2016

Methods for Predicting an Ordinal Response with High-Throughput Genomic Data

Kyle L. Ferber
Virginia Commonwealth University

METHODS FOR PREDICTING AN ORDINAL RESPONSE WITH HIGH-THROUGHPUT GENOMIC DATA

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

KYLE L. FERBER

B.S. Mathematics, College of William and Mary, 2012

Director: Kellie J. Archer, Ph.D.,

Professor, Department of Biostatistics

Virginia Commonwealth University

Richmond, Virginia

December, 2016

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Kellie J. Archer, for being an incredible mentor to me and for serving as an example of a dedicated and successful researcher. I have benefited greatly from her intelligence, patience, and teaching and communication skills over the past three and a half years. I also want to thank her for supporting me as a Research Assistant on her NIH Research Project Grant.

I would also like to thank my committee members for their efforts and for taking the time to read my thesis and attend regular meetings. Dr. Roy Sabo and Dr. Le Kang provided great suggestions regarding the statistical aspects of my dissertation, while Dr. Michael Idowu and Dr. Colleen Jackson-Cook contributed useful insights regarding the clinical and genomic applications.

I would like to thank Russell Boyle for teaching and emphasizing valuable skills as the instructor for Biostatistical Consulting (BCL). Through that class, I have become a more effective communicator of statistical concepts to researchers without a strong quantitative background. This crucial skill is too often overlooked. Furthermore, my critical thinking and public speaking skills also grew over the eight semesters of BCL.

I also need to thank my family for their unwavering encouragement and support. The formidable first year in graduate school would have been much more difficult without weekly dinners with my brother and sister-in-law. I am also incredibly lucky to have parents that have done everything in their power to help me succeed. I am certain that I would not be here without them. Finally, my wife, Laura, has kept me afloat from day one of graduate school. Despite her own busy schedule, she has firmly committed herself to supporting me no matter what, and I will always be grateful for that.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

METHODS FOR PREDICTING AN ORDINAL RESPONSE WITH HIGH-THROUGHPUT GENOMIC DATA

By Kyle L. Ferber

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2016.

Major Director: Kellie J. Archer, Ph.D.,
Professor, Department of Biostatistics

Multigenic diagnostic and prognostic tools can be derived for ordinal clinical outcomes using data from high-throughput genomic experiments. For example, one may wish to classify tissue samples as healthy, pre-malignant, or malignant using data from a microarray experiment. Here, the goal is twofold: to develop an accurate classifier of the ordinal outcome and to select features that play an important role in the tissue's progression from a healthy to a malignant state.

A challenge in this setting is that the number of predictors is much greater than the sample size, so traditional ordinal response modeling techniques must be exchanged for more specialized approaches. Existing methods perform well on some datasets, but there is room for improvement in terms of variable selection and predictive accuracy. Therefore, we extended an impressive binary response modeling technique, Feature Augmentation via Nonparametrics and Selection (FANS), to the ordinal response setting and developed software for implementing the extension. Through simulation studies and analyses of high-throughput genomic datasets, we showed that our Ordinal FANS method is sensitive and specific when discriminating between important and unimportant features from the high-dimensional feature space and is highly competitive in terms of predictive accuracy.

Discrete survival time is another example of an ordinal response. For many illnesses and chronic conditions, it is impossible to record the precise date and time of disease onset or

relapse. Further, the HIPPA Privacy Rule prevents recording of protected health information which includes all elements of dates (except year), so in the absence of a "limited dataset", date of diagnosis or date of death are not available for calculating overall survival. Therefore, increasingly survival is collected as a discrete event time. We previously demonstrated that a penalized forward continuation ratio model can be fit to discrete survival time data in high-dimensional settings, but this model does not incorporate censoring information. Thus, we developed a method that is suitable for modeling high-dimensional discrete survival time data while accommodating censoring information and assessed its performance by conducting a simulation study and by predicting the discrete survival times of acute myeloid leukemia patients using a high-dimensional dataset.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Physicians are faced with difficult medical decisions on a daily basis. Oncologists, for instance, need to personalize treatment plans for their patients. If the desired results are not obtained after a given amount of time, they may need to consider switching therapies. These choices are made based on expert medical knowledge, intuition, and past experiences, and each decision they make will affect their patients' survival times. Consequently, two independent, equally qualified physicians may make different decisions about what is best for the well-being of a patient. A more objective method for making certain medical decisions is to use data collected on past patients to build a predictive model that can be applied to future patients.

Consider the case of a breast cancer patient with three treatment options. The physician will choose one of the three options, the patient will be given that treatment, and then the physician will assess how well the treatment worked based on characteristics of the residual disease. The success of treatment can be measured using the residual cancer burden (RCB) index, an ordinal measure (RCB-I $<$ RCB-II $<$ RCB-III) that takes into account primary tumor bed dimensions, cellularity fraction of invasive cancer, size of largest metastasis, and the number of positive lymph nodes [1]. Alternatively, data on patients who have received each of the three treatments could be used to build a regression model. Then, the patient would be given whichever treatment the model predicted will result in the best outcome. In this situation, neither the physician nor the data-driven model are going to make the best decision 100% of the time. Ayers (2007) argues this by stating "In the end, super crunching is not a substitute for intuition, but rather a complement [2]." By super crunching, he is

referring to the practice of using large amounts of data to make informed decisions. He then goes on to say, "Traditional experts make better decisions when they are provided with the results of statistical prediction. Those who cling to the authority of traditional experts tend to embrace the idea of combining the two forms of 'knowledge' by giving the experts statistical support [2]." Our goal in developing methods for fitting predictive ordinal response models is not to replace a physicians decision making, but rather to support it with additional information. The methods we developed utilize high-dimensional genomic data to predict patient outcome, whereas physicians typically make decisions based on standards of care that are often developed by clinical knowledge, not rigorous analyses of experimental data.

Predictive ordinal response models could also help in simpler situations, such as determining the grade of a particular tumor, which is a relatively subjective procedure done by examining the microscopic cell appearance. Tumor grade is taken into account when deciding on a treatment regimen, so it is important to make an accurate assessment. Gene expression can be taken into account through a predictive model to improve the accuracy of a pathologist's diagnosis [3].

1.2 Analysis of ordinal response data

An ordinal response is unique in that there is an intrinsic ordering to the possible values of the response, but the distance between these possible values, called classes, is unknown. For instance, the four stages of cancer are ordered (stage I is less severe than stage II; stage II is less severe than stage III, etc.), but we cannot objectively measure the quantitative difference in severity between each of the four stages. Thus, ordinal variables are somewhat of a hybrid between a nominal categorical variable and a continuous variable. Ordinal responses are ubiquitous in biomedical data. For example, in cancer research, discrete survival time, tumor grade, and the degree of regional lymph node involvement are all ordinal measurements. Furthermore, the severity of many conditions, such as heart failure, Alzheimer's

disease, and chronic kidney disease, are defined by an ordinal stage of disease [4, 5, 6].

1.2.1 Ordinal regression

Ignoring any aspect of the ordinal response results in a loss of information. For instance, modeling an ordinal outcome using multinomial logistic regression, given below, ignores the ordering of the classes.

$$\log\frac{P(Y_i = k|\boldsymbol{x_i})}{P(Y_i = K|\boldsymbol{x_i})} = \beta_{0k} + \boldsymbol{x_i}\boldsymbol{\beta_k}, \quad k = 1, ..., K-1$$

Given p predictors, this model estimates a different set of p coefficients for each of the $K-1$ logistic regression models, and as a result, it is difficult to interpret and is not parsimonious.

Another approach for modeling an ordinal response is to assign each class an integer rank (i.e. 1, 2, ..., K) and use traditional linear regression to model the response. However, this method makes the unrealistic assumption that the distances between adjacent ordinal classes are equal. Furthermore, assigning integer values to the ordinal classes is arbitrary. For example, one could instead assign *even* integers to the ordinal classes (2, 4, ..., $2K$). Finally, a linear regression model assumes that $Y|\boldsymbol{X}$ is normally distributed. This assumption will clearly fail when the response is ordinal because the response is discrete and predicted values are not restricted to be between 1 and K.

Proportional odds models are a class of models that are appropriate for modeling an ordinal outcome. Models in this class only estimate p coefficient estimates by taking advantage of the ordered structure and making a simple assumption. For example, for observation $i = 1, 2, ..., n$, the cumulative logit model is given by:

$$\log\frac{P(Y_i > k|\boldsymbol{x_i})}{P(Y_i \leq k|\boldsymbol{x_i})} = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}, \quad k = 1, ..., K-1$$

In the estimation procedure, a constraint is placed on the class-specific intercepts, or *thresholds*, such that $\alpha_1 < \alpha_2 < ... < \alpha_{K-1}$, which preserves the positivity of the class-specific probability estimates. The thresholds are the only outcome-specific parameters in the model,

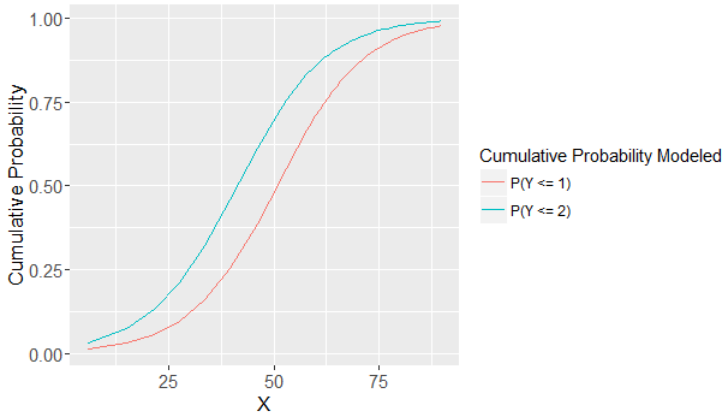

Fig. 1. A cumulative logit model with a single predictor and an ordinal outcome with $K = 3$ classes.

so $K - 1$ parallel logit models are produced. Further, this model is a type of proportional odds model because the cumulative odds ratio is proportional to the difference between two vectors of predictors. The cumulative odds ratio, which is defined as the odds of $Y > k$ given $\boldsymbol{X} = \boldsymbol{x_1}$ compared to the odds of $Y > k$ given $\boldsymbol{X} = \boldsymbol{x_2}$, is given by:

$$\begin{aligned}\frac{\frac{P(Y>k|\boldsymbol{x_1})}{P(Y\leq k|\boldsymbol{x_1})}}{\frac{P(Y>k|\boldsymbol{x_2})}{P(Y\leq k|\boldsymbol{x_2})}} &= \frac{\exp(\alpha_k + \boldsymbol{x_1}\boldsymbol{\beta})}{\exp(\alpha_k + \boldsymbol{x_2}\boldsymbol{\beta})} \\ &= \exp(\alpha_k + \boldsymbol{x_1}\boldsymbol{\beta} - \alpha_k - \boldsymbol{x_2}\boldsymbol{\beta}) \\ &= \exp((\boldsymbol{x_1} - \boldsymbol{x_2})\boldsymbol{\beta})\end{aligned}$$

which does not depend on k. The proportional odds assumption is displayed by the parallel logistic curves in Figure 1.

Other logit-link proportional odds models include:

- The adjacent category model:

$$\log\frac{P(Y_i = k+1|\boldsymbol{x_i})}{P(Y_i = k|\boldsymbol{x_i})} = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}, \quad k = 1, ..., K-1$$

- The backward continuation ratio model:

$$\log\frac{P(Y_i = k|Y_i \leq k, \boldsymbol{x_i})}{P(Y_i < k|Y_i \leq k, \boldsymbol{x_i})} = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}, \quad k = 1, ..., K-1$$

- The forward continuation ratio model:

$$\log\frac{P(Y_i = k|Y_i \geq k, \boldsymbol{x_i})}{P(Y_i > k|Y_i \geq k, \boldsymbol{x_i})} = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}, \quad k = 1, ..., K-1$$

Furthermore, alternative link functions can be used for these models. For example, instead of the logit, the cumulative link model can utilize a:

- Probit link: $\Phi^{-1}(\frac{P(Y_i>k|\boldsymbol{x_i})}{P(Y_i\leq k|\boldsymbol{x_i})}) = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}$, where Φ is the cumulative distribution function (CDF) of the standard normal distribution

- Complimentary log-log (cloglog) link: $\log(\text{-}\log(1 - P(Y_i > k|\boldsymbol{x_i}))) = \alpha_k + \boldsymbol{x_i}\boldsymbol{\beta}$

1.2.2 Maximum likelihood estimation

The length-n ordinal response vector $\boldsymbol{y}$ can be reformatted as an n x K response matrix, $\boldsymbol{Y}$ as follows,

$$y_{ik} = \begin{cases} 1 & \text{if observation } i \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases}$$

Then the log-likelihood can be expressed as [7]

$$L = \sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log[P(Y_i = k|\boldsymbol{x_i})]$$

Pratt (1981) and Burridge (1981) showed that the log-likelihoods of the cumulative link models are concave, so a unique global maximum exists [8, 9]. However, a closed-form solution does not exist, so an iterative algorithm is needed to find the maximum likelihood (ML) estimates of the parameters, $\boldsymbol{\beta}$ and $\alpha_1, ..., \alpha_{K-1}$. One such method that is commonly used in low-dimensional problems (i.e. $p < n$) is the Newton-Raphson algorithm. Let the vector of all parameter estimates be denoted by $\boldsymbol{\theta} = \{\alpha_1, ..., \alpha_{K-1}, \beta_1, ..., \beta_p\}$. The gradient of L, which consists of the partial derivatives of L with respect to each parameter, is given by

$$g(\boldsymbol{\theta}) = (\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, ..., \frac{\partial L}{\partial \theta_{p+K-1}})$$

and the Hessian, the $(p + K - 1)$ x $(p + K - 1)$ matrix of second-order partial derivatives, is denoted by

$$\boldsymbol{H}(\boldsymbol{\theta}) = \left(\left(\frac{\partial^2 L}{\partial \theta_a \partial \theta_b} \right) \right)_{a,b \in \{1,\ldots,p+K-1\}}$$

Let the iteration number be denoted by s. The Newton-Raphson algorithm begins by specifying starting values for the parameters that are to be estimated, $\boldsymbol{\theta}^{[1]}$. The update at the next iteration and each of the subsequent iterations is

$$\boldsymbol{\theta}^{[s+1]} = \boldsymbol{\theta}^{[s]} - (\boldsymbol{H}(\boldsymbol{\theta}^{[s]}))^{-1} g(\boldsymbol{\theta}^{[s]})$$

The algorithm continues until convergence, which occurs when the difference between successive log-likelihoods, $L(\boldsymbol{\theta}^{[s+1]}) - L(\boldsymbol{\theta}^{[s]})$, is negligible.

1.3 Microarray experiments

The central dogma of molecular biology states that the genetic information encoded in the DNA within the nucleus of each cell is *transcribed* into messenger RNA (mRNA), and the mRNA is then *translated* into proteins in the cell cytoplasm. Genes are the regions of DNA that encode instructions to produce proteins, and proteins are "used to support the cell structure, to break down chemicals, to build new chemicals, to transport items, and to regulate production [10]." Put simply, genes dictate which proteins and how much of the proteins should be synthesized, and proteins control cell function. Thus, along with environmental factors, genes determine the characteristics, or phenotypes, of cells and as a result, the entire organism [11]. Furthermore, aberrant cell function can lead to a host of diseases, so interest lies in detecting which genes are involved in causing the problem.

We can measure the activity of a gene (called *gene expression*) by quantifying the amount of mRNA that is transcribed from that particular gene. A microarray does this for thousands of genes simultaneously. There are many assays developed by different companies, but we analyzed data from two of the most common platforms, Affymetrix GeneChips and Illumina BeadChips. The technology involved in the two arrays is quite different, but the goal is

the same: to quantify the expression of the tens of thousands of genes present on the chip. Each gene is represented on the array by a set of short sequences of nucleotides called oligonucleotides. Each chip is used to measure a single patient's gene expression. Thus, a typical experiment involves n microarrays and the goal is to detect differences in expression between groups with different phenotypes (e.g. cancer patients and healthy patients).
There are a multitude of statistical challenges that arise from microarray experiments. The data are inherently noisy because there are many sources of obscuring variation, including from array fabrication and image processing [11]. However, we will be focusing on the statistical challenges that arise once the data has been cleaned and normalized: namely the fact that there are far more predictors than samples in a typical experiment, i.e. $p >> n$.

1.4 High-dimensional classification

Traditional methods for fitting classification and regression models require the number of subjects, n, to be larger than the number of predictors, p, and they assume that the predictors are independent. However, in high-throughput genomic experiments, we cannot safely assume independence of the predictors (often called features in this field) and $p >> n$, which results in a host of complexities. First, the design matrix will not be full-rank, which eliminates the ability to find a stable, unique solution. Second, with a large number of predictors relative to the sample size, a subset of the predictors is likely to exhibit collinearity, which will lead to unstable parameter estimates. Furthermore, genomic data is typically sparse, meaning out of the thousands of features, only a small proportion are associated with the response, which complicates the process of discovering significant features. Model building tools such as forward and backward stepwise selection are not feasible given the huge number of predictors.

1.4.1 Least Absolute Shrinkage and Selection Operator

Recently, penalization (also referred to as regularization) has stood out as an effective method to combat the issues that come with high-dimensional data. There are several popular penalization methods, but the defining characteristic of them all is that they shrink the absolute coefficient estimates towards 0. As a result, bias is introduced into the parameter estimates in exchange for a reduction in variance. In cases where model parsimony and interpretability are important, the least absolute shrinkage and selection operator (LASSO) penalization method is effective as it shrinks parameter estimates deemed unimportant to be exactly zero [12]. LASSO models are fit by maximizing the likelihood of the data minus a penalty term, as given below:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \left(L(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{x}) - \lambda \sum_{p=1}^{P} |\beta_p| \right)$$

The tuning parameter, λ, controls the balance between bias and variance. Traditional maximum likelihood estimation occurs when $\lambda = 0$, but as previously mentioned, when $p > n$, there is no unique solution. Therefore, the parameter estimates will have minimal bias but will be highly unstable (high variance). As λ increases, the parameter estimates shrink towards 0. As this happens, more bias is added to the estimates, but the variance decreases. As λ increases, the number of nonzero coefficient estimates decreases, which leads to a more stable and interpretable model.

Hastie et. al showed that the Generalized Monotone Incremental Forward Stagewise (GMIFS) algorithm solves for the LASSO solution in a penalized logistic regression model [13]. The algorithm proceeds as follows:

1. Enlarge the predictor space as $\tilde{\mathbf{X}} = [\mathbf{X} : -\mathbf{X}]$, where $\mathbf{X}$ represents the standardized predictors.

2. For step s=0, initialize the components of $\hat{\boldsymbol{\beta}}^{(s)}$ as $\hat{\beta}_1 = \hat{\beta}_2 = ... = \hat{\beta}_P = \hat{\beta}_{P+1} = ... = \hat{\beta}_{2P} = 0$.

3. Find $m = \underset{j}{\text{argmin}}$ -$\delta \log L / \delta \beta_j$ at the current estimate $\hat{\boldsymbol{\beta}}^{(s)}$.

4. Update $\hat{\beta}_m^{(s+1)} = \hat{\beta}_m^{(s)} + \epsilon$, where ϵ is a small step size, such as 0.001.

5. Repeat steps 3 to 4 many times.

The design matrix is expanded with its negated version to avoid the computationally burdensome step of calculating the second derivative, which would have been needed to find the step direction in step 4. The final coefficient estimates are given by

$$\hat{\beta}_j = \hat{\beta}_j - \hat{\beta}_{j+P}, \quad j = 1, ..., p$$

The GMIFS method was extended by Archer et al. for fitting logit, probit, and complimentary log-log link ordinal response models (cumulative, adjacent category, stereotype, forward continuation ratio, and backward continuation ratio) to high-throughput genomic data[7]. We discuss this method in detail in Chapter 4.

1.4.2 Component-wise gradient boosting

Another method for fitting a logistic regression model in high-dimensional data settings is called componentwise gradient boosting [14]. According to Hastie et. al, boosting is "one of the most powerful learning ideas introduced in the last twenty years."[15] It is one method in a class of ensemble schemes that combines multiple weak function estimates to form one aggregated estimator. The aim of boosting is to estimate a function, $f^*(\cdot)$, that minimizes the risk:

$$R = E[L(Y, f(X))],$$

where L is the loss function chosen dependent on the structure of the response[14]. Gradient boosting searches for the solution by following the steepest path down the gradient of the empirical risk function, $\frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i))$ in function space. Gradient boosting can be applied to many different estimators including regression trees, generalized linear models (GLMs), and Cox proportional hazards models, but we limit our discussion to GLMs. Most

often the loss function is the negative log-likelihood of the response distribution and f is assumed to belong to a parameterized class of functions, $f(x, \boldsymbol{P})$. The component-wise gradient boosting algorithm for GLMs is as follow:

1. Center the predictors and set $m = 0$.

2. Initialize $\hat{\boldsymbol{\beta}}^{[0]} = 0$, where $\hat{\boldsymbol{\beta}}^{[0]}$ is the vector of initial parameter estimates of length p.

3. Initialize the function estimate as $\hat{\boldsymbol{f}}^{[0]} = \underset{c}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, c)$, where $\hat{\boldsymbol{f}}$ is a vector of length n.

4. Increase m by 1 (m represents the number of iterations).

5. Compute the negative gradient of the loss function with respect to f and, for each observation, evaluate it at the estimate of the previous iteration, $\hat{\boldsymbol{f}}^{[m-1]}(\boldsymbol{x}_i)$:

$$\boldsymbol{U}^{[m]} = \left(U_i^{[m]}\right)_{i=1,\ldots,n} = \left(-\frac{\delta}{\delta f} L(y_i, f)|_{f=\hat{\boldsymbol{f}}^{[m-1]}(\boldsymbol{x}_i)}\right)_{i=1,\ldots,n}$$

6. Fit each predictor, $\boldsymbol{x_j}$, separately to the negative gradient vector $\boldsymbol{U}^{[m]}$ using a "base learner," $h(\boldsymbol{x_j}; \beta_j), \quad j = 1, ..., p$. A base learner in this context is a simple linear model, and the coefficient in each model is estimated using least squares. Each of the p models will result in a different estimate of $\boldsymbol{U}^{[m]}$.

7. Select the model that fits the negative gradient best as determined by the residual sum of squares criterion. Let $\hat{\boldsymbol{U}}^{[m]}$ represent the vector of predicted values for that model and assign the estimated coefficient of that model to $\hat{\beta}^s$.

8. Update the current function estimate and parameter estimate:

$$\hat{\boldsymbol{f}}^{[m]} = \hat{\boldsymbol{f}}^{[m-1]} + \nu \hat{\boldsymbol{U}}^{[m]}$$
$$\hat{\boldsymbol{\beta}}^{[m]} = \hat{\boldsymbol{\beta}}^{[m-1]} + \nu \hat{\beta}^s$$

   Note that ν is a pre-specified small step length factor between 0 and 1. Also, $\hat{\boldsymbol{\beta}}^{[m]}$ is the vector of coefficient estimates, while $\hat{\beta}^s$ is a single coefficient estimate.

