across 2 selected models. Survival time is given in years, p-value=9.06e-06, and chi-square=19.699.

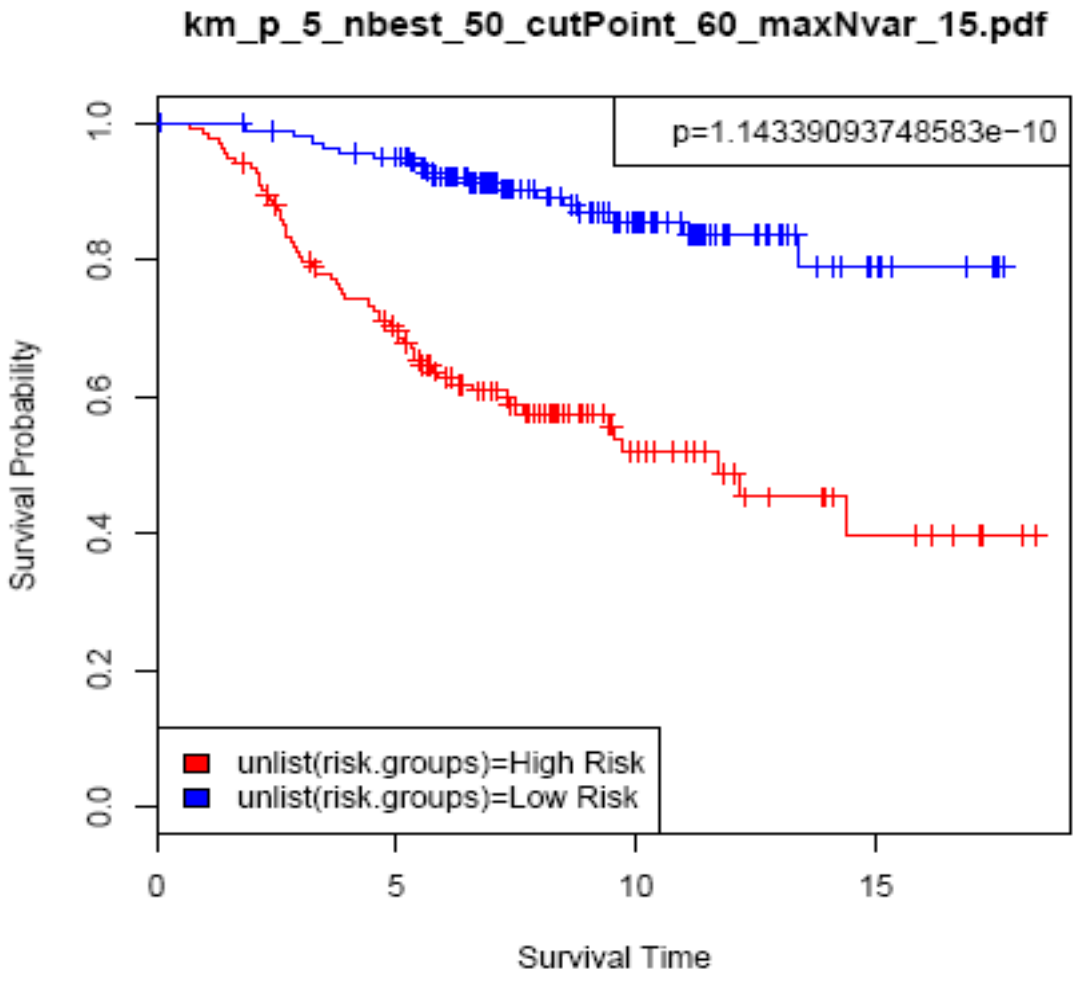

Figure (8). 5-gene Breast Cancer, n=295: Kaplan-Meier survival analysis curve calculated on the full 295-sample breast cancer validation set of van di Vijver et al. (2002). In this analysis, *p*=5, *nbest*=50, *maxNvar*=15, and *cutPoint*=60. Validation set risk scores were predicted using 5 top-ranked genes across 2 selected models. Survival time is given in years, p-value=1.14e-10, and chi-square=41.559.

Table (8). A comparison of p-values and numbers of genes selected across three different survival analysis methods on the full breast cancer validation set of van de Vijver et al. (2002) (n=295), and the partial breast cancer validation set used in this work with 61 overlapping samples removed (n=234). The iterative BMA method produced the best results: the lowest p-value using the smallest number of predictor genes.

	Number of Genes	n=234 p-values	n=295 p-values
iterative BMA using 15 selected genes	15	7.26e-05	3.38e-10
iterative BMA using 5 genes with 100% posterior probabilities	5	9.06e-06	1.14e-10
Bair & Tibshirani (2002) Principle Components	5	0.00328	3.12e-05
Method of van't Veer et al. (2002) (as calculated by Bair & Tibshirani)	70	0.0105	0.00934

DISCUSSION

This paper has proposed an extension of the iterative BMA algorithm of Yeung et al. (2005) for application to survival analysis on high-dimensional microarray data. This multivariate technique accounts for model uncertainty by averaging over the posterior probabilities of the strongest contending models (sets of potentially overlapping predictor genes). This project shows that the *iterativeBMAsurv* algorithm achieves significantly lower p-values on the breast cancer data when compared to alternative methods using the same number of predictor genes. In addition, the algorithm produces similar results on the DLBCL dataset when compared to the best method in the literature. Table (9) provides a summary of the results. The output of the iterative BMA algorithm for survival analysis is particularly well suited to biological interpretation. The posterior probability of a chosen gene represents its overall contribution towards the patient risk score across all selected models. The posterior probabilities of the chosen models indicate the relative strength of the predictor genes from each model in patient risk assessment. The models chosen for the DLBCL and breast cancer datasets are very simple, consisting of anywhere from 5 to 25 genes. The p-values and Kaplan-Meier survival analysis curves are used to estimate the difference between risk groups, demonstrating the strength of the iterative BMA algorithm as measures of high predictive accuracy.

Table (9). A summary of the results from the application of the iterative BMA algorithm to the DLBCL dataset, the partial non-overlapping breast cancer dataset (n=234), and the full overlapping breast cancer dataset (n=295).

	Number of genes	Number of models	p-value	chi-square
DLBCL	25	3	0.00139	10.221
Breast Cancer n=234 (15 genes)	15	84	7.26e-05	15.714
Breast Cancer n=295 (15 genes)	15	84	3.38e-10	39.441
Breast Cancer n=234 (5 genes)	5	2	9.06e-06	19.699
Breast Cancer n=295 (5 genes)	5	2	1.14e-10	41.559

The results from this study show that genes with poor univariate rankings are often selected by the *iterativeBMAsurv* algorithm (see Tables (3) and (6)). In fact, this is one of the most striking merits of the method and results presented in this paper. Most univariate feature selection algorithms typically retain the top-ranked univariate genes and discard those with poor univariate rankings. These methodologies could miss the truly important predictor genes selected by *iterativeBMAsurv*. While it is true that the genes and models selected by the iterative BMA procedure are contingent upon the initial univariate rankings, all *p* top-ranked univariate genes may be included in the models selected by the *iterativeBMAsurv* method. The analyses from this project showed that setting the parameter *p* to a large value (e.g., 1000) generally yields high prediction

accuracy. In addition to the parameter *p*, the algorithm requires the input of a few other user-specified parameters. One example is *nbest*, which is used by the leaps and bounds algorithm from Furnival and Wilson (1974) to isolate the *nbest* strongest models. Higher values of *nbest* increase the computation time, but overly restrictive values undermine predictive accuracy by failing to return potentially contributory models. Experimentation determined that *nbest*=20, 50, and 100 generally yielded good results, with a value of 50 exhibiting the ideal tradeoff between predictive power and computational efficiency on the DLBCL data. For example, it takes about 1.5 hours to run the iterative BMA algorithm with *p*=1000 genes and *nbest*=50 on a machine with 2 gigabytes of RAM and a 2.0 GHz Intel dual core processor. Reducing *nbest* to 20 cuts the running time down to 40 minutes, but the p-value representing the difference between risk groups is slightly larger. On the other hand, setting *nbest* to 100 significantly increases the computation time with no appreciable improvement in prediction. Cross validation can be used to determine the optimal input parameters for each dataset.

While the results obtained in this study are highly encouraging, the iterative BMA algorithm for survival analysis presents some limitations and areas for future development. The mathematical calculations conducted by the BMA methods are close approximations, but they could be computed with greater precision. For example, the maximum likelihood estimate of equation (3) provides a sufficient approximation to the predictive distribution, but the more computationally intensive Markov Chain Monte Carlo methods might yield the true predictive distribution with greater accuracy (Volinsky et al., 1997). The approximation of the posterior model probabilities calculated in equation (6) could also be improved; see Raftery (1996) for a detailed discussion on the various extensions of BMA. In addition, the procedure is subject to the fitter error originating from the R survival package. This error, which usually occurs on small training sets, halts the progress of the algorithm’s iterations upon the

first occurrence of matrix convergence. Reducing the value of the parameter *maxNvar* is a possible solution to this problem, but this in turn limits the algorithm's flexibility to consider larger models. An alternative solution involves modifying the BMA source code. All the calls to the survival package in the *bic.surv* function can be wrapped within a try block to catch the exception, hence allowing the iterative procedure to continue. This technique seems to work well on smaller datasets that are subject to the error at high values of *maxNvar*.

CONCLUSION

This paper has demonstrated that the iterative BMA algorithm for survival analysis consistently isolates a small group of relevant predictor genes from high-dimensional microarray datasets, providing better predictive power than other feature selection and supervised learning methods. The algorithm is easy to use, computationally efficient, and highly accurate. It efficiently reduces vast amounts of microarray data down to a handful of predictor variables, making it a cost-effective diagnostic tool in the clinical setting. Future work in this area would involve a collaboration with cancer biologists to validate the predictor genes selected by applying the *iterativeBMAsurv* algorithm to microarray data, and to assess the prediction accuracy of the methodology on PCR data generated using independent patient samples. Furthermore, the iterative BMA algorithm could be extended to other types of high-throughput data such as proteomics data produced from mass spectrometry. The multivariate nature of BMA combined with its ability to account for model uncertainty makes it a particularly attractive candidate to extract predictive genes from any high-dimensional biological dataset.