Repeat steps 4-8 until $m = m_{stop}$, where m_{stop} represents the stopping iteration which is generally chosen by cross-validation or an information criterion. Furthermore, the choice of ν is not highly influential; any small value (e.g. $\nu = 0.1$) is fine[14]. A cumulative logit ordinal response model can be fit using an extension of component-wise gradient called proportional odds boosting [16]. We will discuss this method in Chapter 2.

1.5 Measuring model performance

We are interested in assessing the models' ability to accurately predict the outcome of new observations as well as their ability to discriminate between important and unimportant features in the high-dimensional feature space.

1.5.1 Prediction

To assess prediction in a classification setting with a binary outcome, the misclassification rate, which is defined as

$$\#\frac{(Y \neq \hat{Y})}{n}$$

is often utilized. The misclassification rate is a useful and simple measure. However, in the ordinal response setting, the severity of the misclassifications is lost. For instance, the same penalty is applied if an observation in class 1 is classified to class 2 or to class 4. Therefore, a better method is to examine the misclassification rate separately for each class. The resulting K measures are called the class-specific misclassification rates.

Another useful metric examines the association between the observed and predicted values in terms of discordance and concordance. Let y_a and $\hat{y}_a$ be the observed and the predicted ordinal response variables, respectively, for subject a. Given a pair of subjects, if the subject that has a larger observed value also has the larger predicted value, the pair of subjects is *concordant*. If the subject that has a larger observed value has the smaller predicted value, the pair of subjects is *discordant*. Now, let C represent the number of concordant pairs in a sample and D represent the number of discordant pairs. Define T_Y

as the number of pairs in which the two subjects had the same observed response value (i.e. they were tied with respect to y). Somer's D is an asymmetric measure of association, meaning we cannot treat the two variables interchangeably. We used Somer's D_{XY}, which measures how well X serves as a predictor of Y [17]. Here, X represents the predicted value, or $\hat{y}_a$, and Y represents the observed outcome, y_a. The sample version of Somer's D_{XY} is given by:

$$D_{XY} = \frac{C - D}{n(n-1)/2 - T_Y},$$

where the numerator represents the difference in the number of concordant and discordant pairs, and the denominator represents the total number of pairs that are untied on Y. D_{XY} ranges from -1 to 1, where $D_{XY} = -1$ indicates a perfect negative association, $D_{XY} = 1$ indicates a perfect positive association, and $D_{XY} = 0$ indicates no association.

1.5.1.1 10-Fold cross-validation

One way of assessing performance is to fit a model to the entire dataset, predict the outcome of the observations in the same dataset, and then estimate the metrics described previously using the predictions. This method gives overly optimistic results and does not estimate how well the model's performance will generalize to independent data. One way of overcoming these shortcomings is to use 10-fold cross-validation, which proceeds as follows:

1. Split the data evenly into 10 equal partitions.
2. For $m = 1, 2, ..., 10$:
   (a) Fit a model using all partitions except partition m.
   (b) Predict the outcome of observations in partition m using the fitted model.
   (c) Estimate the performance measure(s) using the predicted classes of observations in partition m.
3. Let n_m denote the sample size of partition m, and let n denote the total sample size. Also, let $\hat{w}_m$ denote the performance metric estimated using the predicted classes of

observations in partition m (e.g. Somers' D_{XY}). Estimate the 10-fold cross-validation estimate of the metric, $\hat{w}$, as

$$\hat{w} = \sum_{m=1}^{10} \frac{n_m}{n} \hat{w}_m$$

1.5.2 Feature selection

Let the features that are truly associated with the ordinal response be deemed important, and we call the features that are not associated with the outcome unimportant. To assess feature selection in simulations, where we know which features are important and which are unimportant, we estimate

1. Sensitivity = $\frac{\text{Number of important features selected}}{\text{Total number of important features}}$
2. Specificity = $\frac{\text{Number of unimportant features } \textit{not} \text{ selected}}{\text{Total number of unimportant features}}$

In gene expression data analyses when we do not know which features are truly important and which are truly unimportant, we can examine the features selected by the model in the literature in an attempt to validate model findings.

1.6 Discussion

In this chapter, we have provided the necessary overview of ordinal regression as well as methods for fitting penalized classification and regression models. In Chapter 2, we introduce the method we developed, Ordinal Feature Augmentation via Nonparametrics and Selection (FANS). This method is compared to other methods suitable for fitting high-dimensional ordinal response models in Chapter 3. To that end, we examine prediction accuracy as well as feature selection in a simulation study and several gene expression data analyses. Next, in Chapter 4, a novel method for performing discrete time survival analysis in high-dimensions is described. Finally, we conclude with a discussion of the methods developed as well as future work in Chapter 5.

CHAPTER 2

FEATURE AUGMENTATION VIA NONPARAMETRICS AND SELECTION

2.1 Introduction

2.1.1 Ensemble learners

In a supervised learning problem with a discrete outcome, the goal is to use a training sample of known outcomes and predictors, $\{y_i, \boldsymbol{x}_i\}_1^n$, to build a classifier that accurately predicts the outcome of a future observation. An *ensemble learner* is composed of two or more classifiers, with the idea that the combination of classifiers should produce more accurate predictions than any of the individual models. However, the constituents must be both accurate and diverse in order for an ensemble to be more accurate than any of the individual classifiers [18]. Here, an accurate classifier is defined as one with a misclassification rate lower than that of random guessing, and two classifiers are said to be diverse if they make different errors on new data [19]. In the ordinal response setting, the misclassification rate of random guessing is equal to $\frac{K-1}{K}$, where K denotes the number of classes in the outcome.

There are several methods for constructing an ensemble learner that can be applied to a variety of classification and regression algorithms [19]. One method forms an ensemble of classifiers using manipulated versions of the training data. Breiman (1994) created one of the first ensemble schemes in this way [20]. The method, called bootstrap aggregation, or bagging for short, generates multiple *decision trees* using bootstrap replicates of the training data. To build a decision tree, all observations begin together in one set, called a root node. The observations are then partitioned, or split, into two distinct sets, called nodes. Each of these nodes is then further split into two nodes. This process continues until some stopping criteria is met. At each step, to determine the optimal split, we need a set of binary questions

which are defined such that [21]:

1. Each split depends on only one predictor.

2. For ordered (ordinal and continuous) predictors, the question takes the form, "Is $x \leq c$?", where c is in the set of observed values for that predictor.

3. For nominal predictors, the binary question takes the form, "Is $x \in A$?, where A is a subset of the observed values for that predictor.

Bagging begins by sampling n observations with replacement from the training data of size n to create a *bootstrap replicate*. Then, a classifier is built using the bootstrap replicate. This procedure of sampling with replacement from the training data and then building a classifier using the bootstrap replicate is repeated many times. When the outcome is discrete, the classifiers are aggregated by plurality voting. That is, each classifier returns a prediction for a new observation, and the class with the most "votes" is the predicted class [20]. The performance of the ensemble generally improves as the number of constituent classifiers increases. However, at a certain point, the performance stabilizes and adding more classifiers does not help much. With bagging, Breiman succinctly states that "what one loses...is a simple and interpretable structure. What one gains is increased accuracy."[20]

Manipulating the input features is another method for constructing an ensemble [19]. For instance, Cherkauer (1996) trained an ensemble of 32 models based on 8 different subsets of the 119 available features and 4 different tuning parameters [22]. This method for constructing an ensemble only works well when the predictors suffer from collinearity [19].

A third technique for building an ensemble learner is to manipulate the outcome variable. A method called error-correcting output coding randomly partitions the K classes of a discrete (nominal) outcome into two subsets, A and B [23]. The training data are re-coded so that the outcome for all observations whose response is in A is coded as 0 and the outcome for any observations whose response is in B is coded as 1. Then, a classifier is built using the binary outcome. The steps of partitioning the outcome classes, re-coding the multi-class

outcome for each observation in the training data, and then fitting a binary response model is repeated L times. For a new observation, each of the L classifiers will predict whether the outcome is in A or B. Each time a classifier predicts the outcome is in A, the classes in that subset receive a vote, and likewise for subset B. Then, the class with the most votes is the predicted class on the original multi-class scale [23].

Finally, a fourth general method for building an ensemble of classifiers is to inject randomness into the learning algorithm [19]. Random forest is one of the most common ensemble schemes and it improves on bagging by injecting randomness into the tree building procedure [24]. At each step in the tree-building procedure, instead of considering all features when searching for the best split, random forest considers a random subset of the features. This helps to reduce the correlation among the classifiers (trees) that are aggregated, which drives down the variance of the ensemble learner.

2.1.2 Feature Augmentation via Nonparametrics and Selection

Feature Augmentation via Nonparametrics and Selection (FANS) is a two-class modeling procedure that has shown promising results in high-dimensional learning problems [25] by building an ensemble of classifiers in a unique way. Suppose we have n feature-outcome pairs in a training set, $T = \{(\boldsymbol{x_i}, y_i)\}_i^n$, where $y_i \in \{0, 1\}$ and $\boldsymbol{x}_i \in \mathbb{R}^p$. We begin with some definitions. Let $g(\boldsymbol{x}) = P(\boldsymbol{X} = \boldsymbol{x}|Y = 0)$, $f(\boldsymbol{x}) = P(\boldsymbol{X} = \boldsymbol{x}|Y = 1)$, and $\pi = P(Y = 1)$. The Bayes classifier, which minimizes the probability of misclassifying an observation, assigns observations to class 1 if $P(Y = 1|\boldsymbol{X}) > 0.5$ and to class 0 otherwise. Thus, the Bayes-optimal decision boundary is defined as

$$\begin{aligned} & \{\boldsymbol{x} : P(Y = 1|\boldsymbol{x}) = P(Y = 0|\boldsymbol{x})\} \\ \Rightarrow \quad & \{\boldsymbol{x} : \frac{\pi f(\boldsymbol{x})}{\pi f(\boldsymbol{x}) + (1-\pi)g(\boldsymbol{x})} = \frac{(1-\pi)g(\boldsymbol{x})}{\pi f(\boldsymbol{x}) + (1-\pi)g(\boldsymbol{x})}\} \\ \Rightarrow \quad & \{\boldsymbol{x} : \frac{\pi f(\boldsymbol{x})}{(1-\pi)g(\boldsymbol{x})} = 1\} \end{aligned}$$

If we assume equal priors, i.e. $P(Y = 1) = P(Y = 0)$, then the Bayes decision boundary becomes

$$\{\boldsymbol{x} : \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} = 1\} = \{\boldsymbol{x} : \log\frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} = 0\}.$$

Thus, given a single feature, x, the best univariable classifier of y is [25]

$$\hat{y} = \begin{cases} 1 \text{ if} & \log\left(\frac{f(x)}{g(x)}\right) > 0 \\ 0 \text{ if} & \log\left(\frac{f(x)}{g(x)}\right) < 0 \end{cases}$$

The Naïve Bayes classifier assumes the p features within each class are independent, so the joint distribution of features in the two classes becomes $f(\boldsymbol{x}) = \prod_{j=1}^{p} f_j(x_j)$ and $g(\boldsymbol{x}) = \prod_{j=1}^{p} g_j(x_j)$. Consequently, the nonparametric Naïve Bayes decision boundary is

$$\left\{\boldsymbol{x} : \sum_{j=1}^{p} \log\frac{f_j(x_j)}{g_j(x_j)} = 0\right\}.$$

However, this independence assumption is too strong in most cases. For example, in a gene expression microarray experiment, each feature is designed to interrogate a gene (or part of a gene), and we know that genes interact in biological networks. Hence, non-zero correlations exist among some sets of genes. Thus, assuming the features from a microarray experiment are independent would be ill-advised. The FANS method attempts to account for dependence among the features by adding optimized weights to the Naïve Bayes classifier to form the following decision boundary [25]:

$$\text{FANS}_D = \left\{\boldsymbol{x} : \beta_0 + \sum_{j=1}^{p} \beta_j \log\frac{f_j(x_j)}{g_j(x_j)} = 0\right\}.$$

The method combines the best univariable predictors in a linear and additive manner in an attempt to create a powerful multivariable classifier. Note that the decision boundary based on the original features is nonlinear, which makes the classifier more flexible. However, since FANS_D is linear in the coefficients, we can rename the transformed variables (i.e. $z_j = \log\frac{f_j(x_j)}{g_j(x_j)}$) and apply any binary linear classification procedure to estimate the coefficients. When $p > n$, the model is overparameterized, so penalization/regularization techniques are

required to fit the model. These methods shrink the coefficient estimates towards zero, which adds bias in exchange for a reduction in variance, leading to improved model performance [15]. The FANS classifier uses penalized logistic regression [25] and is fit using the following algorithm that utilizes data splitting and prediction averaging to build an ensemble learner [25]:

1. Randomly partition the data into two sets, (D_1, D_1^c), analogous to splitting data into a training set and a test set.

2. Using D_1, estimate $f_j(x_j) = P(X_j = x_j|Y = 1)$ and $g_j(x_j) = P(X_j = x_j|Y = 0), \quad j = 1, ..., p$ by kernel density estimation. Denote the estimates as $\hat{\boldsymbol{f}} = (\hat{f}_1, ..., \hat{f}_p)$ and $\hat{\boldsymbol{g}} = (\hat{g}_1, ..., \hat{g}_p)$.

3. Use the data in D_1^c to calculate the matrix of transformed variables, $\boldsymbol{Z}$, where

$$z_{ij} = \log\left[\frac{\hat{f}_j(x_{ij})}{\hat{g}_j(x_{ij})}\right], \quad x_{ij} \in D_1^c$$

4. Using $\boldsymbol{Z}$ and $\boldsymbol{y} \in D_1^c$, fit an L_1 penalized logistic regression model.

5. Repeat steps 1-4 L times to obtain $(D_1, D_1^c), ..., (D_L, D_L^c)$. However, the partitions in step 1 are formed differently for odd and even numbered iterations. For odd numbered iterations, the training set is randomly partitioned. For even numbered iterations, the roles of the partitions in the preceding odd numbered iteration are reversed. That is, the first partition estimates the marginal densities for odd numbered iterations, but for even numbered iterations, the second partition estimates the marginal densities. In general, $(D_{2l}, D_{2l}^c) = (D_{2l-1}^c, D_{2l-1})$ for $l = 1, ..., \lfloor \frac{L}{2} \rfloor$, where (D_{2l-1}, D_{2l-1}^c) is formed by randomly partitioning T and (D_{2l}, D_{2l}^c) is formed by reversing the roles of D_{2l-1} and D_{2l-1}^c.

In order to predict the outcome of a new observation, $\boldsymbol{x}_i = (x_{i1}, ..., x_{ip})$, we must use the densities and models estimated from the learning set. For each of the L partitions, we

obtain a different estimate of $\boldsymbol{f}$ and $\boldsymbol{g}$, and a different fitted model. Thus, for each of the $l = 1, 2, ..., L$ partitions, we can use $\hat{\boldsymbol{f}}_l$ and $\hat{\boldsymbol{g}}_l$ to transform $\boldsymbol{x}_i$ and obtain $\boldsymbol{z}_{il} = (z_{i1l}, ..., z_{ipl})$, where

$$z_{ijl} = \log\left[\frac{\hat{f}_{jl}(x_{ij})}{\hat{g}_{jl}(x_{ij})}\right], \quad j = 1, ..., p.$$

So, we end up with L vectors of transformed features, one from each iteration of the FANS algorithm. Each one can be input into the corresponding fitted model from step 4 in order to obtain a predicted probability, $p_{il} = \hat{P}(Y_i = 1|\boldsymbol{z}_{il})$. The average of these L predicted probabilities, $\bar{p}_i = \frac{1}{L}\sum_{l=1}^{L} p_{il}$, is used to classify the observation according to the Bayes classifier: if $\bar{p}_i > 0.5$, predict class 1; otherwise, predict class 0 [25].

In step one, the roles of the data used for the density estimation and model fitting are reversed from one partition to the next in order to ensure a balanced assignment of data for the two tasks [25]. Partitioning the data multiple (L) times and averaging the predicted probabilities makes efficient use of the data and ensures the procedure is robust to arbitrary assignment of data to the two tasks [25].

Several simulation examples were reported in the FANS manuscript [25]. The first three simulations generated the features in each class from a multivariate Gaussian distribution. The first let the two classes be linearly separable with an autoregressive (1) correlation structure among the features. The second simulation was the same as the first except the correlation structure was changed to be compound symmetric. The third generated the two classes to be separable by a nonlinear decision boundary with a compound symmetric correlation structure. Finally, the fourth simulation used a nonlinear decision boundary but generated the features using a uniform distribution. The FANS method performed as well or better than the competing methods including penalized logistic regression (PLR), penalized additive logistic regression (penGAM), Naïve Bayes (NB), and support vector machines (SVM) (Table 1) [25]. The most competitive method to FANS in terms of misclassification error was penGAM. However, the median computation time for the simulations ranged from 3.4 to 6.6 seconds for FANS and 243.7 to 4811.9 seconds for penGAM, so FANS seemed to

Table 1. Median classification error (%) on an independent test set with standard errors for the simulation study. Different values of ρ were used for each example.

Simulation (ρ)	FANS	PLR	penGAM	NB	SVM
AR (0)	6.8(1.1)	6.5(1.2)	6.6(1.1)	11.2(1.4)	13.2(1.5)
AR (0.5)	16.5(1.7)	15.9(1.7)	16.9(1.6)	20.6(1.7)	22.5(1.8)
CS (0.5)	4.2(0.9)	2.5(0.6)	3.7(0.9)	43.5(11.1)	5.3(1.1)
CS (0.9)	3.1(1.1)	0.0(0.0)	0.2(1.4)	46.8(8.8)	0.0(0.1)
Nonlinear CS (0)	0.0(0.0)	50.0(1.3)	0.0(0.1)	50.4(2.2)	31.8(2.4)
Nonlinear CS (0.5)	3.4(0.7)	50.0(1.3)	3.7(0.8)	50.0(2.1)	19.8(2.4)
Nonlinear Unif	0.0(0.0)	50.0(10.7)	0.0(0.0)	41.0(1.1)	0.0(0.0)

be much more computationally efficient [25].

The authors also compared the predictive performance of FANS and the other methods using a lung cancer gene expression dataset. There were $p = 12,533$ features and $n = 181$ observations, where the number of observations in the training set, n_{train}, was 32, and the number of observations in the test set, n_{test}, was 149. The aim was to predict whether observations in the test set were adenocarcinoma (ADCA) or mesothelioma [25]. With $L = 20$ partitions, FANS obtained a 0% misclassification rate, while the other methods failed to do so [25].

In order to independently verify the predictive performance of FANS, we coded the method in the R programming environment [26] and compared the performance of models fit using FANS, penalized logistic regression, and Naïve Bayes. We applied the three methods to a publicly available lung cancer gene expression dataset, GSE4115, where there were $n = 163$ observations after removing samples without a definitive cancer diagnosis and $p = 22,215$ features [27]. We split the data into a training set and an independent test set. Given the fact that the training set is partitioned into two sets in the first step of the FANS algorithm, we wanted to ensure there was enough data in the first set to estimate the marginal densities. Therefore, we assigned $\frac{2}{3}$ of the data to the training set ($n_{train} = 136$) and the remaining $\frac{1}{3}$ to the test set ($n_{test} = 27$). For the FANS model, we set $L = 2$, 10, and 20. The aim was to predict whether subjects had lung cancer or not [27]. The model fit using FANS with $L = 20$ achieved a 0% misclassification rate on the independent test set,

while the misclassication rates for the penalized logistic regression and Naïve Bayes models were 25.9% and 18.5%, respectively. Therefore, we also identified FANS as a competitive algorithm in the binary response case. With $L = 2$ and $L = 10$, the FANS misclassication rates were 14.8% and 3.7% respectively, suggesting that the data splitting and prediction averaging scheme in the FANS algorithm is effective. For $L = 2$, 10, and 20, the runtime for FANS was 31.29 minutes, 114.63 minutes, and 312.09 minutes, respectively. Meanwhile, the runtimes for Naïve Bayes and penalized logistic regression were 0.96 minutes and 0.293 minutes, respectively. However, we did not apply parallel programming techniques when we coded the FANS algorithm, which would have significantly reduced the runtime.

2.1.3 Extending FANS to the ordinal response setting

In the binary response setting, FANS models the class-conditional marginal density ratios,

$$z_j = \log \frac{P(X_j = x_j|Y = 1)}{P(X_j = x_j|Y = 0)} = \log \frac{f_j(x_j)}{g_j(x_j)}.$$

Because there are $K > 2$ ordinal outcome classes, the augmented features must be redefined in the FANS algorithm to accommodate all K class-conditional density estimates. One approach to create the ordinal augmented features is to consider the decision boundaries between adjacent classes, which would result in $K - 1$ augmented features for each original feature:

$$\begin{aligned}
z_j^{(1)} &= \log \left[\frac{\hat{P}(X_j = x_j|Y = 1)}{\hat{P}(X_j = x_j|Y = 2)} \right] \\
z_j^{(2)} &= \log \left[\frac{\hat{P}(X_j = x_j|Y = 2)}{\hat{P}(X_j = x_j|Y = 3)} \right] \\
&\vdots \\
z_j^{(K-1)} &= \log \left[\frac{\hat{P}(X_j = x_j|Y = K-1)}{\hat{P}(X_j = x_j|Y = K)} \right].
\end{aligned}$$

This approach reduces to the binary FANS method when $K = 2$. However, one issue is that the augmented features do not use the whole training set. Furthermore, the density estimates may be poor if there is not a sufficient number of observations in each class. Thus, the augmented features will be defined in a way that avoids these issues:

$$\begin{aligned} z_j^{(1)} &= \log\left[\frac{\hat{P}(X_j = x_j|Y = 1)}{\hat{P}(X_j = x_j|Y > 1)}\right] = \log\left[\frac{\hat{f}_j^{(1)}(x_j)}{\hat{g}_j^{(1)}(x_j)}\right] \\ z_j^{(2)} &= \log\left[\frac{\hat{P}(X_j = x_j|Y \leq 2)}{\hat{P}(X_j = x_j|Y > 2)}\right] = \log\left[\frac{\hat{f}_j^{(2)}(x_j)}{\hat{g}_j^{(2)}(x_j)}\right] \\ &\vdots \\ z_j^{(K-1)} &= \log\left[\frac{\hat{P}(X_j = x_j|Y \leq K-1)}{\hat{P}(X_j = x_j|Y = K)}\right] = \log\left[\frac{\hat{f}_j^{(K-1)}(x_j)}{\hat{g}_j^{(K-1)}(x_j)}\right]. \end{aligned} \tag{2.1}$$

This way, each augmented feature uses the entire dataset, and the only densities that are class-specific (i.e. conditional on only one class) are $\hat{f}_j^{(1)}$ and $\hat{g}_j^{(K-1)}$. Furthermore, this approach also reduces to the binary FANS method when $K = 2$.

Now, because there are $K - 1$ augmented features for each original feature, we must discover the best way to model the augmented features. One method is to include all $K - 1$ in the model, resulting in $(K - 1) * p$ input variables for the Ordinal FANS model. A typical Affymetrix HG-U133A microarray contains around 22,215 features, and if $K = 4$, we would need to model $22,215 * 3 = 66,645$ augmented features. This approach would greatly increase the computational time of fitting the model and may reduce the predictive accuracy of the model by increasing the sparsity of the solution. Another option would be to perform principal components analysis (PCA) on each of the p sets of $K - 1$ augmented features. We could then fit a model with the first principal component (PC) from each analysis. However, if the first PC in each of the p PCAs does not explain the vast majority of the total variance, we would lose significant information by excluding the other PC's. Therefore, we implemented two approaches:

1. Fit a penalized binary response model for each of the $K-1$ augmented features and aggregate the results using the technique proposed in the Ordinal Real AdaBoost algorithm [28]. We refer to this approach as the binary aggregation approach.

2. Fit a cumulative logit model using proportional odds boosting [16], where the $K-1$ augmented features for each predictor will either all be included or all be excluded from the fitted model. We refer to this approach as the boosting approach.

2.2 Approach 1: Aggregating penalized binary response models

Define the matrix, $\boldsymbol{Z^{(k)}}$, which contains the elements $z_{ij}^{(k)}$ for subjects $i \in \{1, 2, ..., n\}$ and features $j \in \{1, 2, ..., p\}$. In other words, $\boldsymbol{Z^{(k)}}$ is the design matrix for augmented feature k, and $\boldsymbol{z_i^{(k)}}$ is a 1 x p row in $\boldsymbol{Z^{(k)}}$. We will fit the following $K-1$ binary response models using $\boldsymbol{Z^{(1)}}, ..., \boldsymbol{Z^{(K-1)}}$:

$$\begin{aligned}
\log\left[\frac{P(Y=1|\boldsymbol{z}^{(1)})}{P(Y>1|\boldsymbol{z}^{(1)})}\right] &= \beta_0^{(1)} + \boldsymbol{z^{(1)}\beta^{(1)}} \\
\log\left[\frac{P(Y\leq 2|\boldsymbol{z}^{(2)})}{P(Y>2|\boldsymbol{z}^{(2)})}\right] &= \beta_0^{(2)} + \boldsymbol{z^{(2)}\beta^{(2)}} \\
&\vdots \\
\log\left[\frac{P(Y\leq K-1|\boldsymbol{z}^{(K-1)})}{P(Y=K|\boldsymbol{z}^{(K-1)})}\right] &= \beta_0^{(K-1)} + \boldsymbol{z^{(K-1)}\beta^{(K-1)}}
\end{aligned} \tag{2.2}$$

Each model is fit using L1-penalized logistic regression, which estimates the penalized solution given by

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \left(L(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{z}) - \lambda \sum_{p=1}^{P} |\beta_p| \right)$$

The resulting solution is sparse, meaning that the model fitting algorithm shrinks the coefficient estimates of features deemed unimportant to zero resulting in a parsimonious set of features with non-zero coefficient estimates. Thus, model fitting and feature selection are performed simultaneously. The tuning parameter, λ, controls the amount of shrinkage and is either chosen by cross-validation or by minimizing an information criterion (e.g. AIC, BIC).

As λ increases, the number of parameter estimates that will be shrunk to zero also increases. The log-likelihood, L, of the k^{th} logistic regression model is given by

$$L(\beta) = \sum_{i=1}^{n} \{I(Y_i \leq k) \log(P(Y_i \leq k)|\boldsymbol{z^{(k)}}) + I(Y_i > k) \log(P(Y_i > k)|\boldsymbol{z^{(k)}})\}$$

From equation (2.2),

$$P(Y_i \leq k)|\boldsymbol{z_i^{(k)}}) = \frac{\exp\{\boldsymbol{z_i^{(k)}\beta^{(k)}}\}}{1 + \exp\{\boldsymbol{z_i^{(k)}\beta^{(k)}}\}}$$
$$P(Y_i > k)|\boldsymbol{z_i^{(k)}}) = \frac{1}{1 + \exp\{\boldsymbol{z_i^{(k)}\beta^{(k)}}\}}$$

The binary response model fit using $\boldsymbol{Z^{(k)}}$ will predict whether $y \leq k$ or $y > k$. For a given subject i, define [28]

$$\hat{p}_{1i}^{(k)} = \hat{p}_{2i}^{(k)} = ... = \hat{p}_{ki}^{(k)} = \hat{P}(Y_i \leq k|\boldsymbol{z}_i^{(k)})$$
$$\hat{p}_{(k+1)i}^{(k)} = \hat{p}_{(k+2)i}^{(k)} = ... = \hat{p}_{Ki}^{(k)} = 1 - \hat{P}(Y_i \leq k|\boldsymbol{z}_i^{(k)}).$$

For a given subject i, each of the $K-1$ binary classifiers returns a vector, resulting in $K-1$ length-K vectors:

$$\begin{gathered} (\hat{p}_{1i}^{(1)}, \hat{p}_{2i}^{(1)}, ..., \hat{p}_{Ki}^{(1)}) \\ (\hat{p}_{1i}^{(2)}, \hat{p}_{2i}^{(2)}, ..., \hat{p}_{Ki}^{(2)}) \\ \vdots \\ (\hat{p}_{1i}^{(K-1)}, \hat{p}_{2i}^{(K-1)}, ..., \hat{p}_{Ki}^{(K-1)}) \end{gathered} \tag{2.3}$$

We can then sum the vectors from the binary classifiers fit using $\boldsymbol{Z^{(1)}}, ..., \boldsymbol{Z^{(K-1)}}$ to form an aggregated vector of scores. The class with the largest score is the predicted class for that subject [28]:

$$\hat{y}_i = \underset{m}{\text{argmax}} \sum_{k=1}^{K-1} \hat{p}_{mi}^{(k)} \tag{2.4}$$

We can incorporate this method into the FANS algorithm as follows:

1. Randomly partition the n feature-outcome pairs into two sets, (D_1, D_1^c).

2. Using D_1, estimate $\hat{f}_j^{(1)}(x_j), ..., \hat{f}_j^{(K-1)}(x_j)$ and $\hat{g}_j^{(1)}(x_j), ..., \hat{g}_j^{(K-1)}(x_j)$ for $j = 1, ..., p$ using kernel density estimation.

3. Evaluate the density estimates at the observed values in D_1^c, and calculate the matrices of augmented features, $\boldsymbol{Z^{(1)}}, ..., \boldsymbol{Z^{(K-1)}}$, where $z_{ij}^{(k)} = \log\left[\frac{\hat{f}_j^{(k)}(x_{ij})}{\hat{g}_j^{(k)}(x_{ij})}\right] \in \boldsymbol{Z^{(k)}}$ for $x_{ij} \in D_1^c, i = 1, ..., n$ and $j = 1, ..., p$ and $k = 1, ..., K-1$.

4. Use $\boldsymbol{y} \in D_1^c$ to fit the $K-1$ binary classifiers defined in equation (2.2) using L1-penalized logistic regression. The tuning parameter, λ is chosen by minimizing the AIC.

5. Reverse the roles of D_1 and D_1^c and repeat steps 2 - 4. Let $D_2 = D_1^c$ and $D_2^c = D_1$.

6. Repeat steps 1 - 6 $\frac{L}{2}$ times, which results in $L * (K-1)$ penalized logistic regression models based on the L different partitions, $(D_1, D_1^c), ..., (D_L, D_L^c)$.

2.2.1 Predicting new observations

In order to predict the outcome of a new observation, $\boldsymbol{x} = (x_1, x_2, ..., x_p)$, we must use the densities and models estimated from the learning set. For each of the L partitions, we obtain different kernel density estimates and as a result, different fitted models. Thus, for each of the L partitions, we can use the kernel density estimates to transform $\boldsymbol{x}$ and obtain the $K-1$ length-p vectors of augmented features. Then, the augmented features are plugged into the respective fitted binary classifiers. Finally, the $L * (K-1)$ binary response models are aggregated to arrive at the class prediction on the ordinal scales.

Explicitly, for a new observation, $\boldsymbol{x} = (x_1, x_2, ..., x_p)$,

1. Using the kernel densities estimated with D_1, calculate $\hat{f}_j^{(1)}(x_j), ..., \hat{f}_j^{(K-1)}(x_j)$ and $\hat{g}_j^{(1)}(x_j), ..., \hat{g}_j^{(K-1)}(x_j)$ for $j = 1, ..., p$.

2. Form the $K-1$ vectors augmented features,

$$\boldsymbol{z^{(1)}} = \log\left[\frac{\hat{f}_j^{(1)}(x_j)}{\hat{g}_j^{(1)}(x_j)}\right], \quad j = 1, 2, ..., p$$
$$\boldsymbol{z^{(2)}} = \log\left[\frac{\hat{f}_j^{(2)}(x_j)}{\hat{g}_j^{(2)}(x_j)}\right], \quad j = 1, 2, ..., p$$
$$\vdots$$
$$\boldsymbol{z^{(K-1)}} = \log\left[\frac{\hat{f}_j^{(K-1)}(x_j)}{\hat{g}_j^{(K-1)}(x_j)}\right], \quad j = 1, 2, ..., p$$

3. Plug each of the length-p vectors of augmented features into the respective fitted penalized logistic regression models in equation (2.2) and obtain the estimates of $P(Y \leq k|\boldsymbol{z^{(k)}})$, $k = 1, 2, ..., K-1$.

4. Form the vectors in equation (2.3).

5. Repeat 1-4 for $(D_2, D_2^c), ..., (D_L, D_L^c)$.

6. Sum the $L * (K-1)$ vectors of scores. The element in the vector of sums with the largest value is the predicted value on the original ordinal scale,

$$\hat{y} = \underset{m}{\text{argmax}} \sum_{l=1}^{L} \sum_{k=1}^{K-1} \hat{p}_m^{(k)}$$

2.2.2 Feature selection

In addition to developing an accurate predictive model, we are interested in discriminating between important and unimportant features from the high-dimensional set of predictors. This can be accomplished by running the FANS algorithm once, i.e. by setting $L = 1$. In this approach, the $K-1$ L1-penalized logistic regression models in equation (2.2) are fit, and each model produces a parsimonious set of features with non-zero coefficient estimates. The union of these sets is the final set of features selected.

2.3 Approach 2: Proportional Odds Boosting

2.3.1 Functional gradient descent

In the following setting, let y represent an outcome variable and let $\boldsymbol{x} = \{x_1, ..., x_p\}$ represent a vector of p predictors. Recall that in a predictive learning problem, the goal is to use a training sample of known outcomes and predictors, $\{y_i, \boldsymbol{x}_i\}_1^n$, to estimate a function that maps $\boldsymbol{x}$ to y. The true, unknown function is denoted $F^*(\boldsymbol{x})$ and the estimate is $\hat{F}(\boldsymbol{x})$. Ideally, the function would be estimated by minimizing the expected value of a loss function (chosen depending on the distribution of y), $E_y[L(y, F(\boldsymbol{x}))|\boldsymbol{x}]$ [29]. Minimizing the expected loss (also known as the *risk*) in function space requires treating the function $F(\boldsymbol{x})$ evaluated at each $\boldsymbol{x}$ as a parameter. However, there is an infinite number of such parameters in $\mathbb{R}^p$. If we instead attempt to simplify the estimation by using data, we can evaluate F at each $\boldsymbol{x}_i$ in the training set. Unfortunately, this approach breaks down because the expected loss cannot be estimated accurately using a finite dataset, and even if it could, estimates of $F^*(\boldsymbol{x})$ outside of the training set could not be obtained [29]. To remedy this situation, the function can be assumed to take a parametric form, $F(\boldsymbol{x}; \boldsymbol{P})$, where $\boldsymbol{P} = \{P_1, P_2, ...\}$ is a finite set of parameters [29]. In this case, the function estimation problem can be simplified to one of parameter estimation. Boosting assumes an additive form for the function, given by

$$F(\boldsymbol{x}; \{\nu_m, \boldsymbol{\beta}_m\}_1^M) = \sum_{m=1}^{M} \nu_m h(\boldsymbol{x}; \boldsymbol{\beta}_m).$$

In this form, the *base learner*, $h(\boldsymbol{x}; \boldsymbol{\beta})$, is a simple function of the predictors (or a subset of the predictors), parameterized by $\boldsymbol{\beta}$, and ν is a weight, or step size, for h. Futhermore, M is the number of base learners that are combined to form the estimated function. The goal then becomes to estimate the parameters that minimize the empirical risk,

$$\underset{\{\nu_m, \boldsymbol{\beta}_m\}_1^M}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \nu_m h(\boldsymbol{x}_i; \boldsymbol{\beta}_m)\right).$$

In many problems, including when $p >> n$, this is infeasible, so a "greedy stagewise" approach is taken [29]. For $m = 1, 2, ..., M$,

1. $(\nu_m, \boldsymbol{\beta}_m) = \underset{\nu, \boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\boldsymbol{x}_i) + \nu h(\boldsymbol{x}_i; \boldsymbol{\beta})\right)$
2. $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu_m h(\boldsymbol{x}; \boldsymbol{\beta_m})$

Here, $\nu_m h(\boldsymbol{x}; \boldsymbol{\beta_m})$ is called a *step*, where ν_m is the step size, and the vector $h(\boldsymbol{x}; \boldsymbol{\beta_m})$ is the step direction. The goal is to take the largest possible step in the direction of the minimum of the empirical risk. By construction, the best steepest-descent step direction is the negative gradient evaluated at the prior step's function estimates,

$$-g_m(\boldsymbol{x}_i) = -\left[\frac{\partial}{\partial F(\boldsymbol{x}_i)} L(y_i, F(\boldsymbol{x}_i))\right]_{F(\boldsymbol{x}_i)=F_{m-1}(\boldsymbol{x}_i)}$$

for $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$, which is given by $\boldsymbol{g}_m(\boldsymbol{x}) = \{-g_m(\boldsymbol{x}_i)\}_{i=1}^{n}$. However, this gradient is only defined at $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ [29]. To generalize to other points outside the training set, we can choose the base learner, $h(\boldsymbol{x}_i; \boldsymbol{\beta}_m)$ that is most correlated with $\boldsymbol{g}_m(\boldsymbol{x})$. This corresponds to replacing the difficult two-step function estimation problem given above with a simple least squares estimation problem to choose the base learner, h, among a set of candidate base learners. Specifically, the negative gradient vector is fit using each base learner, and h is set to the base learner with the smallest residual sum of squares (RSS). Then, Freidman states that we can use a line search to estimate ν_m [29], but Bühlmann and Hothorn have suggested that the additional line search is unnecessary [14]. Using empirical evidence, they determined that one can simply use a single small constant step size ν, and further, the choice of ν is unimportant, as long as it is sufficiently small (such as 0.1) [14]. Thus, the generic boosting algorithm, without the line search for step size, is given by [29, 14]:

1. Specify a set of candidate base learners. These can be, for example, regression trees or linear models with a subset of the p predictors. Herein we focus our attention on linear models. The simplest setting specifies p base learners, where each base learner is a simple linear regression model with one of the p predictors as the sole input. For

example, the base learner for the j^{th} predictor would be defined as

$$g_m(\boldsymbol{x}) = x_j\beta + \epsilon$$

2. Let $m = 0$ and initialize the vector of the estimated function evaluated at the n data points as $F_0(\boldsymbol{x}) = \underset{c}{\text{argmin}} \sum_{i=1}^{n} L(y_i, c)$. That is, each element in the length-n vector is initialized to the constant that minimizes the empirical risk.

3. Increase m by one and compute the negative gradient vector, $\boldsymbol{g}_m(\boldsymbol{x})$.

4. Fit the negative gradient vector using each base learner. An estimate of the negative gradient vector is obtained from each fitted model. Denote the estimate from the base learner with the smallest RSS as $\hat{\boldsymbol{g}}_m(\boldsymbol{x})$.

5. Update the function estimate, $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu\hat{\boldsymbol{g}}_m(\boldsymbol{x};\boldsymbol{\beta})$.

6. Iterate steps 3-5 M times, where M is a tuning parameter that is generally chosen by cross-validation [30].

Thus, the final function estimate takes the form $F(\boldsymbol{x}) = F_0(\boldsymbol{x}) + \sum_{m=1}^{M} \nu\hat{\boldsymbol{g}}_m(\boldsymbol{x})$.

2.3.1.1 Proportional Odds Boosting

A cumulative logit proportional odds model for ordinal response data is given by

$$P(Y \leq k|\boldsymbol{x}) = \frac{1}{1 + \exp(F(\boldsymbol{x}) - \theta_k)}, \quad k = 1, ..., K-1,$$

where $F(\boldsymbol{x})$ is the prediction function defined in the previous section and $\{\theta_1, ..., \theta_{K-1}\}$ are class-specific intercepts, or *thresholds*, that are constrained to be increasing from θ_1 to θ_K and are estimated simultaneously with F. The model can be expressed equivalently as

$$\log\left(\frac{P(Y > k|\boldsymbol{x})}{P(Y \leq k|\boldsymbol{x})}\right) = F(\boldsymbol{x}) - \theta_k, \quad k = 1, ..., K-1.$$

Using the given model, the class-specific probabilities can be estimated as:

$$P(Y=k|\boldsymbol{x})=\pi_k=\begin{cases}\frac{1}{1+\exp(F(\boldsymbol{x})-\theta_1)} & \text{for } k=1\\ \frac{1}{1+\exp(F(\boldsymbol{x})-\theta_k)}-\frac{1}{1+\exp(F(\boldsymbol{x})-\theta_{k-1})} & \text{for } 1<k<K\\ 1-\frac{1}{1+\exp(F(\boldsymbol{x})-\theta_{K-1})} & \text{for } k=K,\end{cases} \tag{2.5}$$

which are used to specify the log-likelihood [16]:

$$\begin{aligned}l(F,\boldsymbol{\theta}) = &-\mathrm{I}(Y=1)*\log[1+\exp(F(\boldsymbol{x})-\theta_1)]\\ &+\sum_{k=2}^{K-1}\mathrm{I}(Y=k)*\log\left[(1+\exp(F(\boldsymbol{x})-\theta_k))^{-1}-(1+\exp(F(\boldsymbol{x})-\theta_{k-1}))^{-1}\right]\\ &+\mathrm{I}(Y=K)*\log\left[1-(1+\exp(f-\theta_{K-1}))^{-1}\right]\end{aligned}$$

Schmid et al. extended the generic boosting algorithm to the ordinal response setting, using the negative log-likelihood as the loss function, L, which resulted in the proportional odds (P/O) boosting algorithm [16]:

1. Let $\hat{F}_m$ and $\hat{\boldsymbol{\theta}}_m$ denote the vectors of estimates, $(\hat{F}(\boldsymbol{x}_1),...,\hat{F}(\boldsymbol{x}_n))$ and $(\hat{\theta}_1,...,\hat{\theta}_{K-1})$ at step m. Set $m=0$ and initialize $\hat{F}_0$ and $\hat{\boldsymbol{\theta}}_0$.

2. Specify the base learners.

3. Increase m by 1 and compute the negative gradient of the loss with respect to F and, for each observation, evaluate it at the estimates of the previous iteration, $\hat{F}_{m-1}(\boldsymbol{x}_i), \hat{\boldsymbol{\theta}}_{m-1}$:

$$\boldsymbol{g}_m(\boldsymbol{x})=(-g_m(\boldsymbol{x}_i))_{i=1,...,n}=\left(-\frac{\delta}{\delta F}L(y_i,F,\boldsymbol{\theta})|_{F=\hat{F}_{m-1}(\boldsymbol{x}_i),\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{m-1}}\right)_{i=1,...,n}$$

4. Fit each of the base learners to the negative gradient vector $\boldsymbol{g}_m(\boldsymbol{x})$. Each model results in a different estimate of $\boldsymbol{g}_m(\boldsymbol{x})$.

5. Select the model that fit the negative gradient best as determined by the R^2 goodness-of-fit criterion. Let $\hat{\boldsymbol{g}}_m(\boldsymbol{x})$ represent the vector of fitted values for that model.

6. Update the current function estimate :

$$\hat{F}_m = \hat{F}_{m-1} + \nu \hat{\boldsymbol{g}}_m(\boldsymbol{x})$$

Note that ν is a pre-specified small step length factor between 0 and 1 and is not highly influential as long as it is sufficiently small (e.g. $\nu = 0.1$) [14].

7. Treating the function estimate as fixed, estimate $\boldsymbol{\theta}_m$ by minimizing the empirical risk, $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{F}_m, \boldsymbol{\theta})$.

8. Repeat steps 3-7 until $m = M$, where M represents the stopping iteration chosen by cross-validation.

The final function estimate is given by

$$\begin{aligned}\hat{F}_M =& \sum_{m=0}^{M} \hat{F}_m \\ =& \hat{F}_0 + \sum_{m=1}^{M} \nu \hat{\boldsymbol{g}}_m(\boldsymbol{x})\end{aligned}$$

2.3.2 Fitting the Ordinal FANS model with Proportional Odds Boosting

We can model the augmented features in the Ordinal FANS algorithm by specifying the base learners in step 2 of the P/O boosting algorithm. Specifically, for each of the p original features, there will be a base learner defined as a linear model of the $K-1$ augmented features. Before fitting the model, we center the augmented features because we do not include an intercept term [30]. Recall the definition of the augmented features in equation (2.1). The base learners will be:

$$\boldsymbol{z}_j \boldsymbol{\beta}_j + \epsilon, \quad j = 1, 2, ..., p \tag{2.6}$$

where $\boldsymbol{z}_j = (z_j^{(1)}, z_j^{(2)}, ..., z_j^{(K-1)})$ and $\boldsymbol{\beta}_j = (\beta_{1j}, \beta_{2j}, ..., \beta_{(K-1)j})$.