All analyses in this study were conducted using R statistical software (http://www.r-project.org/). The Bioconductor packages for the iterative BMA algorithms for classification and survival analysis described in this paper are available for download from Bioconductor's website (http://www.bioconductor.org) as the *iterativeBMA* and *iterativeBMAsurv* packages respectively. See appendix I for a vignette describing the functionality of *iterativeBMAsurv*, including sample function calls and output. Appendix II contains the source code for the *iterativeBMAsurv* package. Please visit the supplemental website (username: amanu, password: ibmasurv; http://expression.washington.edu/ibmasurv/protected) for access to the DLBCL and breast cancer datasets, along with other helpful links and information.

REFERENCES

Alizadeh, A., Eisen, M., Davis, R., Ma, C., Sabet, H. et al. (1999). The Lymphochip: A Specialized cDNA Microarray for the Genomic-Scale Analysis of Gene Expression in Normal and Malignant Lymphocytes. *Cold Spring Harbor Symposia on Quantitative Biology*, **64**, 71-78.

Bair, E., & Tibshirani, R. (2004). Semi-Supervised Methods to Predict Patient Survival from Gene Expression Data. *PLOS Biology*, **2**(4), 511-522.

Beer, D., Kardia, S., Huang, C., Giordano, T., Levin, A., et al. (2002). Gene-Expression Profiles Predict Survival of Patients with Lung Adenocarcinoma. *Nature Medicine*, **8**(8), 816-824.

Ben-Dor, A., Bruhn, L., Friedman, N., Nachman, I., Schummer, M., et al. (2000). Tissue Classification with Gene Expression Profiles. *Journal of Computational Biology*, **7**(3-4), 559-583.

Brier, G. (1950). Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, **78**(1), 1-3.

Chen, C., Wu, T., Wu, Y., Huang, Y., & Lee, J. (2003). Characterization of the Univariate and Multivariate Techniques on the Analysis of Simulated and fMRI Datasets with Visual Task. *Nuclear Science Symposium Conference Record, 2003 IEEE*, **4**, 2468-2472.

Chow, M., Moler, E., & Mian, I. (2001). Identifying Marker Genes in Transcription Profiling Data Using a Mixture of Feature Relevance Experts. *Physiol Genomics*, **5**, 99-111.

Cohen, J. (2004). Bioinformatics – An Introduction for Computer Scientists. *ACM Computing Surveys*, **36**(2), 122-158.

Cox, D. (1972). Regression Models and Life Tables. *Journal of the Royal Statistical Society, Series B*, **34**(2), 187-220.

Derksen, S., & Keselman, H. (1992). Backward, Forward and Stepwise Automated Subset Selection Algorithms: Frequency of Obtaining Authentic and Noise Variables. *British Journal of Mathematical and Statistical Psychology*, **45**, 265-282.

Draper, D. (1995). Assessment and Propagation of Model Uncertainty. *Journal of the Royal Statistical Society, Series B*, **57**(1), 45-97.

Dudoit, S., Fridlyand, J., & Speed, T. (2002). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. *Journal of the American Statistical Association*, **97**(457), 77-87.

Furnival, G., & Wilson, R. (1974). Regression by Leaps and Bounds. *Technometrics*, **16**, 499-511.

Geman, D., D'Avignon, C., Naiman, D., & Winslow, R. (2004). Classifying Gene Expression Profiles from Pairwise mRNA Comparisons. *Statistical Applications in Genetics and Molecular Biology*, **3**(1), 1-21.

Golub, T., Lonim, D., Tamayo, P., Huard, C., Gaasenbeek, M. et al. (1999). Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, **286**(5439), 531-537.

Glinsky, G., Glinskii, A., Stephenson, A., Hoffman, R., Gerald, W. (2004). Gene Expression Profiling Predicts Clinical Outcome of Prostate Cancer. *The Journal of Clinical Investigation*, **113**(6), 913-923.

Guyon, I., Weston, J., & Barnhill, S. (2002). Gene Selection for Cancer Classification Using Support Vector Machines. *Machine Learning*, **46**, 389-422.

Hoeting, J., Madigan, D., Raftery, A., & Volinsky, C. (1999). Bayesian Model Averaging: A Tutorial. *Statistical Science*, **14**(4), 382-417.

Hosmer, D., Lemeshow, S., & May, S. (2008). *Applied Survival Analysis: Regression Modeling of Time to Event Data, 2nd Edition.* Wiley Interscience: Wiley Series in Probability and Statistics.

Hu, H., Li, J., Plank, A., Wang, H., & Daggard, G. (2006). A Comparative Study of Classification Methods for Microarray Data Analysis. *Proceedings of the Fifth Australasian Conference on Data Mining and Analystics*, **61**, 33-37.

Huang, T., Kecman, V., & Kopriva, I. (2006). *Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-Supervised, and Unsupervised Learning.* Springer: Germany. Studies in Computational Intelligence, Vol. 17.

Jiang, H., Deng, Y., Chen, H., Tao, L., Sha, Q., et al. (2004). Joint Analysis of Two Microarray Gene-Expression Data Sets to Select Lung Adenocarcinoma Marker Genes. *BMC Bioinformatics*, **5**(81). http://www.biomedcentral.com/1471-2105/5/81.

Jiangeng, L., Yanhua, D., & Xiaogang, R. (2007). A Novel Hybrid Approach to Selecting Marker Genes for Cancer Classification Using Gene Expression Data. *The 1st International Conference on Bioinformatics and Biomedical Engineering, 2007, ICBBE 2007*, 264-267.

Kaplan, E., & Meier, P. (1958). Nonparametric Estimation from Incomplete Observations. *Journal of the American Statistical Association*, **53**, 457-481.

Korkola, J. E., Blaveri, E., DeVries, S., Moore, D. H., Hwang, E. S., et al. (2007). Identification of a Robust Gene Signature that Predicts Breast Cancer Outcome in Independent Data Sets. *BMC Cancer*, **7**(61), http://www.biomedcentral.com/1471-2407/7/61.

Kotsiantis, S. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, **31**, 249-268.

Kuo, L., & Smith, A. (1992). Bayesian Computations in Survival Models Via the Gibbs Sampler. In Klein, J., & Goel, P. (Eds.), *Survival Analysis: State of the Art*. Dordrecht: Boston. Proceedings of the NATO Advanced Research Workshop on Survival Analysis and Related Topics, Columbus, Ohio, pp. 11-24.

Lai, C., Reinders, M., van't Veer, L., & Wessels, L. (2006). A Comparison of Univariate and Multivariate Gene Selection Techniques for Classification of Cancer Datasets. *BMC Bioinformatics*, **7**(235), http://www.biomedcentral.com/1471-2105/7/235.

Li, L., Weinberg, C., Darden, T., & Pedersen, L. (2001). Gene Selection for Sample Classification Based on Gene Expression Data: Study of Sensitivity to Choice of Parameters of the GA/KNN Method. *Bioinformatics*, **17**(12), 1131-1142.

Liu, H. & Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Springer: The Springer International Series in Engineering and Computer Science, Vol. 454.

Liu, H., & Motoda, H., (Eds). (2008). *Computational Methods of Feature Selection*. Chapman & Hall/CRC: Boca Raton. Data Mining and Knowledge Discovery Series.

Lu, Y., Lemon, W., Liu, P., Yi, Y., Morrison, C., et al. (2006). A Gene Expression Signature Predicts Survival of Patients with Stage I Non-Small Cell Lung Cancer. *PLoS Medicine*, **3**(12), 2229-2243.

Madigan, D., & Raftery, A. (1994). Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window. *Journal of the American Statistical Association*, **89**, 1335-1346.

Nguyen, D., & Rocke, D. (2002). Tumor classification by Partial Least Squares Using Microarray Gene Expression Data. *Bioinformatics*, **18**(1), 39-50.

[NHLCP] Non-Hodgkin's Lymphoma Classification Project. (1997). A Clinical Evaluation of the International Lymphoma Study Group Classification of Non-Hodgkin's Lymphoma. *Blood*, **89**, 3909-3918.

Piatetsky-Shapiro, G., Khabaza, T., Ramaswamy, S. (2003). Capturing Best Practice for Microarray Gene Expression Data Analysis. *Conference on Knowledge Discovery in Data. Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*. Washington D.C.: Industrial/government track, 407-415.

Piatetsky-Shapiro, G., Tamayo, P. (2003). Microarray Data Mining: Facing the Challenges. *ACM SIGKDD Explorations Newsletter*, **5**(2), 1-5.

Pudil, P., Novovicova, J., & Kittler, J. (1994). Floating Search Methods in Feature Selection. *Physical Review Letters*, **15**, 1119-1125.

Raftery, A. (1995). Bayesian Model Selection in Social Research (with Discussion). In Marsden, P. (Ed.), *Sociological Methodology 1995*. Blackwell: Cambridge, MA, pp. 111-196.

Raftery, A. (1996). Approximate Bayes Factors and Accounting for Model Uncertainty in Generalised Linear Models. *Biometrika*, **83**(2), 251-266.

Raponi, M., Zhang, Y., Yu, J., Chen, G., Lee G., et al. (2006). Gene Expression Signatures for Predicting Prognosis of Squamous Cell and Adenocarcinomas of the Lung. *Cancer Research*, **66**(15), 7466-7472.

Rosenwald, A., Wright, G., Wing, C., Connors, J., Campo, E. et al. (2002). The Use of Molecular Profiling to Predict Survival After Chemotherapy for Diffuse Large-B-Cell Lymphoma. *The New England Journal of Medicine*, **346**(25), 1937-1947.

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, **6**(2), 461-464.

Shipp, M., Ross, K., Tamayo, P., Weng, A., Kutok, J., et al. (2002). Diffuse Large B-Cell Lymphoma Outcome Prediction by Gene-Expression Profiling and Supervised Machine Learning. *Nature Medicine*, **8**(1), 68-74.

Silva, P., Hashimoto, R., Kim, S., Barrera, J., Brandao, L, et al. (2005). Feature Selection Algorithms to Find Strong Genes. *Pattern Recognition Letters*, **26**(10), 1444-1453.

Sotiriou, C., Neo, S., McShane, L., Korn, E. Long, P. et al. (2003). Breast Cancer Classification and Prognosis Based on Gene Expression Profiles from a Population-Based Study. *PNAS*, **100**(18), 10393-10398.

Tan, A., Naiman, D., Xu, L., Winslow, R., & Geman, D. (2005). Simple Decision Rules for Classifying Human Cancers from Gene Expression Profiles. *Bioinformatics*, **21**(20), 3896-3904.

Taplin, R. (1993). Robust Likelihood Calculation for Time Series. *Journal of the Royal Statistical Society, Series B*, **55**(4), 829-836.

Taplin, R., & Raftery, A. (1994). Analysis of Agricultural Field Trials in the Presence of Outliers and Fertility Jumps. *Biometrics*, **50**(3), 764-781.

van de Vijver, M., He, Y., van't Veer, L., Dai, H., Hart, A., et al. (2002). A Gene-Expression Signature as a Predictor of Survival in Breast Cancer. *The New England Journal of Medicine*, **347**(25), 1999-2009.

van't Veer, L., Dai, H., van de Vijver, M., He Y., Hart, A., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, **415**(8), 530-536.

Volinsky, C., Madigan, D., Raftery, A., & Kronmal, R. (1997). Bayesian Model Averaging in Proprtional Hazard Models: Assessing the Risk of a Stroke. *Applied Statistics*, **46**(4), 433-448.

Witten, I., & Frank, R. (2005). *Data mining: Practical Machine Learning Tools and Techniques, Second Edition*. Elsevier, Inc: San Francisco. Morgan Kaufman Series in Data Management Systems.

Xu, L., Geman, D., & Winslow, R. (2007). Large-Scale Integration of Cancer Microarray Data Identifies a Robust Common Cancer Signature. *BMC Bioinformatics*, **8**(275), http://www.biomedcentral.com/1471-2105/8/275.

Xu, L., Tan, A., Naiman, D., Geman, D., & Winslow, R. (2005). Robust Prostate Cancer Marker Genes Emerge from Direct Integration of Inter-Study Microarray Data. *Bioinformatics*, **21**(20), 3905-3911.

Yeung, K., Bumgarner, R., & Raftery, A. (2005). Bayesian Model Averaging: Development of an Improved Multi-Class, Gene Selection and Classification Tool for Microarray Data. *Bioinformatics*, **21**(10), 2394-2402.

Yu, J., Yu, J., Almal, A., Dhanasekaran, S., Ghosh, D., et al. (2007). Feature Selection and Molecular Classification of Cancer Using Genetic Programming. *Neoplasia*, **9**(4), 292-303.

Appendix I: Vignette for the *iterativeBMAsurv* Package

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

May 19, 2008

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 (Annest et al.). 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 (Raftery, 1995). 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 (Cox, 1972). 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 (Rosenwald et al., 2002) 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 (Annest et al.) for recommended input parameters.

```
> data (trainData)
> data (trainSurv)
> 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.

```
> ret.list <- iterateBMAsurv.train.wrapper(x = trainData,
  surv.time = trainSurv, cens.vec = trainCens, nbest = 5)

[1]  "17: Explored up to variable # 100"
[1]  "Iterate bic.surv is done!"
[1]  "Selected genes:"
[1]  "X31687" "X33840" "X31242" "X16948" "X31471" "X17154"
[7]  "X28531" "X19241" "X26146" "X17804" "X27332" "X17241"
[13] "X32212" "X29911" "X33558" "X33013" "X27884" "X33706"
[19] "X16817" "X31968" "X30209" "X29650" "X25054" "X16988"
[25] "X32904"
[1]  "Posterior probabilities of selected genes:"
[1]  100.0  47.5  47.3  2.4  38.5  28.5  40.1  96.7  2.8
[10] 1.7  0.0  59.9  0.0  0.0  10.0  0.0  2.5  58.3  2.1
[20] 98.8  28.4  7.1  95.1  0.0 100.0

> ret.bic.surv <- ret.list$obj
> gene.names <- ret.list$curr.names
> top.gene.names <- gene.names[ret.bic.surv$probne0 > 0]
> top.gene.names

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

> ret.bic.surv$postprob

[1]  0.075782322 0.068183539 0.062240254 0.056227073
[5]  0.045761712 0.044794588 0.043328132 0.042831731
[9]  0.039567629 0.039285627 0.038997242 0.034867824
[13] 0.032225236 0.030210326 0.026904418 0.025508701
[17] 0.025052995 0.024869256 0.021711946 0.021061750
[21] 0.020689119 0.020114454 0.017345536 0.017179713
[25] 0.017104052 0.015294500 0.014059561 0.014050900
[29] 0.012658966 0.010182444 0.008768581 0.007844758
```

```
[33] 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.

```
> data(testData)
> data(testSurv)
> data(testCens)
> curr.test.dat <- testData[, top.gene.names]
> y.pred.test <- apply(curr.test.dat, 1, predictBicSurv,
```

```
  postprob.vec = ret.bic.surv$postprob, mle.mat =
  ret.bic.surv$mle)
> y.pred.train <- apply(trainData[, top.gene.names], 1,
  predictBicSurv, postprob.vec = ret.bic.surv$postprob,
  mle.mat = ret.bic.surv$mle)
> ret.table <- predictiveAssessCategory(y.pred.test,
  y.pred.train, testCens, cutPoint = 50)
> 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

> mySurv.obj <- Surv(testSurv, testCens)
> 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
High Risk  18    10       10.5      0.0267    0.0538
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 (Yeung et al., 2005). 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.

```
> ret.bma <- iterateBMAsurv.train.predict.assess(train.dat
  = trainData, test.dat = testData, surv.time.train =
  trainSurv, surv.time.test = testSurv,
  cens.vec.train = trainCens, cens.vec.test = testCens, p
  = 10, nbest = 5)

[1]  "1: Explored up to variable # 10"
[1]  "Iterate bic.surv is done!"
[1]  "Selected genes:"
[1]  "X33310" "X28197" "X19373" "X16527" "X27415" "X24394"
[7]  "X28531" "X27585" "X27766" "X26940"
[1]  "Posterior probabilities of selected genes:"
[1]  100.0  18.9  16.9  0.0  19.6  0.0  47.0  12.6  0.0
[10] 4.3
[1]  "# selected genes = 7"
[1]  "# selected models = 11"
```

```
[1]  "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
High Risk  16     9       8.21     0.0752      0.126
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

> 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
High Risk  16    9       8.21       0.0752     0.126
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.

```
> cv <- crossVal(exset = trainData, survTime = trainSurv,
  censor = trainCens, diseaseType = "DLBCL", noRuns = 1,
  noFolds = 2, p = 10, nbest = 5)

[1]  "*************** BEGINNING CV RUN 1 ***************"
[1]  "---------- BEGINNING FOLD 1 ----------"
[1]  "1: Explored up to variable # 10"
[1]  "Iterate bic.surv is done!"
[1]  "Selected genes:"
[1]  "X24394" "X27415" "X24298" "X16164" "X19373" "X16527"
[7]  "X24396" "X28531" "X26940" "X27766"
[1]  "Posterior probabilities of selected genes:"
[1]  65.8  42.4  39.8  25.5  4.1  0.8  0.9  10.3  42.7
[10] 2.2
[1]  "# selected genes = 10"
[1]  "# selected models = 25"
[1]  "Risk Table:"
           cens.vec.test
              0   1
  High Risk   6   6
  Low Risk   11   9

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

           N  Observed  Expected  (O-E)^2/E  (O-E)^2/V
High Risk  12     6       4.9       0.246      0.376
Low Risk   20     9      10.1       0.119      0.376

Chisq = 0.4 on 1 degrees of freedom, p = 0.54

          [,1]      [,2]
[1,]  3.206838 -3.206838
[2,] -3.206838  3.206838
[1]  0.3756349
[1]  "---------- BEGINNING FOLD 2 ----------"
[1]  "1: Explored up to variable # 10"
[1]  "Iterate bic.surv is done!"
[1]  "Selected genes:"
```

```
[1]  "X29224" "X31968" "X33166" "X33014" "X33706" "X27774"
[7]  "X24816" "X33310" "X27585" "X26586"
[1]  "Posterior probabilities of selected genes:"
[1]  63.5  96.6  17.7  10.7  100.0  84.1  10.8  11.7  15.3
[10] 4.1
[1]  "# selected genes = 10"
[1]  "# selected models = 23"
[1]  "Risk Table:"
           cens.vec.test
              0   1
  High Risk   1   7
  Low Risk   12   13

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

           N  Observed  Expected  (O-E)^2/E  (O-E)^2/V
High Risk  8       7      2.67       7.03       8.85
Low Risk   25      13    17.33       1.08       8.85

Chisq = 8.9 on 1 degrees of freedom, p = 0.00293

          [,1]      [,2]
[1,]  2.119116 -2.119116
[2,] -2.119116  2.119116
[1]  8.850042
[1]  "***************** END CV RUN 1 *****************"

[1]  "Overall results from this run:"
          Censored Uncensored Percent Uncensored
Low Risk        23         22           48.88889
High Risk        7         13           65.00000

[1]  "Overall average result matrix over all runs:"
          Censored Uncensored Percent Uncensored
Low Risk        23         22           48.88889
High Risk        7         13           65.00000

[1]  "Average p-value across all folds and all runs:"
[1]  0.2714398

[1]  "Standard deviation of p-values across all folds and
     all runs:"
[1]  0.379729
```

```
[1]  "Average chi-square value across all folds and all
     runs:"
[1]  4.612839

[1]  "Standard deviation of chi-square across all folds
     and all runs:"
[1]  5.992311
```

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 1).

In Figure 1, 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.