Thus, in step 4 of the P/O boosting algorithm, we fit each base learner to the negative

gradient vector calculated in step 3:

$$\begin{aligned} g_m(\boldsymbol{z}) =& \boldsymbol{z}_1 \boldsymbol{\beta}_1^{[m]} + \epsilon \\ g_m(\boldsymbol{z}) =& \boldsymbol{z}_2 \boldsymbol{\beta}_2^{[m]} + \epsilon \\ & \vdots \\ g_m(\boldsymbol{z}) =& \boldsymbol{z}_p \boldsymbol{\beta}_p^{[m]} + \epsilon \end{aligned}$$

and we set $\hat{\boldsymbol{g}}_m(\boldsymbol{z})$ to the fitted values from the base learner with the largest R^2. As a result, the coefficient estimates for that base learner are implicitly updated in step 6 of the P/O boosting algorithm by a factor of ν. For instance, assume that in step m, the base learner employing the augmented features for the second predictor fit the negative gradient best. That is,

$$\hat{\boldsymbol{g}}_m(\boldsymbol{z}) = \boldsymbol{z}_2 \hat{\boldsymbol{\beta}}_2^{[m]}.$$

Then the coefficient estimates in that model are updated as

$$\hat{\boldsymbol{\beta}}_2 = \hat{\boldsymbol{\beta}}_2^{[m-1]} + \nu \hat{\boldsymbol{\beta}}_2^{[m]}$$

We can incorporate P/O boosting into the FANS algorithm as follows:

1. Randomly partition the n feature-outcome pairs into two sets, (D_1, D_1^c).

2. Using D_1, estimate $\hat{f}_j^{(1)}(x_j), ..., \hat{f}_j^{(K-1)}(x_j)$ and $\hat{g}_j^{(1)}(x_j), ..., \hat{g}_j^{(K-1)}(x_j)$ for $j = 1, ..., p$ using kernel density estimation.

3. Evaluate the density estimates at the observed values in D_1^c, and calculate the augmented features where $z_{ij}^{(k)} = \log\left[\frac{\hat{f}_j^{(k)}(x_{ij})}{\hat{g}_j^{(k)}(x_{ij})}\right]$ for $x_{ij} \in D_1^c$, $i = 1, ..., n$ and $j = 1, ..., p$ and $k = 1, ..., K-1$.

4. Use $\boldsymbol{y} \in D_1^c$ to fit a proportional odds model with P/O boosting, using the base learners specified in equation (2.6).

5. Reverse the roles of D_1 and D_1^c and repeat steps 2 - 4. Let $D_2 = D_1^c$ and $D_2^c = D_1$.

6. Repeat steps 1 - 5 $\frac{L}{2}$ times to form $(D_1, D_1^c), ..., (D_L, D_L^c)$.

7. Repeat steps 1 - 5 $\frac{L}{2}$ times, which results in L fitted proportional odds based on the L different partitions, $(D_1, D_1^c), ..., (D_L, D_L^c)$.

2.3.2.1 Predicting new observations

To predict a new observation, $\boldsymbol{x} = (x_1, x_2, ..., x_p)$,

1. Using the kernel densities estimated with D_1, calculate $\hat{f}_j^{(1)}(x_j), ..., \hat{f}_j^{(K-1)}(x_j)$ and $\hat{g}_j^{(1)}(x_j), ..., \hat{g}_j^{(K-1)}(x_j)$ for $j = 1, ..., p$.

2. For each of the p original features, form the $K-1$ augmented features,

$$\begin{aligned}
\boldsymbol{z_1} &= \log\left[\frac{\hat{f}_1^{(k)}(x_1)}{\hat{g}_1^{(k)}(x_1)}\right], \quad k = 1, 2, ..., K-1 \\
\boldsymbol{z_2} &= \log\left[\frac{\hat{f}_2^{(k)}(x_2)}{\hat{g}_2^{(k)}(x_2)}\right], \quad k = 1, 2, ..., K-1 \\
&\vdots \\
\boldsymbol{z_p} &= \log\left[\frac{\hat{f}_p^{(k)}(x_p)}{\hat{g}_p^{(k)}(x_p)}\right], \quad k = 1, 2, ..., K-1
\end{aligned}$$

3. Calculate $\sum_{j=1}^{p} \boldsymbol{z}_j \hat{\boldsymbol{\beta}}_j^{[M]} - \theta_k, \quad k = 1, 2, ..., K-1$ and estimate the class-specific probabilities using equation (2.5).

4. Repeat 1-4 for $(D_2, D_2^c), ..., (D_L, D_L^c)$.

5. Average the class-specific probability estimates calculated from the L partitions of the

training set,

$$\bar{\pi}_1 = \frac{1}{L}\sum_{l=1}^{L} \hat{P}(Y = 1|\boldsymbol{z}_1, ..., \boldsymbol{z}_p)_l$$
$$\bar{\pi}_2 = \frac{1}{L}\sum_{l=1}^{L} \hat{P}(Y = 2|\boldsymbol{z}_1, ..., \boldsymbol{z}_p)_l$$
$$\vdots$$
$$\bar{\pi}_K = \frac{1}{L}\sum_{l=1}^{L} \hat{P}(Y = K|\boldsymbol{z}_1, ..., \boldsymbol{z}_p)_l$$

6. Predict the class with the maximum class-specific probability estimate,

$$\hat{y} = \text{argmax}(\bar{\pi}_1, \bar{\pi}_2, ..., \bar{\pi}_K)$$

2.3.2.2 Feature selection

In addition to developing an accurate predictive model, we are interested in discriminating between important and unimportant features from the high-dimensional set of predictors. This can be accomplished by runnning the FANS algorithm once, i.e. by setting $L = 1$. Each feature in the dataset is represented by a single base learner, so the base learners that were selected for updating during the model fitting procedure at least once correspond to the important features.

2.4 Simulation study

We are interested in assessing the models' ability to accurately predict the outcome of new observations as well as their ability to select important features from the high-dimensional feature space. Thus, we examined:

1. Somers' D_{XY}, a measure of association between two ordinal variables. In this case, the ordinal variables are the observed and predicted values.

2. Misclassification rate $= \#\frac{(Y \neq \hat{Y})}{n}$

3. Class-specific misclassification rates

4. Sensitivity = $\frac{\text{Number of important features selected}}{\text{Total number of important features}}$

5. Specificity = $\frac{\text{Number of unimportant features } \textit{not} \text{ selected}}{\text{Total number of unimportant features}}$

To assess predictive performance, observations were split into a training set and a test set in each simulation. The models were fit on the training set, and predictions were made on the test set. However, to assess feature selection, the entire dataset was used to fit the models.

We examined the effects on performance of sample size ($n \in \{50, 100, 200, 300\}$), the number of classes in the outcome ($K \in \{3, 4\}$), and the number of FANS iterations ($L \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$). Furthermore, because FANS fits a nonlinear decision boundary, we simulated both a nonlinear and a linear decision boundary to assess how well the method adapts to each situation. The number of features, p, was set to 1,000 in all simulations, and publicly-available gene expression datasets were used as templates for simulating the data.

2.4.1 Simulations with $K = 3$

For simulations with $K = 3$, we downloaded the raw CEL files for GSE7390 [31] from the Gene Expression Omnibus (GEO) [32]. The dataset consisted of $p = 22,283$ gene expression values for $n = 198$ frozen tumor samples of systemically untreated node-negative breast cancer patients. However, two subjects with missing outcome variables were removed leaving $n = 196$. To preprocess the data, we performed robust multi-array average (RMA) normalization [33] and then removed the control probe sets as well as probe sets called absent in all samples by the MAS5 detection call algorithm [34].

Next, we fit a univariable cumulative logit model for each of the remaining 18,887 features to tumor grade (grade 1 < grade 2 < grade 3). There were 30 grade 1 samples, 83 grade 2 samples, and 83 grade 3 samples. The resulting p-values were adjusted for multiple comparisons using the Benjamini-Yekutieli procedure [35]. In order to restrict the calculation

of class-conditional means to coefficients that were in the same direction, we only examined features with a negative coefficient estimate. Of the 18,887 features, 1,537 had a negative coefficient estimate and an adjusted p-value less than 0.05.

Linear decision boundary For the simulations with a linear decision boundary, we calculated the class-conditional means of the 10 features with the smallest adjusted p-values. Let the class-conditional means be denoted by $\bar{x}_1, \bar{x}_2, \bar{x}_3$. Next, we estimated the mean of the features with an adjusted p-value greater than 0.05, denoted by $\bar{w}$. The sample covariance matrix, S, was calculated for 10 randomly sampled significant features and 990 randomly sampled nonsignificant features. Finally, using these estimates, the $\frac{n}{K}$ observations in each class were simulated from a multivariate normal distribution:

$$MVN(\boldsymbol{\mu} = ((\bar{\boldsymbol{x}}_{\boldsymbol{k}})_{10}, \bar{\boldsymbol{w}}_{990}), \Sigma = S), \quad k \in \{1, 2, 3\}$$

If n was not divisible by K then the remainder of observations were simulated in class K. For example, when $n = 50$, 16 observations were simulated in classes 1 and 2, while 18 observations were simulated in class 3.

The means of the first 10 features differ between classes, while the means of the remaining 990 features do not. Thus, ideally, the models should select the first 10 features because they are important for class separation but none of the other 990. Figure 2 shows the sample correlation matrix for the important features. Several features were highly correlated, which makes feature selection more difficult because correlated features contain redundant information. Thus, only one of the features is needed to predict the outcome. However, we would like to select all important features, regardless of their correlation, in order to learn more about the underlying biology and interaction between genes.

Nonlinear decision boundary For the simulations with a nonlinear decision boundary, we calculated the class-conditional means of all 1,537 significant features with a negative coefficient estimate. Let these means be represented by $\bar{x}_1, \bar{x}_2, \bar{x}_3$. Also, let the mean of the

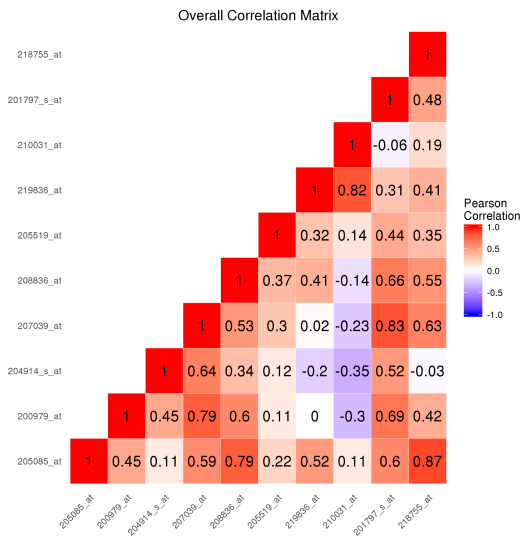

Fig. 2. The sample correlation matrix of the features simulated to be important to class separation for the linear decision boundary simulation with $K = 3$ classes.

features with an adjusted p-value greater than 0.05 be denoted by $\bar{w}$. Next, we estimated the class-conditional sample covariance matrices, S_1, S_2, S_3. Using these estimates, the observations in each class were generated from a multivariate normal distribution:

$$MVN(\boldsymbol{\mu} = ((\bar{\boldsymbol{x}}_{\boldsymbol{k}})_{10}, \bar{\boldsymbol{w}}_{990}), \Sigma = S_k), \quad k \in \{1, 2, 3\}$$

Thus, in both the linear and nonlinear settings, the first 10 features were simulated to be important to class separation while the remaining 990 were simulated to be unimportant. Figures 3, 4, and 5 show that the features simulated to differ between classes are highly correlated within each class.

2.4.2 Simulations with $K = 4$

For simulations with $K = 4$, we used the RMA-normalized gene expression data in the ALL R package [36], which consisted of $p = 12,626$ features measured on $n = 128$ acute lymphoblastic leukemia (ALL) patients. The dataset consisted of 95 B-cell and 33 T-cell leukemia patients. We limited our simulation to B-cell patients, and we examined tumor grade (B1 $<$ B2 $<$ B3 $<$ B4) as the ordinal outcome. The class-specific sample sizes were

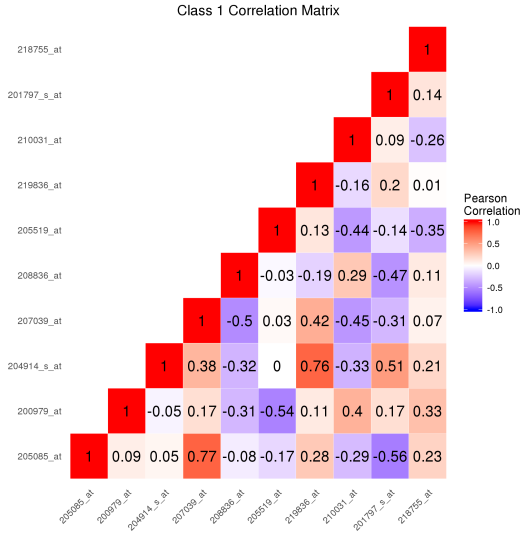

Fig. 3. Among observations in class 1, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 3$ classes.

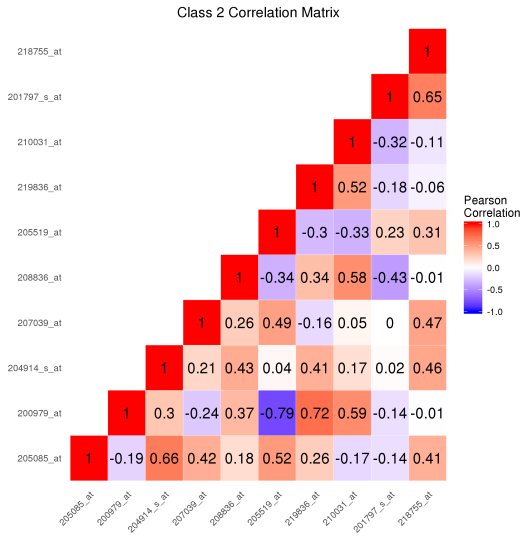

Fig. 4. Among observations in class 2, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 3$ classes.

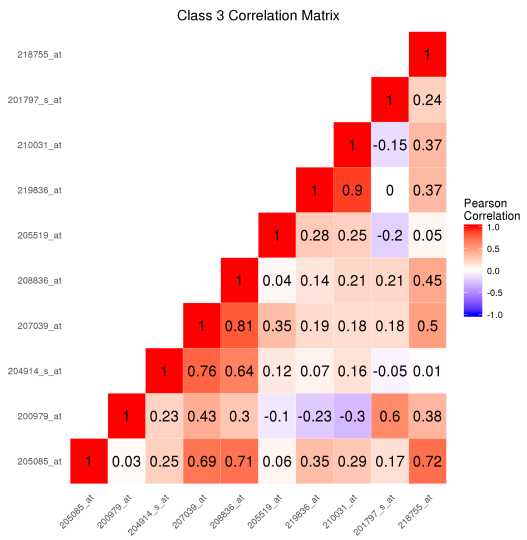

Fig. 5. Among observations in class 3, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 3$ classes.

19, 36, 23, and 12, respectively.

The data were simulated analogously to the simulations with 3 classes in the outcome. As in the previous simulation design, the features whose expression levels differed across classes were highly correlated both overall (Figure 6) and within each class (Figures 7, 8, 9, and 10).

2.5 Data analysis

We also compared the performance of the two Ordinal FANS approaches on a gene expression dataset. We downloaded the raw CEL files for GSE7803 [37] from the Gene Expression Omnibus (GEO) [32]. The dataset contained samples from normal squamous cervical epitheilia samples, high-grade squamous intraepithelial lesions (HSIL), and invasive squamous cell carcinomas of the cervix. After removing three test samples, 10 normal, 7 HSIL, and 21 carcinoma samples remained. The investigators used Affymetrix HG-U133A arrays to measure gene expression on 22,283 probe sets from the $n = 38$ samples, and we used the RMA method to normalize and obtain probe set expression summaries on these

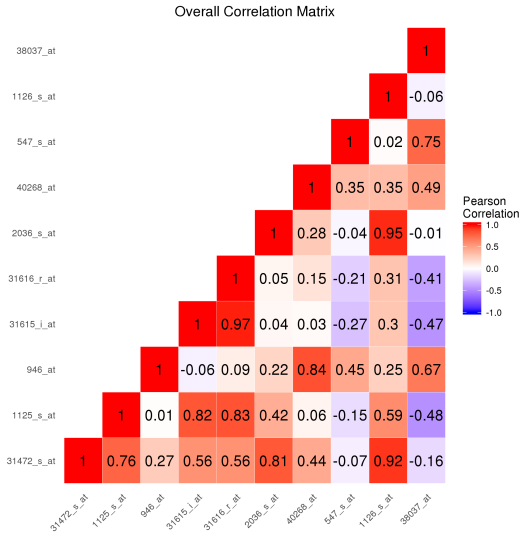

Fig. 6. The sample correlation matrix of the features simulated to be important to class separation for the linear decision boundary simulation with $K = 4$ classes.

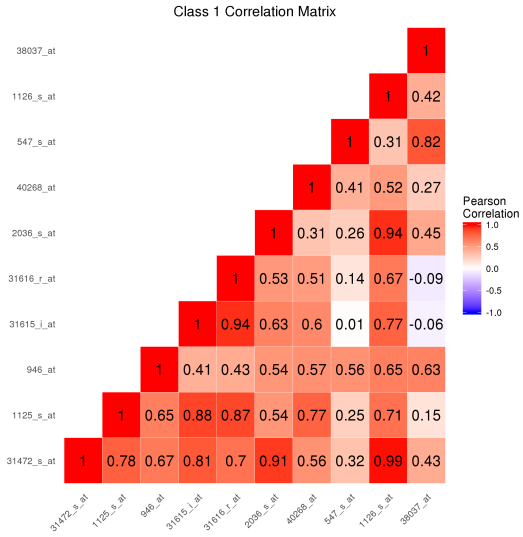

Fig. 7. Among observations in class 1, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 4$ classes.

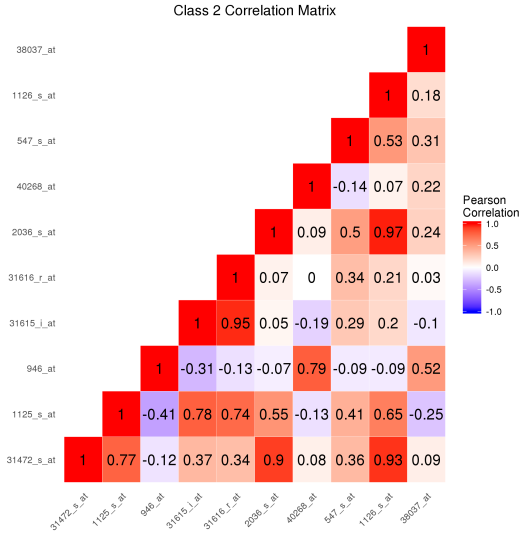

Fig. 8. Among observations in class 2, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 4$ classes.

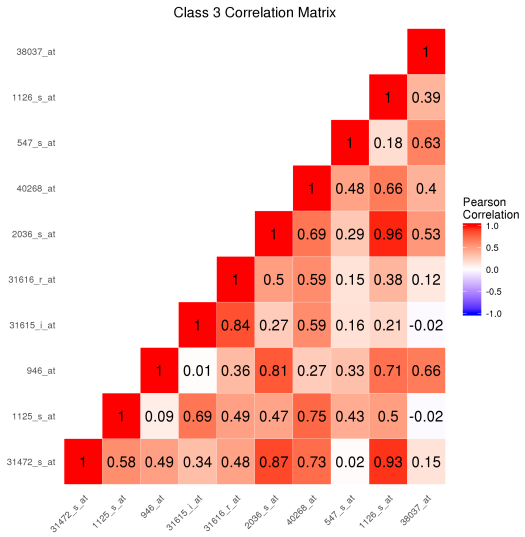

Fig. 9. Among observations in class 3, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 4$ classes.

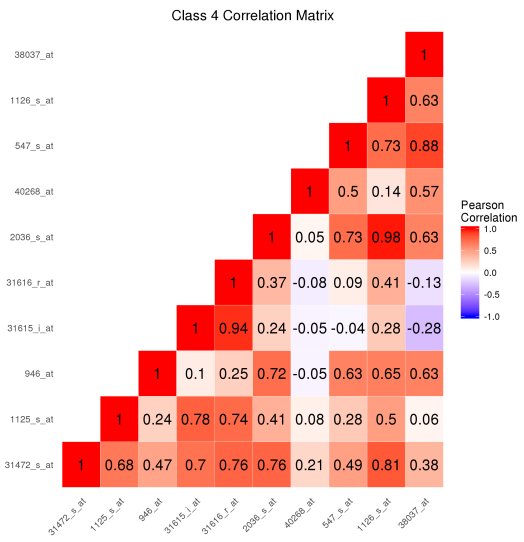

Fig. 10. Among observations in class 4, the sample correlation matrix of the features simulated to be important to class separation for the nonlinear decision boundary simulation with $K = 4$ classes.

features. After removing the control probe sets as well as probe sets that were not called present in any of the samples by the MAS5 detection call algorithm [38], $p = 13,667$ features remained.

We fit models using both Ordinal FANS approaches and evaluated their performances predicting the tissue classification (normal $<$ HSIL $<$ carcinoma) using the 10-fold cross-validation estimates of:

- Somers' D_{XY}
- Misclassification rate
- Class-specific misclassification rate

We also fit a model to the entire dataset using each approach and examined the features with non-zero coefficient estimates. These features are deemed important by the model fitting procedure and should be investigated further regarding their role in the progression from normal squamous cervical epithelia to HSIL to carcinoma.

2.6 Results

2.6.1 Simulation results

First, we examine performance in terms of Somers' D_{XY}. As the number of FANS iterations (i.e. L, the number of ordinal response classifiers that are averaged in the FANS ensemble) increases, the predictive performance improves monotonically for both the binary aggregation approach (Figure 11) and the boosting approach (Figure 12).

The effects of L, n, K, and the linearity of the true decision boundary on the performance in terms of the test set estimate of Somers' D_{XY} was similar for both approaches. First, we examine the simulations when the true decision boundary is nonlinear. As the number of ordinal response classifiers in the FANS ensemble increases, the association between the observed and predicted values increases and the variability of the Somers' D_{XY} estimates decreases. Thus, the model becomes more precise and more accurate. We suspected that the Ordinal FANS method would perform much better on larger samples sizes because of the data splitting and class-conditional kernel density estimation that each reduce the sample size for important aspects of the model fitting procedure. This is certainly the case with a nonlinear decision boundary. The improvement in the median as well as the variability of Somers' D_{XY} from a sample size of 50 to a sample size of 200 is substantial. From $n = 200$ to $n = 300$, the performance improves only slightly. Also, we expected that the number of classes in the outcome, K, would affect the performance. Holding the training set sample sizes equal, the within-class sample sizes are larger for smaller values of K, which results in more data to estimate the class-conditional kernel densities. The simulations confirmed our intuition that increasing K would deteriorate performance, especially for sample sizes greater than 50.

Next, we examine the simulations with a true linear decision boundary. When $K = 3$, the improvement obtained by building an ensemble is only in terms of the variance for sample sizes of 50 and 100, and no improvement is achieved for sample sizes of 200 and 300. Also,

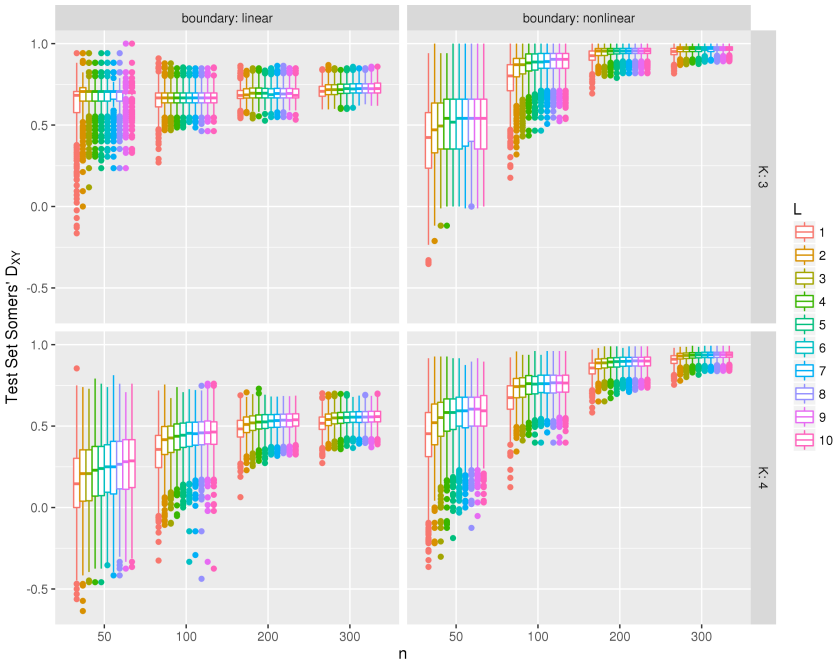

Fig. 11. Test set Somers' D_{XY} for models fit using the binary aggregation approach to the Ordinal FANS algorithm. As the number of classifiers in the FANS ensemble (L) increases, the predictive performance improves monotonically.

surprisingly, the sample size doesn't seem to have much of an effect on the median Somers' D_{XY} when $K = 3$. The only improvement seems to come as a reduction in variance. With 4 classes in the outcome, as L increases, the performance improves slightly in terms of the median Somers' D_{XY} but not in terms of the variance. An increase in sample size improves both the median and variance.

Both approaches seemed to benefit from more FANS iterations. Therefore, we compared the predictive performance of the two approaches with $L = 10$ FANS iterations (Figure 13 and Table 2). The boosting approach seemed to adapt better to a linear decision boundary better, outperforming the binary aggregation approach across the board. However, when the true decision boundary was nonlinear, the binary aggregation approach performed better, especially for larger sample sizes.

The overall misclassification rate conveyed the same message as Somers' D_{XY} (Table 2). However, the class-specific misclassification rates in Table 2 show that in simulations with

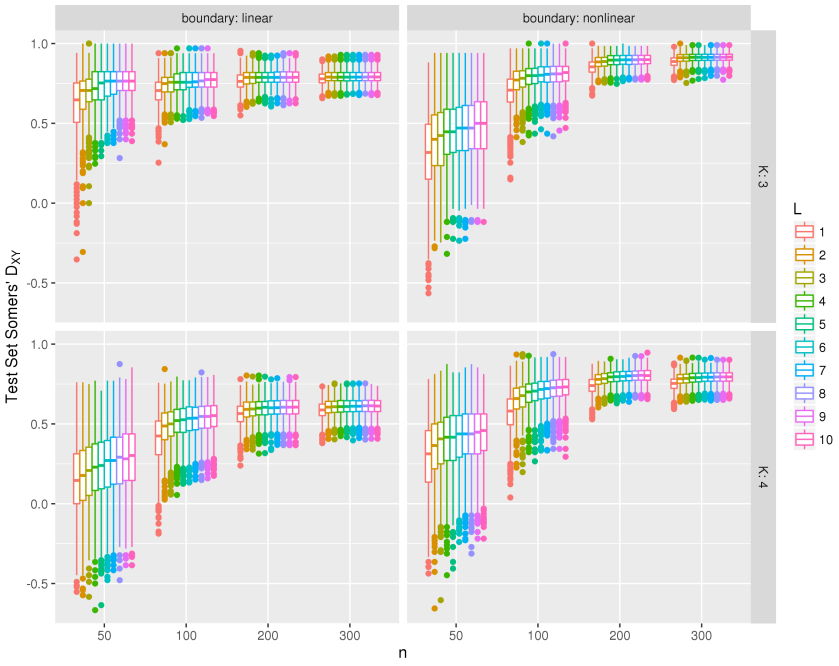

Fig. 12. Test set Somers' D_{XY} for models fit using the boosting approach to the Ordinal FANS algorithm. As the number of classifiers in the FANS ensemble (L) increases, the predictive performance improves monotonically.

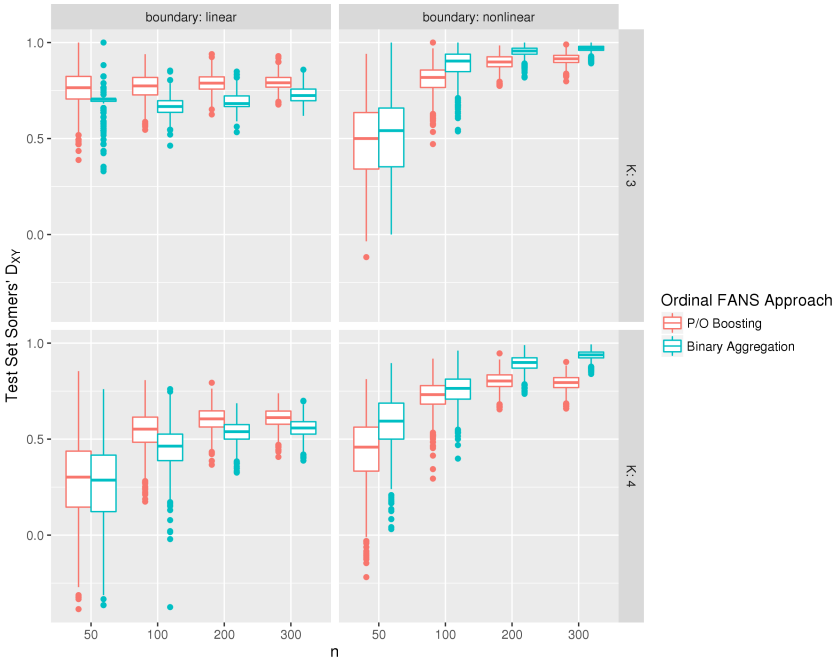

Fig. 13. Test set Somers' D_{XY} for models fit using the Ordinal FANS binary aggregation approach (blue) and the Ordinal FANS boosting approach (red). Results are presented for $L = 10$ FANS iterations.

$K = 3$ ordinal classes, errors were least common among observations in the third class. In contrast, the models had the most difficulty classifying observations in the second ordinal class. For simulations with $K = 4$ ordinal classes, the misclassification rates were lower for observations in the first and fourth classes than for observations in the second and third classes.

Overall, the proportional odds boosting approach seems to perform feature selection better than the binary aggregation approach as measured by sensitivity and specificity with $L = 1$ FANS iteration (Figures 14 and 15 and Table 3). With $K = 3$ classes, the boosting approach is more specific but slightly less sensitive than the binary aggregation approach regardless of the linearity of the decision boundary. With $K = 4$ classes, the boosting approach is more specific and more sensitive across all sample sizes and decision boundaries with one exception. When the sample size is smallest ($n = 50$) and the true decision boundary is linear, the binary aggregation approach was more specific but still less sensitive.

Furthermore, although not exactly monotonically, sample size did seem to improve the sensitivity of both approaches. That is, as the sample size increased the Ordinal FANS models selected more important truly features. However, the relationship between specificity and sample size was not as clear. In some scenarios, for instance when the true decision boundary was linear and there were $K = 3$ ordinal classes, sample size did not seem to have an effect on specificity. However, when the decision boundary remained linear but the number of classes increased to $K = 4$, specificity decreased for models fit using the binary aggregation approach but increased for models fit using the boosting approach as sample size increased. Finally, with a nonlinear decision boundary and $K = 4$ classes, specificity decreased for both approaches as sample size increased with an exception at $n = 300$. Specificity improved at $n = 300$ for models fit using the boosting approach.

If we examine the results of sensitivity and specificity together, there seems to be a weak negative association between the two. As the number of features included in the model increases, the number of truly important features selected will likely increase, which leads to

Table 2. Median test set performance for models fit using the Ordinal FANS binary aggregation approach and the Ordinal FANS boosting approach. Results are presented for $L = 10$ FANS iterations.

Approach	K	n	Boundary	Misclassification Rate (Class-Specific Rates)	Somers' D_{XY}
Boosting	3	50	Linear	0.26 (0.34, 0.43, 0.02)	0.76
Binary Aggregation	3	50	Linear	0.32 (0.45, 0.51, 0.01)	0.71
Boosting	3	100	Linear	0.23 (0.31, 0.36, 0.01)	0.77
Binary Aggregation	3	100	Linear	0.33 (0.37, 0.50, 0.01)	0.67
Boosting	3	200	Linear	0.21 (0.31, 0.30, 0.01)	0.79
Binary Aggregation	3	200	Linear	0.31 (0.30, 0.48, 0.01)	0.68
Boosting	3	300	Linear	0.21 (0.33, 0.28, 0.01)	0.79
Binary Aggregation	3	300	Linear	0.27 (0.28, 0.44, 0.01)	0.72
Boosting	3	50	Nonlinear	0.44 (0.24, 0.59, 0.09)	0.5
Binary Aggregation	3	50	Nonlinear	0.38 (0.14, 0.55, 0.02)	0.54
Boosting	3	100	Nonlinear	0.20 (0.17, 0.31, 0.08)	0.82
Binary Aggregation	3	100	Nonlinear	0.10 (0.05, 0.21, 0.01)	0.90
Boosting	3	200	Nonlinear	0.11 (0.09, 0.18, 0.05)	0.90
Binary Aggregation	3	200	Nonlinear	0.04 (0.04, 0.09, 0.005)	0.96
Boosting	3	300	Nonlinear	0.09 (0.07, 0.15, 0.04)	0.92
Binary Aggregation	3	300	Nonlinear	0.03 (0.02, 0.06, 0.003)	0.97
Boosting	4	50	Linear	0.66 (0.47, 0.72, 0.68, 0.65)	0.30
Binary Aggregation	4	50	Linear	0.69 (0.21, 0.74, 0.70, 0.72)	0.29
Boosting	4	100	Linear	0.53 (0.18, 0.62, 0.64, 0.53)	0.55
Binary Aggregation	4	100	Linear	0.60 (0.11, 0.68, 0.65, 0.64)	0.46
Boosting	4	200	Linear	0.49 (0.16, 0.57, 0.64, 0.52)	0.61
Binary Aggregation	4	200	Linear	0.54 (0.10, 0.61, 0.63, 0.55)	0.54
Boosting	4	300	Linear	0.48 (0.17, 0.57, 0.64, 0.52)	0.61
Binary Aggregation	4	300	Linear	0.52 (0.10, 0.57, 0.63, 0.49)	0.56
Boosting	4	50	Nonlinear	0.57 (0.35, 0.65, 0.62, 0.59)	0.46
Binary Aggregation	4	50	Nonlinear	0.50 (0.08, 0.57, 0.59, 0.11)	0.59
Boosting	4	100	Nonlinear	0.38 (0.10, 0.51, 0.53, 0.13)	0.73
Binary Aggregation	4	100	Nonlinear	0.32 (0.04, 0.37, 0.48, 0.01)	0.76
Boosting	4	200	Nonlinear	0.29 (0.07, 0.47, 0.45, 0.05)	0.80
Binary Aggregation	4	200	Nonlinear	0.15 (0.02, 0.21, 0.28, 0.01)	0.90
Boosting	4	300	Nonlinear	0.30 (0.12, 0.42, 0.44, 0.18)	0.79
Binary Aggregation	4	300	Nonlinear	0.09 (0.02, 0.14, 0.18, 0.004)	0.94

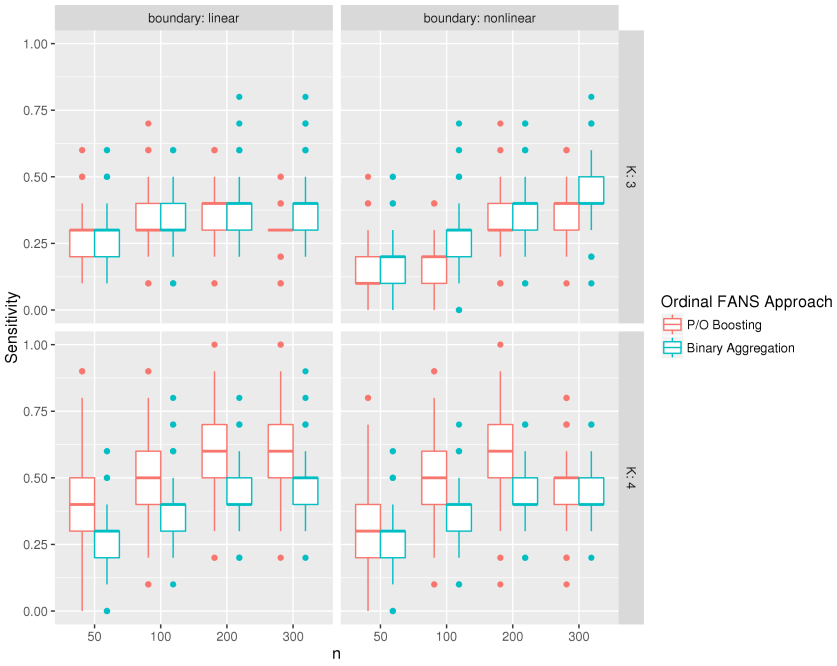

Fig. 14. Sensitivity for models fit using the Ordinal FANS binary aggregation approach (blue) and the Ordinal FANS boosting approach (red). Results are presented for $L = 1$ FANS iteration.

improved sensitivity. However, more truly unimportant features will also be selected which decreases specificity.

2.6.2 Data analysis results

The results of the analysis of GSE7803 in terms of Somers' D_{XY}, misclassification rate, and class-specific misclassification rates are shown in Figures 16, 17, 18 respectively.

In terms of Somers' D_{XY} and the misclassification rate, as the number of FANS iterations increases, the ability of the Ordinal FANS model fit using P/O boosting to discriminate between normal, HSIL, and carcinoma samples improves monotonically and stabilizes after 6 iterations. However, the performance of the Ordinal FANS model fit by aggregating penalized logistic regression models improves as the number of iterations increases from 1 to 3, but then oscillates from 4 iterations to 7 iterations, and then stabilizes at a value lower than its maximum, which was achieved at 3, 4, and 6 iterations. This type of behavior is commonly

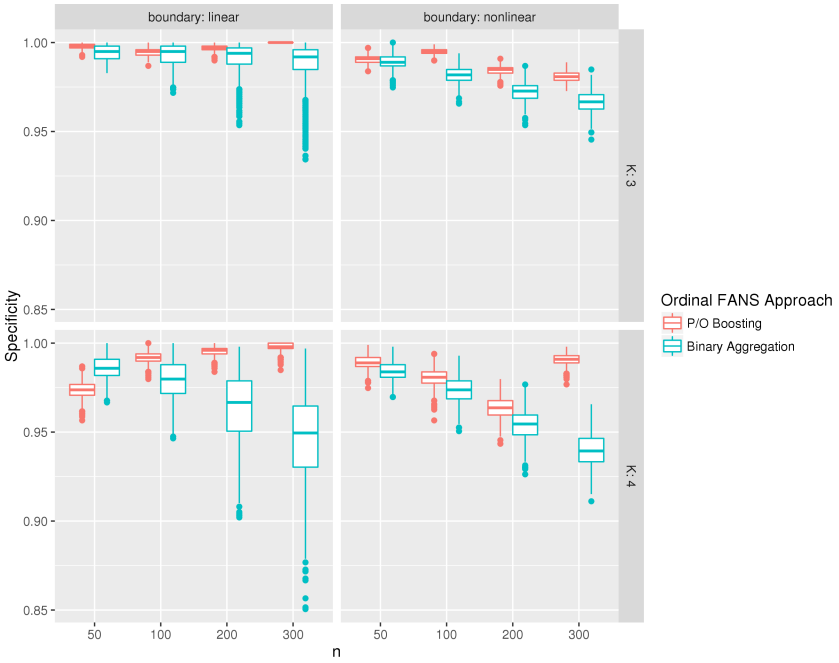

Fig. 15. Specificity for models fit using the Ordinal FANS binary aggregation approach (blue) and the Ordinal FANS boosting approach (red). Results are presented for $L = 1$ FANS iteration.

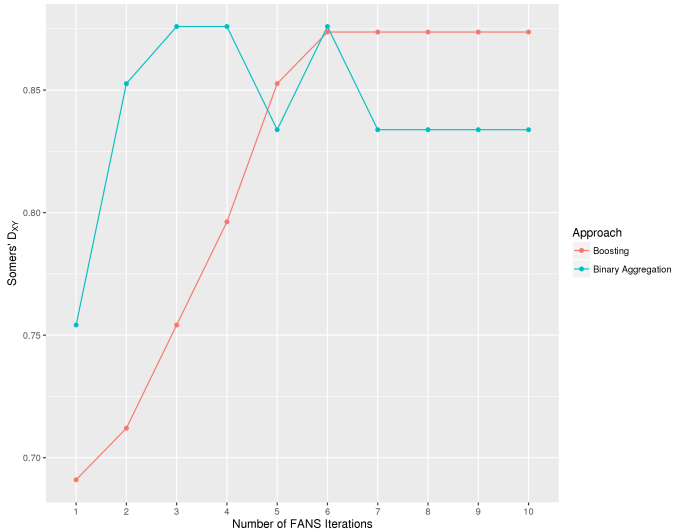

Fig. 16. 10-fold cross-validation estimates of Somers' D_{XY} for the classification of normal, HSIL, and cervical carcinoma samples from GSE7803. Results are shown for the P/O boosting approach (red) and binary model aggregation approach (blue).

Table 3. Median sensitivity and specificity for models fit using the Ordinal FANS binary aggregation approach and the Ordinal FANS boosting approach. Results are presented for $L = 1$ FANS iteration.

Approach	K	n	Boundary	Sensitivity	Specificity
Boosting	3	50	Linear	0.30	0.998
Binary Aggregation	3	50	Linear	0.30	0.995
Boosting	3	100	Linear	0.30	0.995
Binary Aggregation	3	100	Linear	0.30	0.995
Boosting	3	200	Linear	0.40	0.997
Binary Aggregation	3	200	Linear	0.40	0.994
Boosting	3	300	Linear	0.30	1.000
Binary Aggregation	3	300	Linear	0.40	0.992
Boosting	3	50	Nonlinear	0.10	0.991
Binary Aggregation	3	50	Nonlinear	0.20	0.989
Boosting	3	100	Nonlinear	0.20	0.995
Binary Aggregation	3	100	Nonlinear	0.30	0.982
Boosting	3	200	Nonlinear	0.30	0.985
Binary Aggregation	3	200	Nonlinear	0.40	0.973
Boosting	3	300	Nonlinear	0.40	0.981
Binary Aggregation	3	300	Nonlinear	0.40	0.967
Boosting	4	50	Linear	0.40	0.974
Binary Aggregation	4	50	Linear	0.30	0.986
Boosting	4	100	Linear	0.50	0.992
Binary Aggregation	4	100	Linear	0.40	0.980
Boosting	4	200	Linear	0.60	0.996
Binary Aggregation	4	200	Linear	0.40	0.967
Boosting	4	300	Linear	0.60	0.998
Binary Aggregation	4	300	Linear	0.50	0.949
Boosting	4	50	Nonlinear	0.30	0.989
Binary Aggregation	4	50	Nonlinear	0.30	0.984
Boosting	4	100	Nonlinear	0.50	0.981
Binary Aggregation	4	100	Nonlinear	0.40	0.974
Boosting	4	200	Nonlinear	0.60	0.964
Binary Aggregation	4	200	Nonlinear	0.40	0.955
Boosting	4	300	Nonlinear	0.50	0.991
Binary Aggregation	4	300	Nonlinear	0.40	0.939

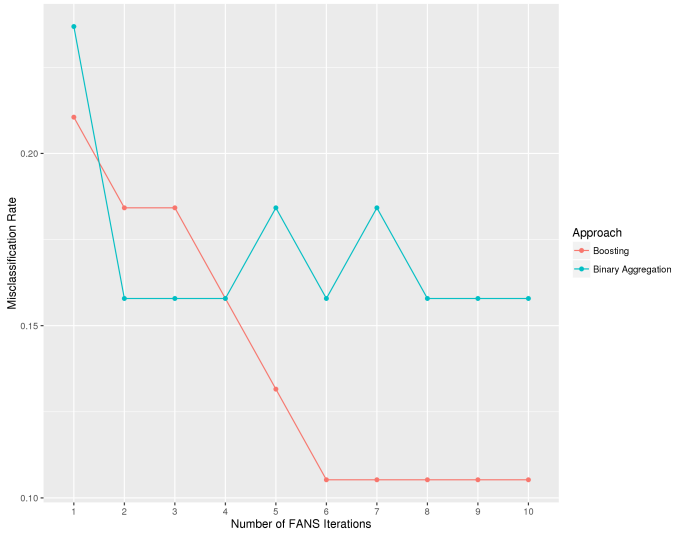

Fig. 17. 10-fold cross-validation estimates of the misclassification rate for the classification of normal, HSIL, and cervical carcinoma samples from GSE7803. Results are shown for the P/O boosting approach (red) and binary model aggregation approach (blue).

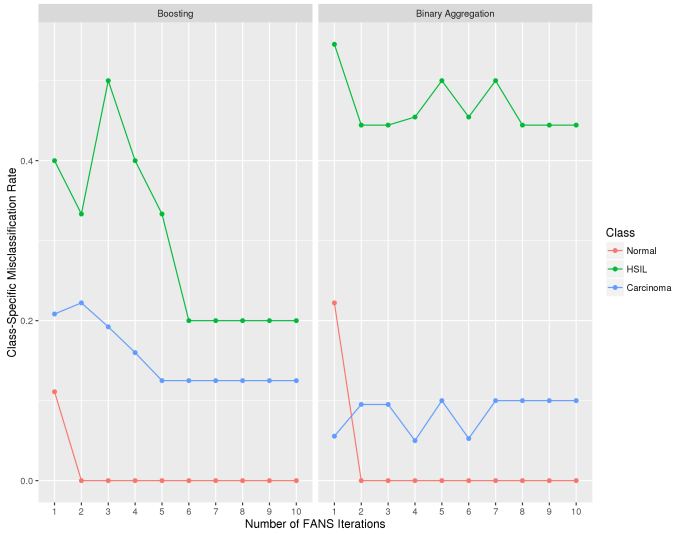

Fig. 18. 10-fold cross-validation estimates of the class-specific misclassification rates for the classification of normal (red), HSIL (green), and cervical carcinoma (blue) samples from GSE7803. Results are shown for the P/O boosting approach (red) and binary model aggregation approach (blue).

seen in other ensemble methods, such as bagging and random forests. As the number of classifiers in the ensemble learner increases, the performance tends to improve rapidly, then oscillate, and finally stabilize as the number of trees reaches a sufficient number. However, we did not see this type of behavior in the simulation study. One potential reason for this is that the classes were perfectly or nearly perfectly balanced in the simulation, but the classes in the gene expression dataset were extremely imbalanced. This imbalance along with the small sample size in the HSIL class ($n = 7$) may have led to an increase in the variance of the classifiers within the ensemble. In the future, we plan on examining the effect of class imbalance as well as within-class sample size on performance.