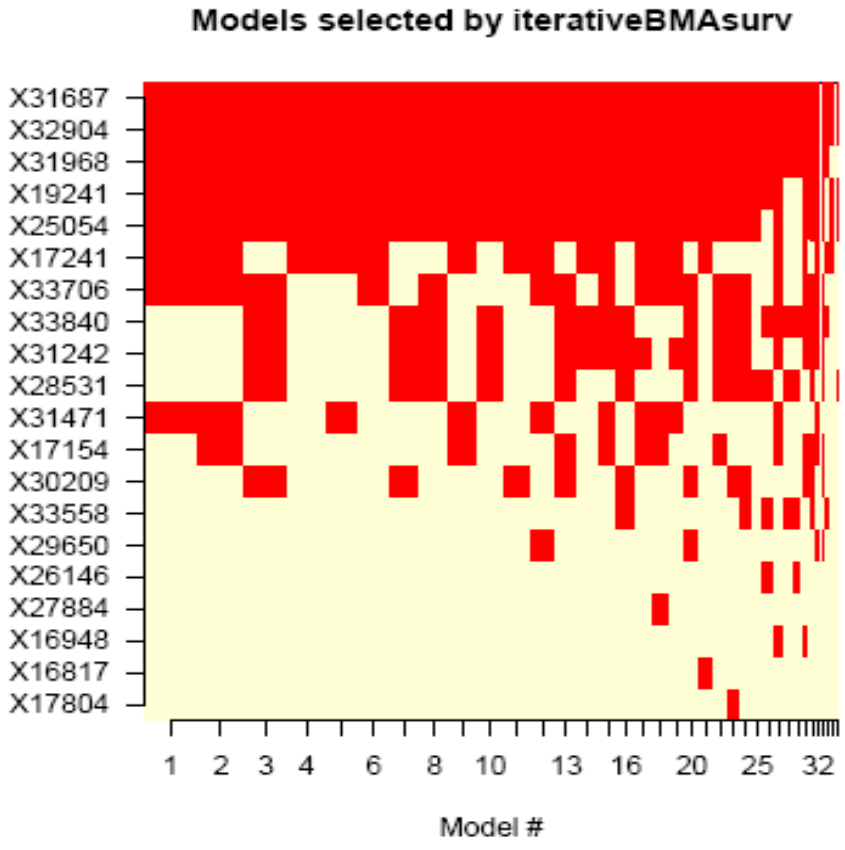

Figure 1: An image plot showing the selected genes and models.

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.

References

A. Annest, R.E. Bumgarner, A.E. Raftery, and K.Y. Yeung. Iterative Bayesian Model Averaging: A method for the application of survival analysis to high-dimensional microarray data. *To come*.

D. Cox. Regression models and life tables. *Journal of the Royal Statistical Society*, 34:187-220, 1972.

A.E. Raftery. Bayesian model selection in social research (with discussion). *Sociological Methodology*, 25:111-196, 1995.

A. Rosenwald, G. Wright, C. Wing, J. Connors, E. Campo, R. Fisher, R. Gascoyne, H. Muller-Hermelink, E. Smeland, J. Giltnane, E. Hurt, H. Zhao, L. Averett, L. Yang, W. Wilson, E. Jaffe, R. Simon, R. Klausner, J. Powell, P. Duffey, D. Longo, T. Greiner, D. Weisenburger, W. Sanger, B. Dave, J. Lynch, J. Vose, J. Armitage, E. Montserrat, A. Lopez- Guillermo, T. Grogan, T. Miller, M. LeBlanc, G. Ott, S. Kvaloy, J. Delabie, H. Holte, P. Krajci, T. Stokke, L. Staudt. The use of molecular profiling to predict survival after chemotherapy for diffuse large B-cell lymphoma. *The New England Journal of Medicine*, 346(25):1937-1947, 2002.

K.Y. Yeung, R.E. Bumgarner, and A.E. Raftery. Bayesian Model Averaging: Development of an improved multi-class, gene selection and classification tool for microarray data. *Bioinformatics*, 21(10):2394-2402, 2005.

Appendix II: Source Code for the *iterativeBMAsurv* Package

iterateBMAsurv.R

```
# Authors: Amalia Annest and Ka Yee Yeung

# Function:    iterateBMAsurv.train
# Goal:        Carry out iterative bic.surv in R
# Arguments: x = matrix of independent variables in the training set
#                (variables = columns; samples = rows)
#            surv.time = vector of survival times for all samples in the training
#                        set
#            cens.vec = vector of censor data for all samples in the training set
#                       (0 = censored, 1 = uncensored)
#            curr.mat = matrix of independent variables in the active bic.surv
#                       window
#            stopVar = 0 to continue iterations, 1 to stop iterations (default 0)
#            nextVar = placeholder indicating the next variable to be brought
#                      into the active bic.surv window
#            nbest = number specifying the number of models of each size
#                    returned to bic.surv in the BMA package (default 10)
#            maxNvar = size of the active bic.surv window (default 25
#                      variables - may need to be reduced with smaller
#                      training sets)
#            maxIter = max number of iterations in repeating bic.surv (default
#                      200000)
#            thresProbne0 = threshold to remove genes with low probne0
#                           (default 1%)
#            verbose = a boolean variable indicating whether or not to print
#                      interim information to the console (default FALSE)
# Returns:   updated curr.mat
#            updated nextVar
#            updated stopVar

iterateBMAsurv.train <- function (x, surv.time, cens.vec,
     curr.mat, stopVar=0, nextVar, nbest=10, maxNvar = 25,
     maxIter=200000, thresProbne0 = 1, verbose = FALSE,
     suff.string="") {

     # Iterative bic.surv
     currIter <- 0

     while (stopVar == 0 && currIter < maxIter) {
          # Run bic.surv
          if (verbose == TRUE) {
```

```
     cat (paste("Current iteration is ",
          currIter, "\n", sep = ""))
     cat ("Apply bic.surv now\n")
}

ret.bic.surv <- bic.surv (x=curr.mat,
     surv.t=surv.time, cens=cens.vec,
     nbest=nbest, maxCol=(maxNvar+1))

if (verbose == TRUE) {
     cat ("After bic.surv\n")
}

# Get a logical vector for which probne0 < thresProbne0; i.e., get
#      a vector of variables whose probne0's are too low
rmVector <- which (ret.bic.surv$probne0 <
     thresProbne0)

if (verbose == TRUE) {
     cat("Posterior Probabilities of selected
          genes:\n")
     print (ret.bic.surv$probne0)
     cat (paste("Length of rmVector is ",
          length(rmVector), "\n", sep = ""))
}

if (any(ret.bic.surv$probne0 < thresProbne0) ==
     FALSE) {

     # No gene to swap in!! Increase threshold
     currMin <- min (ret.bic.surv$probne0)
     if (verbose ==TRUE) {
          cat (paste("No gene to swap! Min
               probne0 = ", currMin, "\n",
               sep=""))
     }

     # Assign new threshold
     newThresProbne0 <- currMin + 1
     rmVector <- which (ret.bic.surv$probne0 <
          newThresProbne0)
     if (verbose == TRUE) {
          cat (paste("New probne0 threshold =
               ", newThresProbne0, "\n",
               sep=""))
```

```
                cat ("New rmVector after applying
                increased threshold:\n")
                print (rmVector)
            }
        }

        # Now, any(ret.bic.surv$probne0 < thresProbne0) = TRUE; i.e.,
        #     there is at least 1 gene to swap, guaranteed
        if (nextVar <= ncol(x)) {
            # Set up new X
            if (verbose == TRUE) {
                cat ("Set up new X\n")
                cat (paste("nextVar is ", nextVar,
                    "\n", sep=""))
            }
            lastVar <- length(rmVector) + nextVar - 1

            # Make sure lastVar <= ncol(x)
            if (lastVar > ncol(x)) {
                rmVector <- rmVector [1: (ncol(x) -
                    nextVar + 1)]
                lastVar <- ncol(x)
            }

            # Update curr.mat
            curr.mat[,rmVector] <- x[,nextVar:lastVar]

            # Change the column names as well!!
            dimnames(curr.mat)[[2]][rmVector] <-
                dimnames(x)[[2]][nextVar:lastVar]
                nextVar <- lastVar + 1

        } else {
            # There is no variable to be removed OR exhausted all
            #     data
            stopVar <- 1
        }
        currIter <- currIter + 1
}

cat (paste(currIter, ": Explored up to variable #
    ", nextVar-1, "\n", sep=""))

# Print out selected genes if iterateBMAsurv.train has completed
if (stopVar == 1) {
    cat ("Iterate bic.surv is done!\n")
```

```
            cat("Selected genes:\n")
            print (dimnames(curr.mat)[[2]])
            cat("Posterior probabilities of selected
                 genes:\n")
            print (ret.bic.surv$probne0)
            selectedGenes <- which(ret.bic.surv$probne0 >=
                 thresProbne0)
            currfilename <- paste ("final_nbest", nbest,
                 "_adaptThres", thresProbne0, suff.string,
                 ".txt", sep="")
            write.table (curr.mat[, selectedGenes],
                 file=currfilename, sep="\t", quote=FALSE)
      }

      list(curr.mat=curr.mat, stopVar=stopVar,
            nextVar=nextVar)
}
```

Function: *iterateBMAinit*
Goal: Initialize *iterateBMAsurv.train*
Arguments: x = matrix of independent variables in the training set
(variables = columns; samples = rows)
maxNvar = size of the active *bic.surv* window (default 25
variables - may need to be reduced with smaller
training sets)
Returns: curr.mat
nextVar
stopVar

```
iterateBMAinit <- function (x, maxNvar = 25) {

      maxNvar <- min (maxNvar, ncol(x))
      curr.mat <- x[, 1:maxNvar]
      stopVar <- 0
      nextVar <- maxNvar + 1

      list(curr.mat=curr.mat, stopVar=stopVar,
            nextVar=nextVar)
}
```