Figure 18 conveys additional information regarding the misclassification rates of the two approaches. The binary aggregation approach performs as well or better than the boosting approach with respect to classifying normal and carcinoma samples. Both methods have more of a difficult time classifying HSIL samples, but as the number of FANS iterations increases, the gap between the two methods widens. At 10 iterations, the boosting approach vastly outperforms the binary aggregation approach in classifying HSIL samples, which explains the difference in overall misclassification rates seen in Figure 17.

The Ordinal FANS boosting approach and binary aggregation approach included 10 and 9 non-zero coefficients, respectively. The genes corresponding to these coefficients are listed in Table 4. The gene symbols for the features that had non-zero coefficient estimates in both models are SPAG5, EDN3, B4GALT4, and UPK1A. Upregulation of SPAG5 has been show to predict poor prognosis among cervical cancer patients [39], and the genes EDN3 and UPK1A were found to be down regulated in cancerous cervix compared to normal cervical epithelium [40, 41]. No study has published a link between B4GALT4 and squamous cell carcinoma of the cervix, which suggests that the feature was either a false positive in both models or a novel discovery.

Table 4. Genes deemed important in the classification of normal, HSIL, and cervical carcinoma samples by the Ordinal FANS models fit to GSE7803. The genes listed were included in either the model fit using the boosting approach or in the model fit using the binary aggregation approach (or in both models). A check mark denotes that the gene was included in the model fit using the given approach. One Affymetrix probe id could not be matched to a unique gene symbol and is denoted by <NA>.

Gene Symbol	Binary Aggregation	Boosting
PLOD2	✓	
SPAG5	✓	✓
ACOX2	✓	
SLC16A7	✓	
EDN3	✓	✓
U2SURP	✓	
B4GALT4	✓	✓
UPK1A	✓	✓
PITPNA		✓
MTF2		✓
CCND1		✓
KRT4		✓
TIPIN		✓
BRWD1		✓
<NA>	✓	

2.7 R package

We developed R [26] code for both approaches of the Ordinal FANS method and assembled each into a unique R package. We followed the S3 object system, so we defined methods for generic functions including `print()`, `summary()`, `predict()`, `plot()`, and `coef()`. The default method for fitting the model takes the design matrix and the ordinal response as the first two arguments and performs the analysis by iterating through the FANS algorithm using our predefined functions. Additional arguments for both approaches include `newx` if the user wishes to make predictions on new data using the fitted model, the number of FANS iterations (`niter`), a random seed to set so the model is reproducible (`seed`), `scale`, which determines whether the predictors will be centered and scaled, and `parallel`, which determines whether or not the `niter` FANS iterations will be run in parallel. Additionally, the function for fitting the model using boosting has as arguments: the multiplicative step size, `eps`, and the number of boosting iterations, `mstop`. Furthermore, we wrote a function named `cv.modelselect` for estimating the tuning parameter, `mstop`, by maximizing the K-fold CV

estimate of Somers' D_{XY}.

CHAPTER 3

COMPARATIVE ANALYSIS

3.1 Methods

We compared the following methods to the method that we developed in the previous chapter, Ordinal FANS. From here on, we refer to the first approach of the Ordinal FANS method, which aggregates binary response models and was described in Chapter 2, Section 2, as Ordinal FANS 1 and the second approach of proportional odds boosting, described in Chapter 2, Section 3, as Ordinal FANS 2. We performed a simulation study as well as several data analyses to assess how well each method a) predicts the outcome of a future observation, measured using Somers' D_{XY}, misclassification rate, and class-specific misclassification rates and b) performs feature selection, measured using sensitivity and specificity. (All measures were defined in Chapter 2, Section 4.)

3.1.1 Weighted k-Nearest Neighbors

The k-nearest neighbors method is one of the simplest and most intuitive classification techniques. Assume there are n observations in the training data,

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}.$$

The central idea is that, to classify a new observation, $\boldsymbol{x}$, we find the k observations from the training set closest to the new observation in terms of a distance measure, $d(\boldsymbol{x}, \boldsymbol{x}_i)$ on the feature vectors. The predicted value of the new observation is the class with the largest representation among the k neighbors. The k nearest neighbors are given by $(y_{(1)}, x_{(1)}), (y_{(2)}, x_{(2)}), ..., (y_{(k)}, x_{(k)})$. A commonly used distance measure is the Minkowski

distance:

$$d(\boldsymbol{x_i}, \boldsymbol{x}_j) = \left(\sum_{r=1}^{p} |x_{ir} - x_{jr}|^q \right)^{1/q}.$$

Note that $q = 1$ results in the absolute distance and $q = 2$ results in the Euclidean distance. To avoid introducing bias into the selection of neighbors, the predictors must first be standardized (this can be done differently for continuous, ordinal, and nominal predictors) [42]. In the weighted k-nearest neighbors method, the distances of the k nearest neighbors are standardized by dividing each distance by the distance to the $(k+1)^{st}$ nearest observation:[42]

$$D(\boldsymbol{x}, \boldsymbol{x}_i) = \frac{d(\boldsymbol{x}, \boldsymbol{x}_i)}{d(\boldsymbol{x}, \boldsymbol{x}_{k+1})}, \quad i = 1, 2, ..., k$$

Then, these standardized distances are converted to weights using a kernel function, K, which has the following properties,

- $K(D) \geq 0$.
- $K(D)$ is maximized when $D = 0$.
- $K(D)$ is monotonically decreasing function of D for $D \geq 0$.

Now, the estimated class probability distribution becomes [42]

$$\hat{P}(y = c|\boldsymbol{x}, T) = \frac{\sum_{i=1}^{k} K(D(\boldsymbol{x}, \boldsymbol{x}_{(i)}))I(y_{(i)} = c)}{\sum_{i=1}^{k} K(D(\boldsymbol{x}, \boldsymbol{x}_{(i)}))} \tag{3.1}$$

Commonly, the predicted class for the new observation is the class with the largest predicted probability, which is equivalent to the mode of equation 3.1. For an ordinal outcome, one can use the median of the estimated class distribution as the predicted outcome [42]. The weighted k-nearest neighbor method is implemented in the `kknn R` package on the Comprehensive R Archive Network (CRAN).

3.1.2 Component-Wise Proportional Odds Boosting

According to Hastie et. al, boosting is one of the most powerful learning ideas introduced in the last twenty years [15]. It is one method in a class of ensemble schemes that combines

multiple weak function estimates to form one aggregated estimator. The aim of boosting is to estimate a function, $f^*(.)$, that minimizes the risk:

$$R = E[L(Y, f(X)))],$$

where L is the loss function chosen dependent on the structure of the response [14]; often, the loss function is equal to the negative log-likelihood. Gradient boosting searches for the solution by following the steepest path down the gradient of the empirical risk function,

$$\frac{1}{n}\sum_{1}^{n} L(Y_i, f(X_i))$$

in function space. Gradient boosting can be applied to many different estimators including regression trees, generalized linear models (GLMs), and Cox proportional hazards models. Generally, we assume that f belongs to a parameterized class of functions $F(X, \boldsymbol{P})$. The component-wise gradient boosting algorithm for GLMs was extended by Schmid et al. to the ordinal response setting. The method, called Proportional Odds (P/O) boosting, fits a cumulative logit model as follows [16]:

1. Let $\hat{F}_m$ and $\hat{\boldsymbol{\theta}}_m$ denote the vectors of estimates, $(\hat{F}(\boldsymbol{x}_1), ..., \hat{F}(\boldsymbol{x}_n))$ and $(\hat{\theta}_1, ..., \hat{\theta}_{K-1})$ at step m. Set $m = 0$ and initialize $\hat{F}_0$ and $\hat{\boldsymbol{\theta}}_0$.

2. Specify the base learners.

3. Increase m by 1 and compute the negative gradient of the loss with respect to F and, for each observation, evaluate it at the estimates of the previous iteration, $\hat{F}_{m-1}(\boldsymbol{x}_i), \hat{\boldsymbol{\theta}}_{m-1}$:

$$\boldsymbol{g}_m(\boldsymbol{x}) = (-g_m(\boldsymbol{x}_i))_{i=1,...,n} = \left(-\frac{\delta}{\delta F} L(y_i, F, \boldsymbol{\theta})|_{F=\hat{F}_{m-1}(\boldsymbol{x}_i), \boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{m-1}}\right)_{i=1,...,n}$$

4. Fit each of the base learners to the negative gradient vector $\boldsymbol{g}_m(\boldsymbol{x})$. Each model results in a different estimate of $\boldsymbol{g}_m(\boldsymbol{x})$.

5. Select the model that fit the negative gradient best as determined by the R^2 goodness-of-fit criterion. Let $\hat{\boldsymbol{g}}_m(\boldsymbol{x})$ represent the vector of fitted values for that model.

6. Update the current function estimate :

$$\hat{F}_m = \hat{F}_{m-1} + \nu \hat{\boldsymbol{g}}_m(\boldsymbol{x})$$

Note that ν is a pre-specified small step length factor between 0 and 1 and is not highly influential as long as it is sufficiently small (e.g. $\nu = 0.1$) [14].

7. Treating the function estimate as fixed, estimate $\boldsymbol{\theta}_m$ by minimizing the empirical risk, $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{F}_m, \boldsymbol{\theta})$.

8. Repeat steps 3-7 until $m = M$, where M represents the stopping iteration chosen by cross-validation.

The final function estimate is given by

$$\begin{aligned}\hat{F}_M =& \sum_{m=0}^{M} \hat{F}_m \\ =& \hat{F}_0 + \sum_{m=1}^{M} \nu \hat{\boldsymbol{g}}_m(\boldsymbol{x})\end{aligned}$$

Component-wise boosting was implemented in the `mboost R` package and is available on CRAN. To fit a model with proportional odds boosting, we must specify the `family` argument as `PropOdds()`.

3.1.3 Generalized Monotone Incremental Forward Stagewise Method

The generalized monotone incremental forward stagewise (GMIFS) algorithm was developed by Hastie et al. [13] in order to obtain a penalized solution to overparameterized linear and logistic regression models. The GMIFS method was extended by Archer et al. for fitting logit, probit, and complimentary log-log link ordinal response models (cumulative, adjacent category, stereotype, forward continuation ratio, and backward continuation ratio) to high-throughput genomic data[7]. The GMIFS algorithm for ordinal response modeling is as follows[7]:

1. Enlarge the predictor space as $\tilde{\mathbf{X}} = [\mathbf{X} : -\mathbf{X}]$, where $\mathbf{X}$ represents the standardized design matrix.

2. Initialize the α's to their empirical values.

3. For step $s = 0$, all components of $\hat{\boldsymbol{\beta}}^{(s)}$ are initialized to 0. That is, $\hat{\beta}_1 = \hat{\beta}_2 = ... = \hat{\beta}_P = \hat{\beta}_{P+1} = ... = \hat{\beta}_{2P} = 0$.

4. Find $m = \underset{p}{\operatorname{argmin}} -\frac{\delta \log L}{\delta \beta_p}$ at the current estimate $\hat{\boldsymbol{\beta}}^{(s)}$.

5. Update $\hat{\beta}_m^{(s+1)} = \hat{\beta}_m^{(s)} + \epsilon$, where ϵ is a small constant such as 0.001.

6. Estimate the α's by maximum likelihood, treating $\hat{\boldsymbol{\beta}}^{(s)}$ (from step 5) as fixed.

7. Repeat steps 4 to 6 until the difference between two successive log-likelihoods is smaller than a pre-specified tolerance, τ.

The predictor space is enlarged in step 1 in order to avoid the computationally-intensive process of taking the second derivative of the log-likelihood. Once the algorithm has converged, the penalized solution is given by $\hat{\beta}_p = \hat{\beta}_p - \hat{\beta}_{p+P}, p = 1, ..., P$ [7]. Furthermore, we can choose the model resulting from the convergence of the algorithm as our final model, or we may select the model that minimizes either the AIC or BIC criteria. The latter generally have better predictive accuracy on an independent test set because the model resulting from the convergence of the algorithm is often overfit.

We utilized the `ordinalgmifs R` package, which is available on CRAN to fit cumulative logit models with the ordinal GMIFS method.

3.2 Simulation study

Recall the simulation designs from Chapter 2, Section 4. For simulations with a linear decision boundary, the data were generated by class from a multivariate normal distribution:

$$MVN(\boldsymbol{\mu} = ((\bar{x}_k)_{10}, \bar{w}_{990}), \Sigma = S), \quad k \in \{1, ..., K\}$$

The class-conditional means of significant features, $\bar{x}_1, \bar{x}_2, ..., \bar{x}_K$, the overall mean of non-significant features, $\bar{w}$, and the sample covariance matrix S were estimated from gene expression datasets, which served as templates for the simulations.

For simulations with a nonlinear decision boundary, the data were simulated similarly except that each class was simulated using the class-conditional sample covariance matrices $S_1, S_2, ..., S_K$ estimated from the data:

$$MVN(\boldsymbol{\mu} = ((\bar{x}_k)_{10}, \bar{w}_{990}), \Sigma = S_k), \quad k \in \{1, ..., K\}$$

We examined the effects of the number of classes in the outcome, $K \in \{3, 4\}$, sample size, $n \in \{50, 100, 200, 300\}$, and linearity of the decision boundary (linear and nonlinear) on the predictive performance and feature selection performance. In all cases, the first 10 features were simulated to be important to class separation, while the remaining 990 features were not. Thus, to assess feature selection, we examined sensitivity and specificity are defined as:

- Sensitivity = $\frac{\text{Number of important features selected}}{\text{Total number of important features}}$
- Specificity = $\frac{\text{Number of unimportant features } \textit{not} \text{ selected}}{\text{Total number of unimportant features}}$

To assess predictive performance, we examined the association between the predicted and observed values in an independent test set using Somers' D_{XY}, misclassification rate, and class-specific misclassification rates.

3.3 Data analyses

For each of the following datasets, we fit a predictive model using each of the previously defined methods. Performance was measured using 10-fold cross-validation estimates of Somers' D_{XY}, misclassification rate, and class-specific misclassification rates.

3.3.1 Progression to cervical cancer

Nearly all cases of cervical cancer are caused by human papillomavirus (HPV), the most common type of sexually transmitted infection [43, 44]. Cervical cancer generally progresses slowly but may not have any noticeable symptoms, so The U.S. Preventive Services Task Force recommends regular screening for women ages 21 to 65 [44]. Screening can take the form of a Papanicolaou smear, or pap smear, every three years or a pap smear in combination with a test for the presence of greater than two high-risk or carcinogenic HPV types every five years[44]. HPV testing has a higher sensitivity but a lower specificity than the pap smear [44].

Cervical disease progresses from normal epithelium, to low-grade squamous intraepithelial lesion (LSIL), to high-grade squamous intraepithelial lesion (HSIL), and then to invasive carcinoma. The mechanism allowing cells to progress from intraepithelial lesions to malignancy is still an active area of research [37]. Our goal in this analysis was to compare the methods' abilities to determine a sample's stage in this progression.

3.3.1.1 Data preprocessing

We downloaded the raw CEL files for GSE7803 [37] from the Gene Expression Omnibus (GEO) and preprocessed and normalized the data as we described in Chapter 2, Section 5. Recall that the preprocessed and normalized dataset contained $n = 38$ samples, including 10 normal samples, 7 high grade intraepithelial lesion (HSIL) samples, and 21 carcinoma samples. Furthermore, there were $p = 13,667$ features.

3.3.2 Progression to malignant melanoma

Melanoma occurs when malignant cells form in the skin cells that produce melanin, the pigment that gives skin its color. Generally, it occurs in areas of the skin that are exposed to sunlight, but it can be found anywhere on the body, including in the eyes. Melanoma is the sixth most common type of cancer, and the rates for new cases have been rising on average

1.4% over the past 10 years [45]. Early diagnosis is dependent on visual inspection of a mole or nevus, relying on the "ABCDE" rule, which describes the early features of melanoma, namely [46, 47]:

- Asymmetry
- Border that is irregular
- Color that is uneven
- Diameter that is large
- Evolving over time

If the melanoma is caught early, while it is confined to the primary site, 5-year relative survival is 98.4%. However, 5-year relative survival is drastically lower if the cancer spreads to the regional lymph nodes (62.4%) or metastasizes (17.9%) [45]. Thus, early diagnosis is imperative. A multigenic classifier to distinguish between normal, benign nevus, and malignant melanoma skin samples would be useful to supplement the visual inspection of the mole to aid in determining a diagnosis. This classifier has the potential to give physicians increased confidence in a diagnosis made from visual inspection of the skin, which could shorten the time to treatment. Furthermore, the genes included in the classifier could be investigated as potential molecular targets for new therapies.

3.3.2.1 Data preprocessing

We downloaded GSE3189 from the Gene Expression Omnibus [32, 48]. The investigators used Affymetrix HG-U133A GeneChips to measure gene expression from skin tissue specimens. In the dataset, there were $n = 70$ observations, of which 7 were normal skin samples, 18 were nevus samples, and 45 were malignant melanoma samples. There were $p = 22,215$ features after excluding the control probe sets. The $\log_2$ transformed expression values of these features were used in the analysis.

3.3.3 Pathogenesis of hepatocellular carcinoma

Cirrhosis develops after years of chronic liver disease and results in a destruction of liver cells, which may eventually lead to the development of cancerous nodules. Consequently, the majority of patients with hepatocellular carcinoma (HCC), the most common form of primary liver cancer, suffer concurrently from liver cirrhosis [49]. Thus, a cirrhotic liver has been described as being pre-malignant [50]. Furthermore, infection with hepatitis C virus (HCV) is a major risk factor for the development of HCC.

Methylation, an epigenetic event in which a methyl group attaches to a 5' cytosine in a CG dinucleotide, or CpG site, is thought to lead to chromosomal instability in some cases. These CpG sites commonly occur in clusters called CpG islands. When these CpG islands are located in gene promoter regions of tumor suppressor genes and are densely methylated, or "hypermethylated," transcription of the gene can be silenced. Further, a gene promoter region can be sparsely methylated, or "hypomethylated," leading to increased expression of the gene. Hypermethylation of tumor suppression genes and hypomethylation of proto-oncogenes have been implicated in various types of cancer.

3.3.3.1 Data preprocessing

We downloaded GSE18081 from the Gene Expression Omnibus [32]. The investigators profiled p =1,505 CpG sites in $n = 76$ liver tissues using the Illumina GoldenGate Methylation BeadArray Cancer Panel I [50]. Included in the data were HCV-HCC tumor samples and their adjacent non-cancerous cirrhotic tissue samples, independent non-cancerous HCV-cirrhotic tissues, and normal liver tissues. We removed the matched cirrhotic samples from subjects with HCC as well as technical replicate samples, leaving 20 normal, 16 non-HCC HCV-cirrhotic, and 20 cirrhotic HCV-HCC samples. We removed 10 CpG sites for which one or more samples had a missing value as well as 26 CpG sites that had a variance of 0. Our goal was to use the remaining p =1,469 predictors to classify tissue samples as normal, HCV-cirrhotic (pre-malignant), or HCV-HCC (malignant).

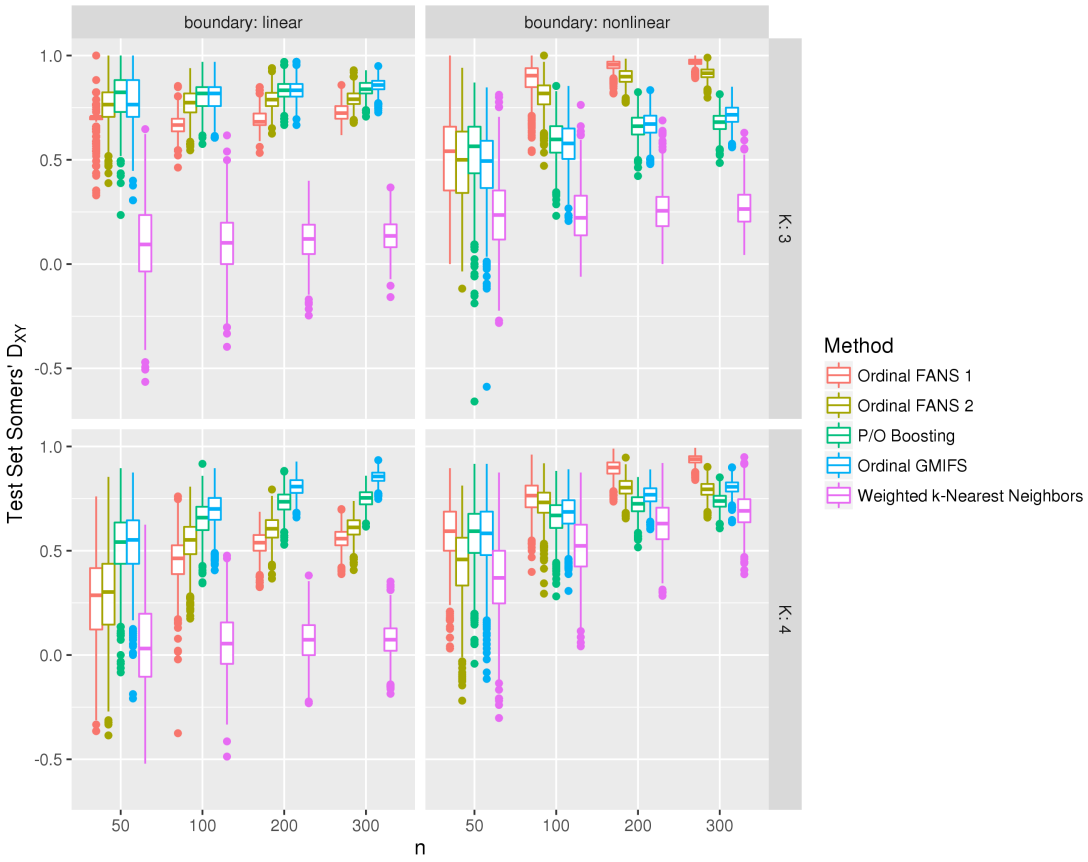

Fig. 19. Simulation results: Distribution of test set Somers' D_{XY} estimates for varying K, n, and decision boundaries for each method in the comparative analysis.

3.4 Results

3.4.1 Simulation study

3.4.1.1 Prediction

The results of the simulation study in terms of Somers' D_{XY} are displayed in Figure 19.

True linear decision boundary When the true decision boundary separating the K classes was linear, weighted k-nearest neighbors clearly delivered the worst performance. With $K = 3$ classes in the outcome, sample size had a marginal effect on the performance of

Table 5. Median test set misclassification rates and class-specific misclassification rates (in parentheses) by method.

			Misclassification Rate (Class-Specific Rates)				
K	n	Boundary	Ordinal FANS 1	Ordinal FANS 2	P/O Boosting	Ordinal GMIFS	Weighted k-Nearest Neighbors
3	50	Linear	0.323 (0.447, 0.508, 0.011)	0.259 (0.343, 0.431, 0.024)	0.219 (0.292, 0.372, 0.008)	0.226 (0.297, 0.384, 0.008)	0.635 (0.638, 0.681, 0.546)
3	100	Linear	0.330 (0.365, 0.497, 0.011)	0.231 (0.310, 0.362, 0.012)	0.191 (0.261, 0.313, 0.003)	0.195 (0.261, 0.313, 0.003)	0.630 (0.627, 0.661, 0.561)
3	200	Linear	0.306 (0.296, 0.476, 0.009)	0.209 (0.312, 0.304, 0.011)	0.168 (0.240, 0.261, 0.001)	0.163 (0.230, 0.257, 0.001)	0.622 (0.614, 0.658, 0.540)
3	300	Linear	0.274 (0.279, 0.443, 0.009)	0.209 (0.326, 0.282, 0.012)	0.160 (0.232, 0.247, 0.001)	0.143 (0.209, 0.220, 0.000)	0.617 (0.605, 0.659, 0.514)
3	50	Nonlinear	0.383 (0.140, 0.549. 0.020)	0.439 (0.235, 0.591, 0.093)	0.433 (0.468, 0.611, 0.164)	0.468 (0.470, 0.625, 0.142)	0.508 (0.596, 0.362, 0.084)
3	100	Nonlinear	0.104 (0.053, 0.212, 0.007)	0.196 (0.173, 0.309, 0.082)	0.393 (0.426, 0.560, 0.107)	0.327 (0.422, 0.560, 0.092)	0.438 (0.565, 0.200, 0.052)
3	200	Nonlinear	0.045 (0.036, 0.088, 0.005)	0.107 (0.085, 0.177, 0.048)	0.339 (0.392, 0.505, 0.057)	0.327 (0.387, 0.488, 0.041)	0.438 (0.557, 0.115, 0.018)
3	300	Nonlinear	0.030 (0.025, 0.060, 0.003)	0.088 (0.069, 0.151, 0.036)	0.322 (0.388, 0.483, 0.043)	0.286 (0.368, 0.433, 0.019)	0.421 (0.549, 0.090, 0.010)
4	50	Linear	0.691 (0.215, 0.738, 0.700, 0.723)	0.661 (0.466, 0.721, 0.679, 0.647)	0.534 (0.213, 0.635, 0.667, 0.569)	0.542 (0.159, 0.636, 0.645, 0.517)	0.744 (0.710, 0.752, 0.746, 0.749)
4	100	Linear	0.605 (0.113, 0.680, 0.650. 0.636)	0.532 (0.181, 0.618, 0.643, 0.534)	0.443 (0.092, 0.510, 0.606, 0.460)	0.403 (0.054, 0.425, 0.591, 0.426)	0.738 (0.681, 0.754, 0.741, 0.729)
4	200	Linear	0.543 (0.103, 0.609, 0.635, 0.552)	0.487 (0.163, 0.572, 0.636, 0.518)	0.364 (0.041, 0.363, 0.573, 0.425)	0.277 (0.011, 0.146, 0.514, 0.371)	0.729 (0.659, 0.746, 0.735, 0.713)
4	300	Linear	0.520 (0.103, 0.574, 0.630, 0.485)	0.482 (0.165, 0.568, 0.641, 0.521)	0.345 (0.032, 0.327, 0.565, 0.417)	0.216 (0.002, 0.049, 0.439, 0.329)	0.731 (0.656, 0.750, 0.736, 0.716)
4	50	Nonlinear	0.498 (0.078, 0.566, 0.590, 0.113)	0.568 (0.349, 0.651, 0.621, 0.593)	0.499 (0.146, 0.551, 0.643, 0.574)	0.498 (0.113, 0.552, 0.612, 0.546)	0.476 (0.281, 0.436, 0.403, 0.570)
4	100	Nonlinear	0.322 (0.036, 0.368, 0.480, 0.010)	0.380 (0.101, 0.508, 0.533, 0.134)	0.429 (0.088, 0.433, 0.586, 0.514)	0.402 (0.073, 0.384, 0.568, 0.496)	0.354 (0.185, 0.286, 0.267, 0.485)
4	200	Nonlinear	0.149 (0.021, 0.209, 0.278, 0.005)	0.293 (0.067, 0.474, 0.454, 0.048)	0.380 (0.065, 0.310, 0.574, 0.498)	0.323 (0.031, 0.177, 0.543, 0.469)	0.260 (0.111, 0.117, 0.143, 0.434)
4	300	Nonlinear	0.092 (0.017, 0.138, 0.181, 0.004)	0.299 (0.122, 0.417, 0.438, 0.179)	0.369 (0.066, 0.269, 0.577, 0.500)	0.282 (0.013, 0.078, 0.519, 0.463)	0.210 (0.085, 0.060, 0.103, 0.390)

each method; the median Somers's D_{XY} stayed about the same, but the variability of the distributions decreased. With a sample size of 50, there did not seem to be a clear best method among Ordinal GMIFS, and P/O Boosting, and both Ordinal FANS approaches. However, when the sample size was greater than 50, Ordinal GMIFS and P/O Boosting had slightly superior performance to both Ordinal FANS approaches. Ordinal GMIFS achieved the best performance when the sample size was largest ($n = 300$).

When the number of classes increased from $K = 3$ to $K = 4$, the predictive performance of each method deteriorated, but the difference between the methods' performance became more clear. Furthermore, when the number of classes was $K = 4$, sample size improved the median test set Somers' D_{XY} and reduced the variability of the distribution for all methods. When the sample size was $n = 50$, Ordinal GMIFS and P/O Boosting performed similarly and better than the other methods. However, the gap between the two methods increased with the sample size – Ordinal GMIFS performed best for all sample sizes greater than 50.

True nonlinear decision boundary When the simulated decision boundary was nonlinear, weighted k-nearest neighbors again performed the worst in all cases. With $K = 3$ classes in the outcome, sample size played a large role in the performance of both Ordinal FANS approaches. Aside from k-nearest neighbors, all methods performed approximately equivalently with a sample size of $n = 50$. However, when the sample size increased to $n = 100$, the Ordinal FANS methods improved dramatically. From $n = 100$ to $n = 300$, Ordinal FANS 1 (binary aggregation approach) clearly performed the best among all methods while Ordinal FANS 2 (boosting approach) performed second best. Ordinal GMIFS and P/O Boosting performed similarly behind the Ordinal FANS methods.

The results did not change drastically when the number of classes in the ordinal outcome increased from $K = 3$ to $K = 4$. The estimates of Somers' D_{XY} exhibited high variance for all methods when the sample size was $n = 50$, and all methods performed similarly. The performance of all methods improved as the sample size increased, but the rates of

improvement differed. When the sample size increased to 100, a hierarchy of methods was established. The top performer was Ordinal FANS 1, followed by Ordinal FANS 2, then Ordinal GMIFS, P/O Boosting, and weighted k-nearest neighbors in that order. When $n = 200$, Ordinal FANS 1 remained the best performing method, followed by Ordinal FANS 2 and Ordinal GMIFS, which performed approximately equivalently. P/O Boosting and weighted k-nearest neighbors were the bottom two methods. Finally, with a sample size of 300, Ordinal FANS 1 was again the best method, but Ordinal GMIFS was the second best, followed by Ordinal FANS 2, P/O Boosting, and weighted k-nearest neighbors.

Summary of predictive performance comparison The best method in terms of predictive performance depended on the shape of the underlying decision boundary. When the true decision boundary is linear, Ordinal GMIFS seemed to be the preferable method, but when the true decision boundary was nonlinear, Ordinal FANS 1 (the binary aggregation approach) performed best. This is somewhat expected because Ordinal FANS fits a nonlinear decision boundary in the original feature space, while Ordinal GMIFS fits a linear decision boundary. However, it is worth noting that Ordinal FANS outperformed the other flexible model, k-nearest neighbors, and Ordinal GMIFS outperformed the other method that fits a linear decision boundary, namely P/O Boosting. Furthermore, the same conclusions were reached when we examined the misclassification rates in Table 5.

3.4.1.2 Feature selection

The weighted k-nearest neighbors method does not explicitly perform feature selection, so we only measured sensitivity (Figure 20) and specificity (Figure 21) for models fit by Ordinal GMIFS, P/O Boosting, and both Ordinal FANS approaches.

True linear decision boundary When there were $K = 3$ classes in the ordinal outcome and the ture decision boundary was linear, there was a clear hierarchy. Both Ordinal FANS approaches outperformed Ordinal GMIFS and P/O Boosting for all sample sizes examined

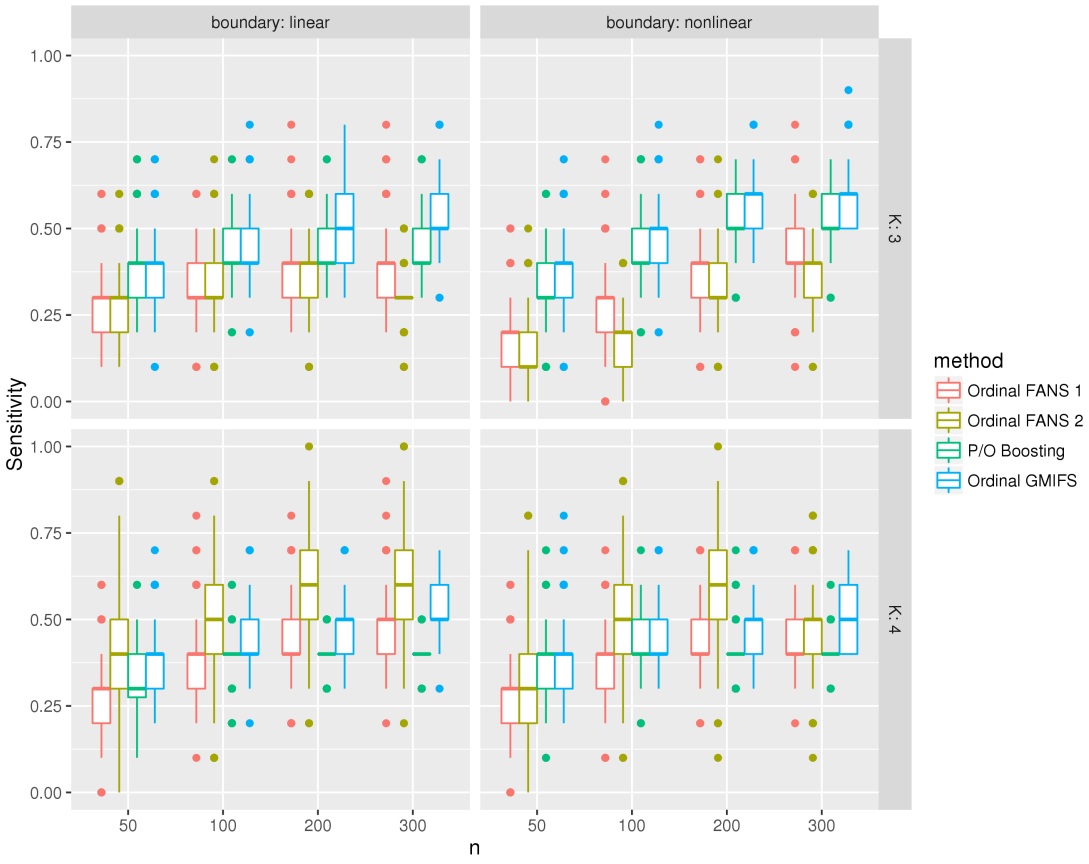

Fig. 20. Simulation results: Distribution of sensitivity for varying K, n, and decision boundaries.

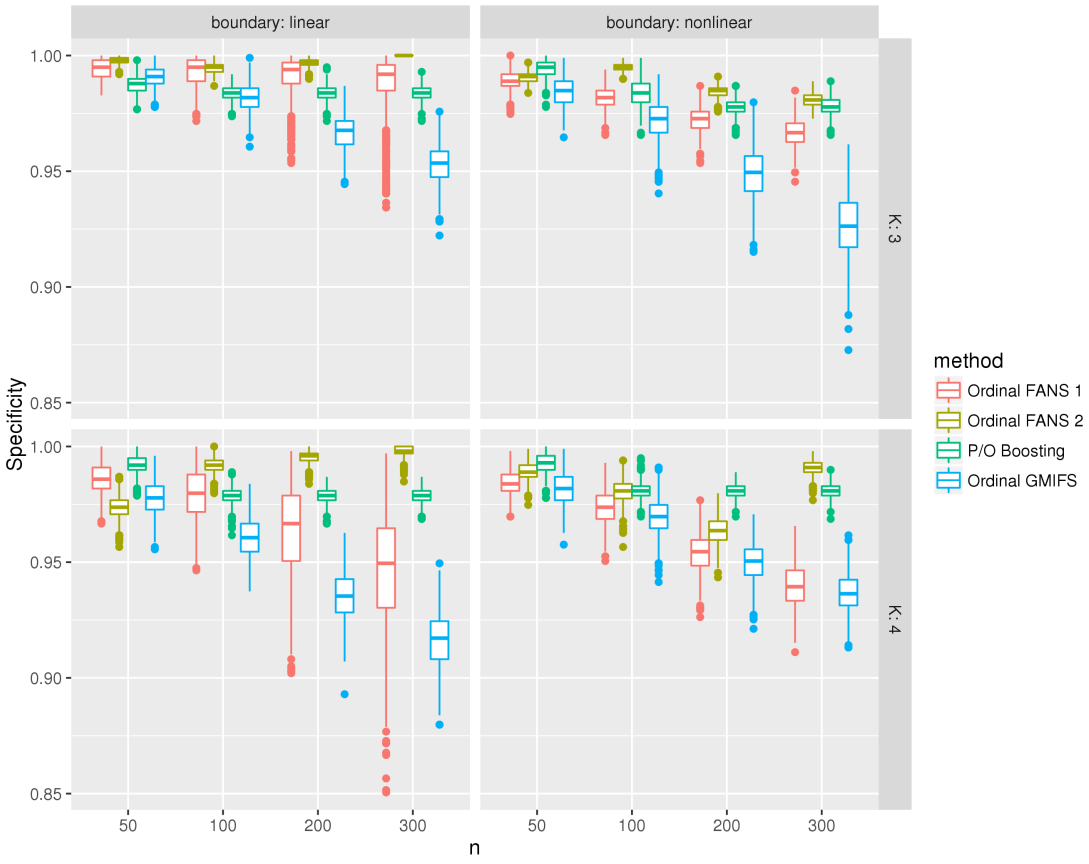

Fig. 21. Simulation results: Distribution of specificity for varying K, n, and decision boundaries.

($n \in \{50, 100, 200, 300\}$) in terms of specificity, but the opposite was true for sensitivity, in which case Ordinal GMIFS was the best method. This suggests that with $K = 3$ classes, models fit using the Ordinal FANS approaches are smaller (i.e. more parsimonious) than models fit using ordinal GMIFS or P/O Boosting, but consequently they miss more important features. Between the two Ordinal FANS approaches, the binary aggregation approach seemed to be slightly more sensitive but less specific.

With $K = 4$ classes in the outcome, Ordinal FANS, approach 2 (boosting) was clearly the most sensitive and specific method when the true decision boundary was linear. The one exception occured when the sample size was 50, in which case it was the most sensitive but not the most specific. P/O Boosting performed best when the sample size was $n = 50$, but Ordinal FANS 2 performed best for all other sample sizes ($n \in \{100, 200, 300)$.

True nonlinear decision boundary When the simulated decision boundary was nonlinear and there were $K = 3$ ordinal classes, there seemed to be a dichotomy in the methods' sensitivity performance – Ordinal GMIFS and P/O Boosting performed similarly and better than Ordinal FANS 1 and Ordinal FANS 2, which performed similarly. However, aside from when the sample size was $n = 50$, in which case all methods performed similarly, Ordinal FANS 2 achieved the highest specificity.

When the number of classes in the outcome increased to 4, the best performing method was dependent on the sample size. P/O Boosting achieved the highest median specificity for sample sizes $n = 50$ and $n = 200$ while Ordinal FANS 2 obtained the highest median specificity when $n = 300$. The two methods performed best and approximately equivalently when $n = 100$. In terms of sensitivity, Ordinal FANS 2 performed best when the sample size was $n = 100$ and $n = 200$. There was no clear best method when $n = 50$ and $n = 300$.

3.4.2 Data analyses

The ranking of methods in terms of predictive performance was dependent on the dataset, which agrees with the simulation study as well as conventional wisdom – no single

Table 6. 10-fold cross-validation estimates of Somers' D_{XY}, misclassification rate, and class-specific misclassification rates for the classification of normal ($n = 10$), HSIL ($n = 7$), and cervical carcinoma ($n = 21$) samples from GSE7803.

Method	Somers' D_{XY}	Misclassification Rate (Class-Specific)
Ordinal FANS 1	0.876	0.158 (0.000, 0.444, 0.095)
Ordinal FANS 2	0.874	0.105 (0.000, 0.200, 0.125)
P/O Boosting	0.876	0.105 (0.000, 0.286, 0.095)
Ordinal GMIFS	0.897	0.079 (0.000, 0.167, 0.091)
Weighted k-Nearest Neighbors	0.939	0.132 (0.000, 0.417, 0.000)

method performs best in all cases.

The results of the analysis of GSE7803 are presented in Table 6. The best method in terms of Somers' D_{XY} was Weighted k-Nearest Neighbors. However, if we instead define predictive performance in terms of the misclassification rate, Ordinal GMIFS was best. Interestingly, all methods correctly classified all normal cervical samples, and the misclassification rate for cervical carcinoma samples was fairly low among all methods as well. Meanwhile, the misclassification rate for HSIL samples was highest among the three ordinal classes for all fitted models.

The results of the analysis of GSE3189 are presented in Table 7. P/O Boosting achieved the highest 10-fold CV estimate of Somers' D_{XY} as well as the lowest misclassification rate. Ordinal FANS approaches 1 and 2 came in third and second, respectively in terms of Somers' D_{XY}, and the Ordinal FANS approaches and Ordinal GMIFS were tied for second in terms of misclassification rate. As with GSE7803, several methods had no misclassifications among the normal skin samples and the highest misclassification rates for the second of the three ordinal class (in this case, nevus skin samples). In the analysis of GSE3189, the methods that did not misclassify any normal skin samples also did not misclassify any melanoma samples. However, in contrast to the analysis of GSE7803, the misclassification rate was highest for normal samples for two methods in this analysis.

Table 8 shows the results of the analysis of GSE18081. The method with the best predictive performance in terms of Somers' D_{XY} was Ordinal FANS 2, followed closely by Ordinal

Table 7. 10-fold cross-validation estimates of Somers' D_{XY}, misclassification rate, and class-specific misclassification rates for the classification of normal ($n = 7$), nevus ($n = 18$), and melanoma ($n = 45$) samples from GSE3189.

Method	Somers' D_{XY}	Misclassification Rate (Class-Specific)
Ordinal FANS 1	0.936	0.057 (0.125, 0.111, 0.023)
Ordinal FANS 2	0.944	0.057 (0.000, 0.182, 0.000)
P/O Boosting	0.958	0.043 (0.000, 0.143, 0.000)
Ordinal GMIFS	0.929	0.057 (0.000, 0.182, 0.000)
Weighted k-Nearest Neighbors	0.888	0.071 (0.222, 0.000, 0.063)

Table 8. 10-fold cross-validation estimates of Somers' D_{XY}, misclassification rate, and class-specific misclassification rates for the classification of normal ($n = 20$), HCV–cirrhotic ($n = 16$), and HCV-HCC ($n = 20$) liver tissue samples from GSE18081.

Method	Somers' D_{XY}	Misclassification Rate (Class-Specific)
Ordinal FANS 1	0.853	0.125 (0.048, 0.273, 0.000)
Ordinal FANS 2	0.888	0.125 (0.000, 0.263, 0.118)
P/O Boosting	0.842	0.179 (0.000, 0.375, 0.083)
Ordinal GMIFS	0.826	0.179 (0.000, 0.375, 0.077)
Weighted k-Nearest Neighbors	0.804	0.143 (0.091, 0.222, 0.125)

FANS 1. The two Ordinal FANS approaches also achieved the lowest misclassification rates. Among the five methods, most of the misclassifications occurred among the HCV-cirrhotic tissues. Several methods (Ordinal FANS 2, P/O Boosting, Ordinal GMIFS) correctly classified all normal samples, while Ordinal FANS 1 correctly classified all HCV-HCC samples.

3.5 Summary

The "no free lunch" theorem states that one method will not perform best for all problems [51]. We have shown that certainly holds true in the case of ordinal response prediction in high-dimensional settings. The simulation study revealed that, overall, the first approach we took to extend FANS to the ordinal response setting performs best when the true decision boundary was nonlinear. However, when the true decision boundary was linear, Ordinal GMIFS seemed to perform best.

With respect to feature selection, for most sample sizes examined, the second Ordinal FANS approach was more sensitive than the other methods when there were $K = 4$ classes

in the outcome. Ordinal GMIFS was the most sensitive when there were $K = 3$ ordinal classes. Furthermore, the second Ordinal FANS approach was also the most specific method in most of the settings we examined. P/O Boosting was slightly more specific in just four of the sixteen settings. Thus, overall, the second Ordinal FANS method seemed to perform feature selection best out of all methods in the comparison.

In the data analyses, each method achieved the best predictive performance in at least one analysis if we considered both Somers' D_{XY} and the misclassification rate. When we examined the class-specific misclassification rates, it became apparent that the methods make different errors. That is, on a given dataset, some methods had the most trouble classifying observations from the first class, while others had the most trouble classifying observations from the second class.

Based on this study, we would recommend applying a suite of predictive models to a given problem and choosing the one that results in the smallest error, estimated using a technique such as bootstrapping or cross-validation.

CHAPTER 4

DISCRETE SURVIVAL TIME ANALYSIS IN HIGH-DIMENSIONAL SETTINGS

4.1 Introduction

Survival analysis involves modeling the time until the occurrence of an event. In biomedical research, common events that are modeled include time to death and disease relapse. For example, a researcher may be interested in determining risk factors that are associated with a shorter remission period for cancer patients. In another case, the goal may be to predict survival times of patients based on a set of independent variables. Here, emphasis is placed on the predictions as opposed to the quantitative relationships between the independent variables and the survival time outcome.