Function: *iterateBMAsurv.train.wrapper*
Goal: Call *iterateBMAinit* and then *iterateBMAsurv.train* once
Caution: Make sure system has enough memory and that the max number
of iterations (*maxIter*) is high enough
Arguments: x = matrix of independent variables in the training set
(variables = columns; samples = rows)

```
#             surv.time = vector of survival times for all samples in the training
#                         set
#             cens.vec = vector of censor data for all samples in the training set
#                         (0 = censored, 1 = uncensored)
#             nbest = number specifying the number of models of each size
#                       returned to bic.surv in the BMA package (default 10)
#             maxNvar = size of the active bic.surv window (default 25
#                         variables - may need to be reduced with smaller
#                         training sets)
#             maxIter = max number of iterations in repeating bic.surv (default
#                       200000)
#             thresProbne0 = threshold to remove genes with low probne0
#                               (default 1%)
#             verbose = a boolean variable indicating whether or not to print
#                       interim information to the console (default FALSE)
# Returns:    A list of 2 components:
#             obj = an object of class bic.surv if the algorithm finishes
#             curr.names = dimnames of curr.mat; i.e., input variable names to
#                               be passed to bic.surv
#             otherwise, returns -1

iterateBMAsurv.train.wrapper <- function (x, surv.time,
     cens.vec, nbest=10, maxNvar=25, maxIter=200000,
     thresProbne0=1, verbose = FALSE, suff.string="") {

     # Get the top maxNvar variables
     ret.bma.init <- iterateBMAinit (x, maxNvar)

     # Call bic.surv repeatedly
     ret.bma <- iterateBMAsurv.train (x, surv.time,
          cens.vec, curr.mat=ret.bma.init$curr.mat,
          stopVar=ret.bma.init$stopVar,
          nextVar=ret.bma.init$nextVar, nbest, maxNvar,
          maxIter, thresProbne0,   verbose = verbose,
          suff.string)

     if (ret.bma$stopVar == 1) {
          # Apply bic.surv again using selected genes
          ret.bic.surv <- bic.surv (x=ret.bma$curr.mat,
               surv.t=surv.time, cens=cens.vec,
               nbest=nbest, maxCol=(maxNvar+1))
          return (list (obj=ret.bic.surv, curr.names=
               dimnames(ret.bma$curr.mat)[[2]]))
     } else {return (-1)}
}
```

```
# Function:    iterateBMAsurv.train.predict.assess
# Goal:        Sort genes in loglik descending order, run iterative bic.surv,
#              predict risk scores in the test set, assign risk categories for test
#              samples, and graph the results

# Arguments: train.dat, test.dat = unsorted training and testing data matrices
#                                  respectively
#            surv.time.train, surv.time.test = survival times for training and test
#                                              sets respectively
#            cens.time.train, cens.time.test = censor vectors for training and
#                                              testing sets respectively
#                                              (0=censored, 1=uncensored)
#            p = number of top genes to be used in iterate bic.surv (default
#                100)
#            nbest = number specifying the number of models of each size
#                    returned to bic.surv in the BMA package (default 10)
#            maxNvar = size of the active bic.surv window (default 25
#                      variables - may need to be reduced with smaller
#                      training sets)
#            maxIter = max number of iterations in repeating bic.surv (default
#                      200000)
#            thresProbne0 = threshold to remove genes with low probne0
#                           (default 1%)
#            cutPoint = threshold for separating high- from low-risk groups;
#                       enter as 50 for a 50% cutoff (default 50)
#            verbose = a boolean variable indicating whether or not to print
#                      interim information to the console (default FALSE)
# Returns:   A list with 7 components:
#            nvar = number of genes after the final iteration of iterative bic.surv
#            nmodel = number of models after the final iteration of iterative
#                     bic.surv
#            ypred = predicted risk scores on the test set
#            result.table = tabulated results of the risk groups for the test set
#            statistics = survdiff object containing the outcome statistics
#            ret.obj = bic.surv object from the last iteration of bic.surv
#            success = boolean indicating whether the iterations completed

iterateBMAsurv.train.predict.assess <- function
     (train.dat, test.dat, surv.time.train,
     surv.time.test, cens.vec.train, cens.vec.test, p=100,
     nbest=10, maxNvar=25, maxIter=200000, thresProbne0=1,
     cutPoint=50, verbose = FALSE, suff.string="") {

     # 1. Sort the genes in loglik descending order using Cox Proportional
          Hazards Regression
```

```
sorted.genes <- singleGeneCoxph(train.dat,
     surv.time.train, cens.vec.train)
sorted.top.genes <- printTopGenes(sorted.genes, p,
     train.dat)
currfilename <- paste("sorted_topCoxphGenes_", p,
     ".txt", sep = "")
```

2. Re-form the training dataset to contain the top *p* variables in sorted order

```
train.dat <- read.table(currfilename, header = TRUE)
if (verbose == TRUE) {
     cat ("Right here, the training data set should
          be in sorted order:\n")
     print(train.dat[1:min(5, nrow(train.dat)),
          1:min(5, ncol(train.dat))])
}
```

3. Run iterative *bic.surv*, starting with calling the *iterativeBMAsurv.train* wrapper

```
ret.list <- iterateBMAsurv.train.wrapper x=train.dat,
     surv.time=surv.time.train, cens.vec=
     cens.vec.train, nbest, maxNvar, maxIter,
     thresProbne0, verbose = verbose, suff.string)

ret.bma <- ret.list$obj
```

4. Save the results

```
write.table (ret.bma$postprob, file=paste("postprob",
     suff.string, ".txt", sep=""), sep="\n",
     quote=FALSE, row.names=FALSE, col.names=FALSE)
write.table (ret.bma$mle, file=paste("mle",
     suff.string, ".txt", sep=""), sep="\t",
     quote=FALSE)
write.table (ret.bma$probne0, file=paste("probne0",
     suff.string, ".txt", sep=""), sep="\n",
     quote=FALSE, row.names=FALSE, col.names=FALSE)
write.table (ret.list$curr.names, file=
     paste("namesx", suff.string, ".txt",
     sep=""), sep="\n", quote=FALSE, row.names=FALSE,
     col.names=FALSE)
write.table (ret.bma$which, file=paste("which",
     suff.string, ".txt", sep=""), sep="\t",
     quote=FALSE)
```

```
# 5. Predict risk scores in the test set

selected.genes <- ret.list$curr.names
     [ret.bma$probne0 > 0]
curr.test.dat <- test.dat [, selected.genes]
y.pred.test <- apply (curr.test.dat, 1,
     predictBicSurv, postprob.vec=ret.bma$postprob,
     mle.mat=ret.bma$mle)

# 6. Compute risk scores in the training set

y.pred.train <- apply (train.dat[, selected.genes],
     1, predictBicSurv, postprob.vec=
     ret.bma$postprob, mle.mat=ret.bma$mle)

# 7. Assign risk categories for test samples

ret.table <- predictiveAssessCategory (y.pred.test,
     y.pred.train, cens.vec.test, cutPoint)
cat (paste("# selected genes = ",
     length(selected.genes), "\n", sep=""))
cat (paste("# selected models = ",
     length(ret.bma$postprob), "\n", sep=""))
cat ("Risk Table:\n")
print (ret.table$assign.risk)

risk.groups = ret.table$groups
if (verbose == TRUE) {
     cat("Risk.groups -- will be error if all
          samples are assigned to the same risk
          group or all samples are in the same censor
          category.\n")
     print(risk.groups)
}

# 8. Create a survival object from the test set and graph the results

mySurv.obj <- Surv(surv.time.test, cens.vec.test)
km.fit <- survfit(mySurv.obj ~ unlist(risk.groups))
strata.levels <- summary(km.fit)$strata
filenameThisRun <- paste("km_p_", p, "_nbest_",
     nbest,"_cutPoint_", cutPoint, "_maxNvar_",
     maxNvar, ".pdf", sep="")
```

```
    if ((nrow(ret.table$assign.risk) == 2) &&
        (ncol(ret.table$assign.risk) == 2)) {

        # There are both high- and low-risk and censored/uncensored
        #     samples in the test set; get stats and graph differences
        stats <- survdiff(mySurv.obj ~
            unlist(risk.groups))
        print(stats)
        print(stats$var)
        print(stats$chisq)
        pvalue <- 1 - pchisq(stats$chisq, 1)

        # Graph
        pdf(filenameThisRun)
        plot(km.fit, col = c("red", "blue"), main =
            filenameThisRun, xlab = "Survival Time",
            ylab = "Survival Probability")
        legend("bottomleft", legend =
            c(levels(strata.levels)[1],
            levels(strata.levels)[2]), fill = c("red",
            "blue"))
        legend("topright", legend = paste("p=", pvalue,
            sep=""))
        dev.off()

        list (nvar=length(selected.genes),
            nmodel=length(ret.bma$postprob),
            ypred=y.pred.test,
            result.table=ret.table$assign.risk,
            statistics=stats, ret.obj=ret.bma,
            success=TRUE)
    } else {
        # All samples in the test set were assigned as either high-risk or
        #     low-risk or all samples were in the same censor category
        cat("ERROR: All test samples were assigned to
            one risk group or all samples
            were in the same censor category.\n")
        list(success=FALSE)
    }
}

# Function:    predictiveAssessCategory
# Goal:        To assess predictive discrimination of bic.surv results by
#              assigning samples to 2 discrete risk categories: high and low risk
# Arguments: y.pred.test = predicted risk scores on the test samples
#              y.pred.train = computed risk scores on the training samples
```

```
#             cens.vec.test = censored vector on the test set
#             cutPoint = threshold percentage for dividing high- from low-risk
#             (default 50)
# Returns:    assign.risk = the tabulated results of risk categories (row) by
#                   censored vector
#             groups = a list of each patient samples paired with its “high risk”
#                   or “low risk” designation

predictiveAssessCategory <- function (y.pred.test,
     y.pred.train, cens.vec.test, cutPoint=50) {

     # Define high- and low-risk groups by the empirical argument cutoff point
     #       of the risk score from the training set
     # Low-risk group: [min(y.pred.train), ret.quantile[1])
     # High-risk group: [ret.quantile[1], max(y.pred.train)]
     ret.quantile <- quantile (y.pred.train,
          probs=c(cutPoint/100, 1), na.rm = TRUE)


     # Assign each sample to a risk group
     risk.gp.test <- lapply (y.pred.test, assignRiskGroup,
          quantile.cutPoint=ret.quantile[1])

     # Tabulate results:
     risk.table <- table (as.vector(risk.gp.test,
          mode="character"), cens.vec.test)

     list(assign.risk=risk.table, groups=risk.gp.test)
}

# Function:   assignRiskGroup
# Goal:       For a given value x, decide which of 2 risk groups x belongs to
# Arguments: x = current sample whose risk group is being assessed
#             quantile.cutPoint = the real-number risk score threshold
# Returns:    0 for low risk group if x < quantile.cutPoint
#             1 for hi risk group if x >= quantile.cutPoint

assignRiskGroup <- function (x, quantile.cutPoint) {
     if (x < quantile.cutPoint) {
          return ("Low Risk")
     }
     else {
          return ("High Risk")
     }
}
```

CVSurvivalCode.R

```
# Authors: Amalia Annest and Ka Yee Yeung

# Function:    crossVal
# Goal:        Complete k runs of n-fold cross validation on a training dataset
# Requires     BMA, iterativeBMA, and iterativeBMAsurv libraries
# Arguments: exset = matrix of independent variables in the training set
#                    (variables = columns; samples = rows)
#            survTime = vector of survival times for all samples in the training
#                       set
#            censor = vector of censor data for all samples in the training set
#                     (0 = censored, 1 = uncensored)
#            diseaseType = string denoting the type of disease in the training
#                          dataset (used for writing to file; default "cancer")
#            nbest = number specifying the number of models of each size
#                    returned to bic.surv in the BMA package (default 10)
#            maxNvar = size of the active bic.surv window (default 25 - may
#                      need to be reduced with smaller training sets)
#            p = number of top-ranked genes to be included in the iterations
#                (default 100)
#            cutPoint = threshold for separating high- from low-risk groups;
#                       enter as 50 for a 50% cutoff (default 50)
#            verbose = a boolean variable indicating whether or not to print
#                      interim information to the console (default FALSE)
#            noFolds = the number of folds in each cross validation run
#                      (default 10)
#            noRuns = the number of cross validation runs (default 10)
# Output:    files giving fold results, run results, stats-by-fold results, and
#            average-over-all-runs results

crossVal <- function(exset, survTime, censor, diseaseType=
     "cancer", nbest=10, maxNvar=25, p=100,  cutPoint=50,
     verbose = FALSE, noFolds=10, noRuns=10) {

     # Initialize matrices and filenames
     cv <- crossVal.init(exset, noFolds, noRuns, nbest,
          maxNvar, p, cutPoint)

     # Perform cross validation runs from 1:noRuns
     for (i in 1:noRuns) {
          cv.run <- crossVal.run(exset, survTime, censor,
               diseaseType, nbest, maxNvar, p, cutPoint,
               cv$rp.mat, cv$overall.average.results,
```

```
            cv$p.val.result.mat, cv$chi.sq.result.mat,
            noFolds, cv, run=i, verbose=verbose)
        cv$overall.average.results <-
            cv.run$overall.average.results
        cv$rp.mat <- cv.run$rp.mat
        cv$p.val.result.mat <- cv.run$p.val.result.mat
        cv$chi.sq.result.mat <- cv.run$chi.sq.result.mat
}

# Calculate overall average result matrix and final uncensored
#       percentage
calc <- crossVal.final.calc
        (cv$overall.average.results, noRuns)
cv$overall.average.results <-
        calc$overall.average.results

# Make sure there are no zero-values in the p-value and chi-square
#       matrices for mean/sd calculations
non.zero.p.val.mat <-
        cv$p.val.result.mat[cv$p.val.result.mat > 0]
non.zero.chi.square.mat <-
        cv$chi.sq.result.mat[cv$chi.sq.result.mat > 0]

# Print and write final overall CV results to file
cat("Overall average result matrix over all runs:\n")
print(cv$overall.average.results)
write.table("Overall average results across all CV
        runs:", cv$filenamerun, append=TRUE,
        quote=FALSE)
write.table(cv$overall.average.results,
        cv$filenamerun, append=TRUE, quote=FALSE,
        row.names=TRUE, col.names=TRUE, sep="\t")

# Print and write final overall p-value and chi-square statistics to file
cat("Average p-value across all folds and all
        runs:\n")
print(mean(non.zero.p.val.mat))
cat("Standard deviation of p-values across all
        folds and all runs:\n")
print(sd(non.zero.p.val.mat))
cat("Average chi-square value across all folds and
        all runs:\n")
print(mean(cv$chi.sq.result.mat))
cat("Standard deviation of chi-square across all
        folds and all runs:\n")
print(sd(non.zero.chi.square.mat))
```

```
        statsfilename <- paste("avg_p_value_chi_square_", p,
             "_genes_nbest_", nbest, "_maxNvar_", maxNvar,
             "_cutPoint_", cutPoint, ".txt")
        write.table("P-value matrix:", statsfilename, quote =
             FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table(cv$p.val.result.mat, statsfilename, quote
             = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table("Mean:", statsfilename, quote = FALSE,
             append = TRUE, row.names = FALSE, col.names =
             FALSE)
        write.table(mean(non.zero.p.val.mat), statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table("Standard Deviation:", statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table(sd(non.zero.p.val.mat), statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table("Chi-Square Matrix:", statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table(cv$chi.sq.result.mat, statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table("Mean:", statsfilename, quote = FALSE,
             append = TRUE, row.names = FALSE, col.names =
             FALSE)
        write.table(mean(non.zero.chi.square.mat),
             statsfilename, quote = FALSE, append = TRUE,
             row.names = FALSE, col.names = FALSE)
        write.table("Standard Deviation:", statsfilename,
             quote = FALSE, append = TRUE, row.names = FALSE,
             col.names = FALSE)
        write.table(sd(non.zero.chi.square.mat),
             statsfilename, quote = FALSE, append = TRUE,
             row.names = FALSE, col.names = FALSE)
}

# Function:    crossVal.init
# Goal:        Initialize matrices and file names for cross validation runs
# Arguments: exset = matrix of independent variables in the training set
#                    (variables = columns; samples = rows)
#            noFolds = the number of folds in each cross validation run
```

```
#                       (default 10)
#             noRuns = the number of cross validation runs (default 10)
#             nbest = number specifying the number of models of each size
#                     returned to bic.surv in the BMA package (default 10)
#             maxNvar = size of the active bic.surv window (default 25 - may
#                       need to be reduced with smaller training sets)
#             p = number of top-ranked genes to be included in the iterations
#                 (default 100)
#             cutPoint = threshold for separating high- from low-risk groups;
#                       enter as 50 for a 50% cutoff (default 50)
# Returns:    noSamples = number of samples in the training set
#             noGenes = number of genes in the training set
#             rp.mat = matrix to store the permutation order for each run
#             overall.average.results = matrix to store the overall high-risk/low-
#                                       risk life/death information
#             p.val.result.mat = matrix to store p-value results for all runs
#             chi_sqaure_result_matrix = matrix to store chi square results for
#                                        all runs
#             filenamerun = filename for the run results
#             filenamefold = filename for the fold results

crossVal.init <- function(exset, noFolds, noRuns, nbest,
     maxNvar, p, cutPoint) {

     # Save the number of patient samples and the number of genes
     noSamples <- nrow(exset)
     noGenes <- ncol(exset)

     # Random permutation matrix - CV runs are rows and samples are
     #      columns
     rp.mat <- mat.or.vec(noRuns, noSamples)

     # Average results matrix to average over all runs
     overall.average.results <- mat.or.vec(2, 3)
     rownames(overall.average.results) <- c("Low Risk",
          "High Risk")
     colnames(overall.average.results) <- c("Censored",
          "Uncensored", "Percent Uncensored")

     # Matrices to store p-values and chi-square statistics - CV runs are rows
     #      and folds are columns
     p.val.result.mat <- mat.or.vec(noRuns, noFolds)
     chi.sq.result.mat <- mat.or.vec(noRuns, noFolds)

     # Create filenames to store run- and fold-results
```

```
        filenamerun = paste("runresults_", p,
             "_genes_nbest_", nbest, "_maxNvar_", maxNvar,
             "_cutPoint_", cutPoint, ".txt")
        filenamefold = paste("foldresults_", p,
             "_genes_nbest_", nbest, "_maxNvar_", maxNvar,
             "_cutPoint_", cutPoint, ".txt")

        list(noSamples=noSamples, noGenes=noGenes,
             rp.mat=rp.mat, overall.average.results=
             overall.average.results, p.val.result.mat=
             p.val.result.mat, chi.sq.result.mat=
             chi.sq.result.mat, filenamerun=filenamerun,
             filenamefold=filenamefold)
}

# Function:    crossVal.run
# Goal:        Perform a single run of cross validation
# Arguments: exset = matrix of independent variables in the training set
#              (variables = columns; samples = rows)
#              survTime = vector of survival times for all samples in the training
#                         set
#              censor = vector of censor data for all samples in the training set
#                       (0 = censored, 1 = uncensored)
#              diseaseType = string denoting the type of cancer in the training
#                            dataset (used for writing to file; default "cancer")
#              nbest = number specifying the number of models of each size
#                      returned to bic.surv in the BMA package (default 10)
#              maxNvar = size of the active bic.surv window (default 25 - may
#                        need to be reduced with smaller training sets)
#              p = number of top-ranked genes to be included in the iterations
#                  (default 100)
#             cutPoint = threshold for separating high- from low-risk groups;
#                        enter as 50 for a 50% cutoff (default 50)
#              rp.mat = matrix for storing the random permutations
#              overall.average.results = matrix for storing high- and low-risk
#                                        samples paired with censored/
#                                        uncensored data across all runs
#              p.val.result.mat = matrix for holding the p-value from each fold
#              chi.sq.result.mat = matrix for holding the chi-square statistic from
#                                  each fold
#              noFolds = the number of folds in each cross validation run
#              cv = the cross validation initialization object
#              run = keeps track of what run is currently being executed (from
#                    1:noRuns)
#              verbose = a boolean variable indicating whether or not to print
#                        interim information to the console (default FALSE)
```

```
# Returns:     updated overall.average.results
#              updated rp.mat
#              updated p.val.result.mat
#              updated chi.square.result.mat

crossVal.run <- function(exset, survTime, censor,
     diseaseType, nbest, maxNvar, p, cutPoint, rp.mat,
     overall.average.results, p.val.result.mat,
     chi.sq.result.mat, noFolds, cv, run, verbose = FALSE)
{

     cat (paste("******************** BEGINNING CV RUN
          ", run, " ********************\n", sep = ""))

     # Hold prediction results for this CV run
     single.run.result.mat <- mat.or.vec(2, 3)
     rownames(single.run.result.mat) <- c("Low Risk",
          "High Risk")
     colnames(single.run.result.mat) <- c("Censored",
          "Uncensored", "Percent Uncensored")

     # Generate a random permutation for current CV run
     rp <- sample(1:cv$noSamples)

     # Add permutation to the random permuation matrix
     for (j in 1:cv$noSamples) {
          rp.mat[run, j] <- rp[j]
     }
     if (verbose == TRUE) {
          cat ("The random permutation matrix looks like
               this so far:\n")
          print (rp.mat[1:run ,])
     }

     # Permute the samples
     cvpass <- exset[rp ,]
     if (verbose == TRUE) {
          cat ("Here are a few samples and genes from
               the permuted data:\n")
          print (cvpass[1:min(5, cv$noSamples), 1:min(5,
               cv$noGenes)])
          cat ("Here are the row names in permuted
               order:\n")
          print (rownames(cvpass))
     }
```

```
# Permute the survival time info
survtimepass <- survTime[rp]
if (verbose == TRUE) {
     cat ("Here are the survival times in permuted
          order:\n")
     print (survtimepass)
}

# Permute the censor info
censorpass <- censor[rp]
if (verbose == TRUE) {
     cat ("Here is the censor data in permuted
          order:\n")
     print (censorpass)
}

lastSample <- 0

# Perform the cross validation folds for this run
for (k in 1:noFolds) {
     cv.folds <- crossVal.fold(cvpass, survtimepass,
          censorpass, diseaseType, nbest, maxNvar, p,
          cutPoint, lastSample, noFolds,
          single.run.result.mat, p.val.result.mat,
          chi.sq.result.mat, cv, fold=k, run, verbose
          = verbose)
     lastSample <- cv.folds$lastSample
     single.run.result.mat <-
          cv.folds$single.run.result.mat
     p.val.result.mat <- cv.folds$p.val.result.mat
     chi.sq.result.mat <- cv.folds$chi.sq.result.mat
}

cat (paste("******************** END CV RUN ", run,
     " ********************\n", sep = ""))

# Print results from this run and write to file
cat("Overall results from this run:\n")
print(single.run.result.mat)
write.table(paste("Results from run ", run, ":", sep
     = ""), cv$filenamerun, append=TRUE, quote=FALSE)
write.table(single.run.result.mat, cv$filenamerun,
     append=TRUE, quote=FALSE, row.names=TRUE,
     col.names=TRUE, sep="\t")
```

```
# Add this run's results to the overall average result matrix
      for (v in 1:2) {
            for (w in 1:2) {
                  overall.average.results[v, w] <-
                        overall.average.results[v, w] +
                        single.run.result.mat[v, w]
            }
      }

      list(overall.average.results=overall.average.results,
            rp.mat=rp.mat, p.val.result.mat=
            p.val.result.mat, chi.sq.result.mat.result=
            chi.sq.result.mat)
}

# Function:    crossVal.fold
# Goal:        Perform a single fold of a cross validation run
# Arugments: cvpass = the permuted matrix of independent variables for this
#                     fold
#            survtimepass = the permuted vector of survival times for this fold
#            censorpass = the permuted vector of censor data for this fold
#            diseaseType = string denoting the type of cancer in the training
#                          dataset (used for writing to file; default "cancer")
#            nbest = number specifying the number of models of each size
#                    returned to bic.surv in the BMA package (default 10)
#            maxNvar = size of the active bic.surv window (default 25 - may
#                      need to be reduced with smaller training sets)
#            p = number of top-ranked genes to be included in the iterations
#                (default 100)
#            cutPoint = threshold for separating high- from low-risk groups;
#                       enter as 50 for a 50% cutoff (default 50)
#            lastSample = number representing the last sample in the test set
#                         for this fold
#            noFolds = the number of folds in each cross validation run
#            single.run.result.mat = matrix for holding the results of a single
#                                    run of cross validation
#            p.val.result.mat = matrix for holding the p-value from each fold
#            chi.square.result.mat = matrix for holding the chi-square statistic
#                                    from each fold
#            cv = the cross validation initialization object
#            fold = keeps track of what fold is currently being executed (from
#                   1:noFolds)
#            run = keeps track of what run is currently being executed (from
#                   1:noRuns)
#           verbose = a boolean variable indicating whether or not to print
#                     interim information to the console (default FALSE)
```

```
# Returns:     updated lastSample
#              updated single.run.result.mat
#              updated p.val.result.mat
#              updated chi.square.result.mat

crossVal.fold <- function(cvpass, survtimepass,
     censorpass, diseaseType, nbest, maxNvar, p, cutPoint,
     lastSample, noFolds, single.run.result.mat,
     p.val.result.mat, chi.sq.result.mat, cv, fold, run,
     verbose = FALSE) {


     cat (paste("---------- BEGINNING FOLD ", fold, " --
          --------\n", sep = ""))

     # Pick the correct "chunk" size for this fold
     newLastSample <- floor((cv$noSamples/noFolds) * fold)
     startSample <- lastSample + 1

     # Assign training sets
     # Test set is at the very beginning of the data
     if (lastSample == 0) {
          train <- cvpass[(newLastSample +
               1):cv$noSamples,]
          trainSurv <- survtimepass[(newLastSample +
               1):cv$noSamples]
          trainCens <- censorpass[(newLastSample +
               1):cv$noSamples]
     }

     # Test set is at the very end of the data
     else if (newLastSample == cv$noSamples) {
          train <- cvpass[1:(startSample - 1) ,]
          trainSurv <- survtimepass[1:(startSample - 1)]
          trainCens <- censorpass[1:(startSample - 1)]
     }

     # Test set is somewhere in the middle of the data
     else {
          train <- cvpass[c(1:lastSample, (newLastSample +
               1):cv$noSamples) ,]
          trainSurv <- survtimepass[c(1:lastSample,
               (newLastSample + 1):cv$noSamples)]
          trainCens <- censorpass[c(1:lastSample,
               (newLastSample + 1):cv$noSamples)]
     }
```

```
# Assign test sets
test <- cvpass[startSample:newLastSample ,]
testSurv <- survtimepass[startSample:newLastSample]
testCens <- censorpass[startSample:newLastSample]
lastSample <- newLastSample

# Print info
if (verbose == TRUE) {
     cat("Row names for training set:\n")
     print(rownames(train))
     cat("Row Names for test set:\n")
     print(rownames(test))
     cat("Training survival time set:\n")
     print(trainSurv)
     cat("Testing survival time set:\n")
     print(testSurv)
     cat("Training censor set:\n")
     print(trainCens)
     cat("Testing censor set:\n")
     print(testCens)
     cat("Total censor set in case you want to
          check the order:\n")
     print(censorpass)
}

# Call the iterative BMA survival algorithm
curr.string <- paste ("_", diseaseType, "_",
     cv$noGenes, sep="")
ret.bma <- iterateBMAsurv.train.predict.assess
     (train.dat=train, test.dat=test,
     surv.time.train=trainSurv,
     surv.time.test=testSurv,
     cens.vec.train=trainCens,
     cens.vec.test=testCens, nbest=nbest,
     maxNvar=maxNvar, p=p, maxIter=200000,
     thresProbne0=1, cutPoint=cutPoint,
     verbose=verbose, suff.string=curr.string)

# Only calculate stats for this fold if both risk groups and both censor
#      categories were present in test set assignments
if(ret.bma$success==TRUE) {
     # Get results from iterative BMA survival algorithm
     results <- ret.bma$result.table
     statistics <- ret.bma$statistics

     # Calculate the p-value
```

```
pvalue <- 1 - pchisq(statistics$chisq, 1)
statsPerFold <- paste("Stats for fold ", fold, "
     in run ", run)
statsdoc <- paste("stats_", p, "_genes_nbest_",
     nbest, "_maxNvar_", maxNvar, "_cutPoint_",
     cutPoint, ".txt")

# Add p-value and chi-square statistics to their corresponding
#      matrices
p.val.result.mat[run,fold] <- pvalue
chi.sq.result.mat[run,fold] <- statistics$chisq

# Write p-value and statistics to file
write.table(statsPerFold, statsdoc, append =
     TRUE, quote = FALSE)
write.table(as.matrix(statistics), statsdoc,
     append = TRUE, quote = FALSE)
write.table("P-Value:", statsdoc, append = TRUE,
     quote = FALSE)
write.table(pvalue, statsdoc, append = TRUE,
     quote = FALSE)

# Add the current results table to the single CV run table and
#      calculate death percentages
cv.tab <- crossVal.tabulate(results,
     single.run.result.mat)
single.run.result.mat <-
     cv.tab$single.run.result.mat

# Print results and write to file
if (verbose == TRUE) {
     cat("Results from this fold:\n")
     print(results)
     cat("Accumulated results from this run so
          far:\n")
     print(single.run.result.mat)
}
write.table(paste("Results from fold ", fold, "
     in run ", run, ":", sep = ""),
     cv$filenamefold, append=TRUE, quote=FALSE)
     write.table(results, cv$filenamefold,
     append=TRUE, quote=FALSE, row.names=TRUE,
     col.names=TRUE, sep="\t")
write.table("Accumulated results from this run
     so far:", cv$filenamefold, append=TRUE,
     quote=FALSE)
```

```
            write.table(single.run.result.mat,
                  cv$filenamefold, append=TRUE, quote=FALSE,
                  row.names=TRUE, col.names=TRUE, sep="\t")
      }

      list(lastSample=lastSample,
            single.run.result.mat=single.run.result.mat,
            p.val.result.mat=p.val.result.mat,
            chi.sq.result.mat=chi.sq.result.mat)
}
```

Function: *crossVal.tabulate*
Goal: Tabulate results for a single CV fold and add them to the matrix
for the current CV run
Arguments: results = table from the current fold indicating how many high-
and low-risk patient samples lived and died
single.run.result.mat = matrix to hold results from a single CV run
Returns: updated single.run.result.mat

```
crossVal.tabulate <- function(results,
      single.run.result.mat) {

      # Add the current results table to the single CV run table
      for (m in 1:nrow(results)) {
            for (b in 1:2) {
                  if (dimnames(results) [[1]][m]==
                        dimnames(single.run.result.mat)
                        [[1]][b]) {
                        if (ncol(results) == 2) {
                              for (n in 1:2) {
                                    single.run.result.mat[b,n]
                                    <-
                                    single.run.result.mat[b,n] +
                                    results[m,n]
                              }
                        }
                        else if (dimnames(results)[[2]][1] ==
                              "0") {
                              single.run.result.mat[b, "Life"]
                              <- single.run.result.mat[b,
                              "Life"] + results[m, 1]
                        }
                        else if (dimnames(results)[[2]][1] ==
                              "1") {
```

```
                              single.run.result.mat[b, "Death"]
                              <- single.run.result.mat[b,
                              "Death"] + results[m, 1]
                         }
                         else {
                              cat("This should never
                              happen!!!\n")
                         }
                    }
               }
          }

          # Calculate the death percentages
          for (q in 1:2) {
               total <- 0
               for (r in 1:2) {
                    total <- total + single.run.result.mat[q,
                         r]
               }
               single.run.result.mat[q, 3] <-
                    100*(single.run.result.mat[q, 2]/total)
          }

          list(single.run.result.mat=single.run.result.mat)
}

# Function:    crossVal.final.calc
# Goal:        Carry out final calculations for the overall average results matrix
# Arguments: overall.average.results = matrix for storing high- and low-risk
#              samples paired with censored/uncensored data across all runs
#              noRuns = the number of cross validation runs
# Returns:     updated overall.average.results

crossVal.final.calc <- function(overall.average.results,
     noRuns) {

     # Calculate overall average result matrix
     for (risk in 1:2) {
          for (survival in 1:2) {
               overall.average.results[risk, survival] <-
                    overall.average.results[risk,
                    survival]/noRuns
          }
     }

     # Calculate overall death percentage
```

```
    for (risk.rows in 1:2) {
         total <- 0
         for (surv in 1:2) {
              total <- total +
                   overall.average.results[risk.rows,
                   surv]
         }
         overall.average.results[risk.rows, 3] <-
              100*(overall.average.results[risk.rows,
              2]/total)
    }

    list(overall.average.results=overall.average.results)
}
```

singleGeneCoxph.R

```
# Authors: Amalia Annest and Ka Yee Yeung

# Function:    singleGeneCoxph
# Goal:        Fit a cox PH model for each gene in the breast cancer data
#              Rank each gene using loglik (higher the better)
# Requires:    Survival library
# Arguments: trainData = matrix of independent variables in the training set
#              (variables = columns; samples = rows)
#              survData = vector of survival times for all samples in the training
#                         set
#              censoredData = vector of censor data for all samples in the
#              training set (0 = censored, 1 = uncensored)
# Output:      print out the loglik for each gene (sorted in descending order)
# Returns:     the sorted matrix, 1st column = sorted loglik, 2nd column =
#              original index in the trainData

singleGeneCoxph <- function(trainData, survData,
     censoredData) {

     # Create a survival object
     mySurvObj <- Surv (survData, censoredData)

     # Fit the coxph model for each gene and save the results
     loglikVec <- rep (0, length=ncol(trainData))

     for (i in 1:ncol(trainData)) {
          curr.fit <- coxph (mySurvObj ~ trainData[, i])
          loglikVec[i] <- curr.fit$loglik[2]
     }

     names (loglikVec) <- dimnames(trainData)[[2]]

     # Sort the log likelihood in descending order
     sortedLogLikVec <- sort (loglikVec, decreasing =
          TRUE, index = TRUE)

     # Results matrix to print out
     retMatrix <- matrix (0, nrow=ncol(trainData), ncol=2)
     retMatrix [, 1] <- sortedLogLikVec$x
     retMatrix [, 2] <- sortedLogLikVec$ix
     dimnames(retMatrix) <- list(names
          (sortedLogLikVec$x), c("sorted numbers",
          "original index"))
```

```
     write.table (retMatrix, file="sorted_loglik.txt",
          quote=FALSE, sep="\t")

     return (retMatrix)
}

# Function:    printTopGenes
# Goal:        Given the sorted return matrix, print out the top G genes
# Arguments: retMatrix = the sorted return matrix
#              numGlist = list of acceptable values for top G genes
#              trainData = matrix of independent variables in the training set
#              (variables = columns; samples = rows)
# Output:      a printed list of the top G genes

printTopGenes <- function(retMatrix, numGlist=c(10, 30,
     50, 100, 500, 1000, ncol(trainData)), trainData,
     myPrefix="sorted_topCoxphGenes_") {

     # Get all the samples (rows), with the genes sorted
     sorted.data <- trainData[, retMatrix[, 2]]

     # Write the sorted genes to file
     for (G in numGlist) {
          curr.data <- sorted.data[, 1:G]
          currfilename <- paste (myPrefix, G, ".txt",
               sep="")
          write.table (curr.data, file=currfilename,
               quote=FALSE, sep="\t")
     }
}
```

predict_bicSurv.R

```
# Authors: Amalia Annest and Ka Yee Yeung

# Function:    predictBicSurv
# Goal:        Compute the risk score of a given patient sample
# Arguments: newdata.vec = vector of test data (1 observation)
#              postprob.vec = postprob from bic.surv
#              mle.mat = mle matrix from bic.surv (models = rows, variables =
#                        columns)
# Output:      A patient sample risk score

predictBicSurv <- function(newdata.vec, postprob.vec,
     mle.mat) {

     # First compute the risk score for each of the models
     # Risk score for each model = sum(coefficient in model k * corresponding
     #      expression level in newdata.vec)
     risk.score.vec <- apply (mle.mat, 1, function(x)
          sum(x * newdata.vec))

     # Patient sample risk score under BMA = sum (postprob for model k *
     #      risk score for model k) over all the selected models
     retprob <- sum (postprob.vec * risk.score.vec)
     retprob
}
```

imageplot_iterate_bma_surv.R

```
# Authors: Amalia Annest and Ka Yee Yeung

# Function:    imageplot.iterate.bma.surv
# Goal:        To plot image plot without variables with probne0 == 0
#              and variables sorted by probne0 in descending order
# Arguments: Same as imageplot.bma
# Calls:       imageplot.bma

imageplot.iterate.bma.surv <- function (bicreg.out,
     color="default", ...) {

     # Get genes with non-zero probne0
     selected.genes <- bicreg.out$namesx
          [bicreg.out$probne0 > 0]
     selected.probne0 <- bicreg.out$probne0
          [bicreg.out$probne0 > 0]

     # Sort by probne0
     sorted.vec <- sort (selected.probne0, decreasing =
          TRUE, index =  TRUE)
     sorted.genes <- selected.genes [sorted.vec$ix]

     # Modify components in bicreg.out: namesx, probne0, n.vars, which
     bicreg.mod.out <- bicreg.out
     bicreg.mod.out$namesx <- sorted.genes
     bicreg.mod.out$probne0 <- sorted.vec$x
     bicreg.mod.out$n.vars <- length (sorted.genes)

     # DO NOT sort the columns in which
     bicreg.mod.out$which <- bicreg.out$which [,
          bicreg.out$probne0 > 0]

     if(class(bicreg.mod.out$which) == "logical") {
          bicreg.mod.out$which <-
               bicreg.mod.out$which[sorted.vec$ix]
     } else {
          bicreg.mod.out$which <- bicreg.mod.out$which
               [, sorted.vec$ix]
     }

     # Calls imageplot.bma with the modified argument
     imageplot.bma.mod (bicreg.mod.out, color=color)
}
```

```
# Function: imageplot.bma.mod
# Code copied from imageplot.bma from the BMA package with a few
#       modifications: margin of plot and title of plot

imageplot.bma.mod <- function (bicreg.out, color =
     "default", ...)  {

     keep.mar <- par()$mar
     par(mar = c(5, 8, 4, 2) + 0.1)
     which.out <- bicreg.out$which
     nvar <- 0
     nmodel <- 0

     if(class(which.out)=="logical") {
          nvar <- length(which.out)
          nmodel <- 1
     } else {
          nvar <- ncol(which.out)
          nmodel <- nrow(which.out)
     }

     filename <- paste("models=", nmodel, "genes=", nvar,
          ".pdf", sep="")

     if (color == "default") {
          if(class(which.out)=="logical") {
               which.out <- as.matrix(which.out)
               which.out <- t(which.out)
               pdf(filename)
               par(las = 1)
               image(cumsum(bicreg.out$postprob), 1:nvar,
                    -which.out[1, nvar:1, drop = FALSE],
                    xlab = "Model #", ylab = "", xaxt =
                    "n", yaxt = "n", xlim = c(0, 1),
                    main = "Models selected by
                    iterativeBMAsurv", ...)
          } else {
               pdf(filename)
               par(las = 1)
               image(c(0, cumsum(bicreg.out$postprob)),
                    1:nvar, -which.out[1:nmodel,
                    nvar:1, drop = FALSE], xlab = "Model
                    #", ylab = "", xaxt = "n",
                    yaxt = "n", xlim = c(0, 1), main =
                    "Models selected by iterativeBMAsurv",
```

```
                    ...)
          }
     }

     if (color == "blackandwhite") {
          if(class(which.out)=="logical") {
               which.out <- as.matrix(which.out)
               which.out <- t(which.out)
               pdf(filename)
               par(las = 1)
               image(cumsum(bicreg.out$postprob), 1:nvar,
                    -which.out[1, nvar:1, drop = FALSE],
                    xlab = "Model #", ylab = "", xaxt =
                    "n", yaxt = "n",
                    col = c("black", "white"), main =
                    "Models selected by BMA", ...)
          } else {
               pdf(filename)
               par(las = 1)
               image(cumsum(bicreg.out$postprob), 1:nvar,
                    -which.out[1:nmodel, nvar:1, drop =
                    FALSE], xlab = "Model #", ylab = "",
                    xaxt = "n", yaxt = "n", col =
                    c("black", "white"), main = "Models
                    selected by BMA", ...)
          }
     }
     xat <- (cumsum(bicreg.out$postprob) + c(0,
          cumsum(bicreg.out$postprob[-nmodel])))/2
     axis(1, at = xat, labels = 1:nmodel, ...)
     axis(2, at = 1:nvar, labels = rev(bicreg.out$namesx),
          ...)
     par(mar = keep.mar)
     dev.off()
}
```