The scale on which survival times are measured (and reported) influence the type of analysis that is performed. A continuous time measurement, calculated using a precise date and time of relapse, provides more information than a discrete time measurement such as the physician visit at which the relapse was detected. However, a continuous time is often unattainable. For instance, patients treated for cutaneous malignant melanoma will typically be seen at follow-up visits every three to six months for the first three years and every six months to a year thereafter [52]. Further, whole body scans may be used for those who were treated for metastases [52]. In this case, if a scan determines the patient has relapsed, the time of relapse may be recorded as the date of their physician visit at which the scan occurred (e.g. follow-up visit 3). All that is known is that the relapse occurred sometime in the interval between visits. Also, researchers may categorize a continuous measurement because they find the discrete measurement to be more interpretable (although this is inadvisable from a statistical perspective). For example, researchers modeling the survival times of glioblastoma

patients reported the subjects as short-term, intermediate, or long-term survivors as opposed to reporting their precise times until death [53]. Furthermore, when de-identifying protected health information, the U.S. Department of Health and Human Services, through the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule, has explicitly prohibited the disclosure of dates more specific than the year of the event[54]. Thus, according to HIPAA, dates of death, relapse, etc., which are used in the calculation of survival times in de-identified datasets, must be reported as a year without a day or month. As a result, survival times would be considered a discrete measurement.

An accurate survival time prediction can have important implications for a patient because choice of treatment is often based on severity of disease [55]. For instance, a patient may be willing to undergo a cancer treatment with severe side effects if she knows she will live long enough to experience the potential benefits. In contrast, a patient given only a short time to live may opt for hospice care. Additionally, an accurate prediction can help the patient and her family prepare for the future. These plans will likely be drastically different if the patient is predicted to live two months versus two years.

Numerous studies have investigated the accuracy of physicians' subjective prognostic abilities, with a unifying theme being that the predictions have poor accuracy [55]. However, when compared to physicians' predictions of survival times of non-small-cell lung cancer patients, a fitted proportional hazards model performed no better [55]. In order to assess the accuracy of continuous survival time predictions, one needs a suitable loss function to compare the prediction with the observed outcome. For survival time predictions, there is no simple choice of function because there are many consequences associated with a poor prediction [55]. For instance, a severe underestimate of a patient's survival time may result in that patient not receiving the treatment that would have extended her life. A drastic overestimate could cause a patient to receive a treatment that reduced the quality of the short time he had left while putting a financial and emotional burden on his family. After an assessment of numerous loss functions and consulting with physicians, Henderson et al.

concluded that the most appropriate loss function for a continuous survival time model considers only whether the prediction is in "serious error" [55]. The loss function was developed by Parkes [56], and assigns no cost if the prediction is within a multiplicative factor of two of the observed outcome and assigns a cost of 1 otherwise [56, 55]. Thus, a predicted survival time of 4 months would be considered reasonable if the patient survived between 2 and 8 months. Essentially, Henderson et al. argue that because standard continuous survival time models have demonstrated poor prediction accuracy, we should consider only what is clinically meaningful and acceptable: a prediction that falls within a certain margin of error of the observed survival time. This provides motivation for modeling grouped survival times.

In the survival analysis literature, several terms are used to describe methods concerned with analyzing event times not measured on a continuous scale. These include discrete time survival analysis, survival analysis of interval-censored data, and grouped-time survival analysis. Some researchers make a distinction between these terms, while others treat them synonymously. Generally, if each event is known to have occurred between two time points, the data are interval-censored [57]. Interval-censored data may be analyzed in several different ways. Cox extended his partial likelihood approach for continuous event times to the case in which ties occur [58, 59]. Ties are assumed to have occurred as a result of the way the event times were recorded (i.e. on an interval). For a distinct event interval, let r represent the number of subjects who are at risk (those who have not yet experienced the event and have not been censored) just prior to that interval, and let d equal the number of subjects who experienced the event during the interval. Cox's partial likelihood requires the summation over all possible subsets of size d of the r at risk [60]. Thus, the likelihood can be computationally prohibitive to calculate if the number of ties is large. Other methods proposed by Breslow and Efron are not as computationally intensive, but they offer rough approximations of the true likelihood [60, 61]. Further, Breslow's likelihood does not acknowledge the grouped nature of the event times and thus would give inconsistent estimates in the presence of grouped times [60]. The distinctions between discrete time survival

analysis, survival analysis of interval-censored data, and grouped-time survival analysis are typically over whether the intervals overlap and if so, the extent of the overlap. Interval-censored data analysis and grouped-time survival analysis are often used interchangeably, but grouped-time survival data arise when all observed intervals either completely overlap or have no overlaps. Further, discrete time survival analysis is used when there are only a small number of distinct, non-overlapping intervals [62]. For example, in studies that involve follow-up periods, such as clinical trials, many ties typically occur: d_1 patients may have experienced the event between the end of the study and follow-up visit 1, d_2 patients may have experienced the event between follow-up visit 1 and follow-up visit 2, etc. In these situations, it is more natural and generally preferred to treat the interval-censored event times as discrete times although the latent variable is continuous [62].

Additionally, when dealing with a time to event outcome, complications to the analysis arise when either a subject has not experienced the event when the study ends or a subject drops out of the study before its conclusion. Outcomes of this nature are called *censored*. The time to event for censored subjects is unknown, but discarding their data would result in a loss of information and a systematic bias. For censored observations, we know that the subject did not experience the event before they were censored. Thus, a lower bound can be placed on their outcome measurement, and this lower bound can be incorporated into the model.

4.1.1 Motivating example: extended phase of the AML DREAM Challenge

Acute myeloid leukemia (AML) is a cancer characterized by the proliferation of immature myeloid cells in the bone marrow and commonly results in hematopoietic insufficiency[63, 64]. Cancerous (i.e. leukemia) cells may build up in the bone marrow and blood, reducing the room for healthy white blood cells, red blood cells, and platelets. This causes infection, anemia, and/or easy bleeding. Furthermore, the leukemia cells can spread to the brain and spinal cord, skin, and gums [64]. There were an estimated 20,830 new cases of AML and approximately 10,460 deaths resulting from the disease in 2015. Furthermore,

only about 25.9% of patients survive 5 or more years after being diagnosed. The median age at diagnosis is 67 and the median age at death is 72 [64]. There are known risk factors for AML, such as age and cytogenetics[65], but discovering additional clinical, genomic, and/or proteomic correlates will help to predict patients who are at an increased risk of relapsing. The Extended Phase of the AML Dialogue for Reverse Engineering Assessments and Methods (DREAM) Challenge provided a dataset of acute myeloid leukemia patients that included clinical covariates (e.g. age, sex), cytogenetic information, and proteomic data. One of the challenges was to develop a model to predict remission duration, measured in intervals, among patients who achieved a complete response. Remission duration was given as: 52 weeks or less, more than 52 weeks but less than or equal to 104 weeks, or greater than 104 weeks [66].

4.1.2 Definitions

Assume there are n independent subjects ($i = 1, 2, 3, ..., n$) and p predictors per subject, where $p >> n$. The data are often presented as follows:

- Let T_i represent the discrete survival time response variable. Here, $T_i = \min(Y_i, C_i)$, where Y_i is the time of event for subject i and C_i is the time at which subject i was censored. Generally, we do not observe both Y_i and C_i; we only observe the minimum of the two times. Now, suppose there are K distinct times ($j = 1, 2, ..., K$) at which at least one subject either experienced the event or was censored, $\tilde{t}_1 < \tilde{t}_2 < ... < \tilde{t}_K$. Then each $T_i \in \{\tilde{t}_1, \tilde{t}_2, ..., \tilde{t}_K\}$.

- An n x 1 vector, $\boldsymbol{\delta}$, is observed where $\delta_i = I(Y_i < C_i)$. That is, $\delta_i = 1$ if an event time was observed for subject i, and $\delta_i = 0$ if subject i was censored.

- A p x 1 vector of covariates, $\boldsymbol{x_i}$, is observed for each subject.

To facilitate the formation of the likelihood, we define an n x K matrix for the event times as follows:

$$y_{ij} = \begin{cases} 1 & \text{if } y_i = \tilde{t}_j \\ 0 & \text{otherwise} \end{cases}$$

4.1.3 Low-dimensional discrete time survival analysis

Traditionally, in cases where $p < n$, discrete survival times are modeled using logistic regression [67, 68]. However, the data needs to be restructured before doing so. For each individual, a new observation is created for each time point until the subject experiences the event. For each of these time points, the outcome is coded as 1 if the subject experienced the event during that period and 0 otherwise. For a given individual, the same covariate values are used for each observation (assuming the covariates are not time dependent) with the addition of a dummy-coded variable for each of the time points (minus one). For example, if a subject experienced the event during year 4 in a study lasting 5 years, four observations would be created with the same covariate values along with four dummy variables for years 1 through 4. The outcomes for these four observations would be a series of three zeros followed by a one. A subject who did not experience the event during the study would have five observations, and the outcome values would be a series of five zeros.

Although it appears the observations for a given subject in the expanded dataset are assumed to be independent, that assumption is not explicitly made. Instead, the binomial likelihood for the expanded dataset is equivalent to the likelihood of the discrete-time survival model obtained before the expansion. Thus, the ability to obtain valid maximum likelihood estimates by treating all observations in the expanded dataset as independent is 'merely an incidental convenience.'[67] We demonstrate this in the following paragraph.

The likelihood of the data can be expressed as [61, 67],

$$\mathrm{L} = \prod_{i=1}^{n} P(Y_i = t_i)^{\delta_i} P(Y_i > t_i)^{1-\delta_i}$$

Now, define the discrete hazard rate, $\pi_{ij} = \pi_j(\boldsymbol{x_i}) = P(Y_i = \tilde{t}_j | Y_i \geq \tilde{t}_j, \boldsymbol{x_i})$, which is interpreted as the probability that a subject experiences the event at time $\tilde{t}_j$ given that they have not already experienced the event. Applying properties of conditional probabilities, we have,

$$P(Y_i = t_i) = \pi_{it_i} \prod_{j=1}^{t_i - 1} (1 - \pi_{ij})$$

$$P(Y_i > t_i) = \prod_{j=1}^{t_i} (1 - \pi_{ij})$$

where $t_i \in \{\tilde{t}_1, \tilde{t}_2, ..., \tilde{t}_K\}$. Plugging these values into the likelihood results in,

$$\mathrm{L} = \prod_{i=1}^{n} \left[\pi_{it_i} \prod_{j=1}^{t_i - 1} (1 - \pi_{ij}) \right]^{\delta_i} \left[\prod_{j=1}^{t_i} (1 - \pi_{ij}) \right]^{1 - \delta_i}$$

The log-likelihood results from taking the logarithm and simplifying:

$$\log \mathrm{L} = \sum_{i=1}^{n} \delta_i \log \left(\frac{\pi_{it_i}}{1 - \pi_{it_i}} \right) + \sum_{i=1}^{n} \sum_{j=1}^{t_i} \log(1 - \pi_{ij})$$

Using y_{ij} as defined previously, the log-likelihood can be re-expressed as [67]

$$\log \mathrm{L} = \sum_{i=1}^{n} \sum_{j=1}^{t_i} \left[y_{ij} \log \left(\frac{\pi_{ij}}{1 - \pi_{ij}} \right) + \log(1 - \pi_{ij}) \right].$$

Notice that this log-likelihood is equivalent to the log-likelihood utilized in logistic regression [67].

Now, re-expressing the log-likelihood as,

$$\log \mathrm{L} = \sum_{i=1}^{n} \sum_{j=1}^{t_i} \left[y_{ij} \log(\pi_{ij}) + (1 - y_{ij}) \log(1 - \pi_{ij}) \right], \tag{4.1}$$

demonstrates that this formulation assumes that censored observations are observed at c_i but not at $c_i + 1$. In other words, the censoring occurs at the end of the interval in which the censoring was recorded [67]. For instance, let $K = 3$ years. If $c_i = 2$, all we know is that observation i was censored between the start of the second year and the end of the second year. The above log-likelihood assumes that observation i was censored at the end of the

second year, implying that $y_i > 2$. However, if the observation was actually censored soon after the start of the second year, bias is added to the model. We will discuss other ways of incorporating the censored observations in section 2.2.

4.2 Forward continuation ratio model

4.2.1 Penalized and unpenalized predictors

In high-throughput experiments, there are often two sets of predictors: a high-dimensional set of genomic predictors (e.g. gene expression data) and a smaller dimensional set of demographic and clinical predictors (e.g. sex, age, platelet count). Given the number of genomic predictors is generally at least in the tens of thousands, in the absence of applying some aggressive data reduction or filtering strategy, a penalized (also called regularized) procedure is required to estimate the model parameters. One of the most frequently used methods utilizes the L1 or LASSO penalty, which constrains the sum of the absolute coefficients to be less than some tuning parameter, λ. In a model with p predictors, the constraint is given by $\sum_{q=1}^{p} |\beta_q| \leq \lambda$. The coefficients of the predictors that are deemed unimportant by the algorithm will be shrunk to zero, so parameter estimation and feature selection are performed simultaneously. However, if certain clinical variables are known to be associated with, or predictive of, the outcome, then they should be forced into the model. That is, the clinical predictors should not be estimated using penalization techniques. The ability to model both penalized and unpenalized predictors is a crucial aspect of our method for modeling high-dimensional data. One reason is that the National Institutes of Health (NIH) released a statement suggesting that all NIH-funded projects going forward must either consider sex as a biological variable or provide convincing justification from the scientific literature or preliminary data for studying only one sex [69]. Going forward, we refer to the high-dimensional set of predictors whose coefficients we estimate using penalization as the penalized predictors, denoted by $\boldsymbol{x}$, and the predictors we force into the model as the unpenalized predictors or unpenalized subset, denoted by $\boldsymbol{z}$.

4.2.2 Likelihood

We developed a method for modeling high-dimensional discrete survival time data from an ordinal modeling perspective. Our method fits the high-dimensional data to a discrete survival time outcome using a forward continuation ratio (FCR) model with a complementary log-log (cloglog) link function given by [70],

$$\log[-\log(1-\pi_{ij})] = \alpha_j + \boldsymbol{x_i}\boldsymbol{\beta} + \boldsymbol{z_i}\boldsymbol{\theta}, \quad j = 1, ..., K-1 \tag{4.2}$$

where α_j represents the intercept, or threshold, for the jth distinct time, $\tilde{t}_j$, and $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are the coefficients for the penalized and unpenalized predictors, respectively [71]. The cloglog link function is appropriate if it is reasonable to assume that the data were generated by a continuous-time proportional hazards model [60]. Furthermore, π_{ij} is the discrete hazard rate as defined previously.

No Censoring Occurs Sometimes in discrete survival time datasets, all subjects experienced the event [53]. Thus, we first developed a model that does not incorporate censoring information [70]. We define the likelihood as a product of n conditionally independent Binomial random variables [72], where $(1-\pi_{ij})$ is the conditional complement of π_{ij} equal to $P(Y_i > \tilde{t}_j | Y_i \geq \tilde{t}_j, \boldsymbol{x}_i, \boldsymbol{z}_i)$:

$$L = \prod_{i=1}^{n} \prod_{j=1}^{K-1} \pi_{ij}^{y_{ij}} (1-\pi_{ij})^{\sum_{k=j}^{K} y_{ik} - y_{ij}}. \tag{4.3}$$

If interest lies in predicting a future observation, we can recursively estimate the probability that a subject experienced the event at a certain time [70]:

$$\begin{aligned} P(Y_i = j|\boldsymbol{x}_i) &= \pi_{ij} * P(Y_i \geq j|\boldsymbol{x}_i, \boldsymbol{z}_i) \\ &= \begin{cases} \pi_{ij} & \text{for } j = 1 \\ \pi_{ij} * \sum_{i=1}^{j-1} P(Y_i = j|\boldsymbol{x}_i, \boldsymbol{z}_i) & \text{for } 1 < j \leq K \end{cases} \end{aligned}$$

Then, we predict the subject will experience the event at the time point with the maximum estimated probability.

Censoring Occurs To account for censoring, we define a response matrix as,

$$t_{ik} = \begin{cases} 1 & \text{if } t_i = \tilde{t}_k \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

Now we can incorporate censored observations by using a modified version of the previously-defined likelihood:

$$L = \prod_{i=1}^{n} \prod_{j=1}^{K} \pi_{ij}^{y_{ij}} (1 - \pi_{ij})^{\sum_{k=j}^{K} t_{ik} - y_{ij}}.$$

There are two modifications from equation 4.3. First, the sum in the second exponent is applied to $\boldsymbol{t}_i = (t_{i1}, ..., t_{iK})$ as opposed to $\boldsymbol{y}_i = (y_{i1}, ..., y_{iK})$. This change is necessary because $y_{ij} = 0$ for $j \in \{1, ..., K\}$ if $\delta_i = 0$. The second modification is that the product over time intervals now goes to K instead of $K - 1$ to account for censoring in the last time period, $\tilde{t}_K$.

The corresponding log-likelihood is given by

$$\log \text{ L} = \sum_{i=1}^{n} \sum_{j=1}^{K} \left[y_{ij} \log(\pi_{ij}) + \left(\sum_{k=j}^{K} t_{ik} - y_{ij} \right) * \log(1 - \pi_{ij}) \right]. \tag{4.5}$$

In our redefined likelihood, t_{ik} can be defined differently depending on our assumption of the timing of the censoring. Recall that in cases when $p < n$, it is often assumed that an observation is censored at the end of the discrete time interval in which the censoring occurred, so the contribution to the likelihood reflects the assumption that $y_i > c_i$ [67]. When the response matrix is defined as it is in (4.4), the log-likelihoods in equations (4.1) and (4.5) are equivalent. In our previous example, we set $K = 3$ years and we assumed the subject was censored during the second time interval, so $c_i = 2$. Using this definition of t_{ik},

the contribution to the likelihood for censored observation i is

$$\log(1 - \pi_{i1}) + \log(1 - \pi_{i2})$$

However, if the censoring occurred towards the beginning of the second year, it may be more appropriate to assume that each censored observation was censored at the end of the first year. In this case the contribution to the likelihood reflects the assumption that $y_i > c_i - 1$. Using this assumption of the timing of censoring, we have

$$t_{ik} = \begin{cases} 1 & \text{if } y_i = \tilde{t}_k \\ 1 & \text{if } c_i = \tilde{t}_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

And so the contribution to the likelihood for censored observation i is

$$\log(1 - \pi_{i1})$$

Finally, we can assume the hazard rate is constant over the time period in which the observation was censored [73]. This method represents a compromise between the previous two assumptions. Here, we define

$$t_{ik} = \begin{cases} 1 & \text{if } y_i = \tilde{t}_k \\ 0.5 & \text{if } c_i = \tilde{t}_k \\ 0.5 & \text{if } c_i = \tilde{t}_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

Using this compromise, the contribution to the likelihood for censored observation i is

$$\log(1 - \pi_{i1}) + 0.5 * \log(1 - \pi_{i2})$$

Now define $\boldsymbol{\pi}_j = (\pi_{1j}, \pi_{2j}, ..., \pi_{nj})^T$. The derivative of the log-likelihood with respect to the q^{th} predictor is given by

$$\frac{\delta \log L}{\delta \beta_q} = \sum_{j=1}^{K} \left[x_q^T * \exp\{-\exp\{\alpha_j + \boldsymbol{x\beta} + \boldsymbol{z\theta}\} + \alpha_j + \boldsymbol{x\beta} + \boldsymbol{z\theta}\} \left[\frac{\boldsymbol{y}_j}{\boldsymbol{\pi}_j} - \frac{\sum_{k=j}^{K} \boldsymbol{t}_k - \boldsymbol{y}_j}{\mathbf{1} - \boldsymbol{\pi}_j} \right] \right] \quad (4.6)$$

When the purpose of a model is strictly to predict a future observation, as opposed to inferring information about the relationship between the predictors and response, researchers will often treat censoring times as event times instead of incorporating a lower bound on these observations into the likelihood [74]. This amounts to setting $y_i = c_i$ for the censored observations and fitting an FCR model using the likelihood in equation 4.3. In this case, the contribution to the likelihood for censored observation i would be

$$\log(1 - \pi_{i1}) + \log(\pi_{i2}).$$

From here on, we will refer to the censoring assumptions by number:

- Assumption 1: $y_i > c_i$
- Assumption 2: $y_i > c_i - 1$
- Assumption 3: Constant hazard rate within the interval in which the censoring occurred.
- Assumption 4: Event times are equal to the censoring times, i.e. $y_i = c_i$.

4.2.3 Model fitting in high-dimensional settings

The Generalized Monotone Incremental Forward Stagewise (GMIFS) method is an algorithm that produces a monotone LASSO solution for loss functions other than squared error. Hastie et al. provided a concrete example using logistic regression [13], and the GMIFS method was subsequently extended by Archer et al. for fitting several different logit link ordinal response models to high-throughput genomic data including the cumulative logit, forward continuation ratio, backward continuation ratio, stereotype logit, and adjacent cat-

egory models [7].

Using the log-likelihood in equation 4.5 and it's derivative in equation 4.6, we extended the GMIFS algorithm for ordinal response modeling using a cloglog link function, which enabled us to estimate a penalized solution to the discrete survival time FCR model in the presence of censoring. We can also include an unpenalized subset of predictors in the model by using the following modified GMIFS algorithm [71]:

1. Enlarge the predictor space as $\tilde{\mathbf{X}} = [\mathbf{X} : -\mathbf{X}]$, where $\mathbf{X}$ represents the standardized predictors.

2. Initialize the α's to their empirical values. For model (4.2), these are initialized as $\alpha_j = \log\left(-\log\left(1 - \frac{\sum_{i=1}^{n} t_{ij}}{\sum_{i=1}^{n}\sum_{k=j}^{K} t_{ik}}\right)\right)$ for $j = 1, ..., p$. For the discrete survival time model with censored data, we initialize an additional threshold as $\alpha_K = \log(-\log(1 - 0.99)) = 1.52718$.

3. For step $s = 0$, initialize the components of $\hat{\boldsymbol{\beta}}^{(s)}$ as $\hat{\beta}_1 = \hat{\beta}_2 = ... = \hat{\beta}_p = \hat{\beta}_{p+1} = ... = \hat{\beta}_{2p} = 0$.

4. Treat $\hat{\boldsymbol{\beta}}^{(s)}$ as fixed and estimate $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ by maximum likelihood.

5. Find $m = \underset{p}{\text{argmin}}$ -$\delta \log L / \delta \beta_p$ at the current estimate $\hat{\boldsymbol{\beta}}^{(s)}$.

6. Update $\hat{\beta}_m^{(s+1)} = \hat{\beta}_m^{(s)} + \epsilon$, where ϵ is a small increment such as 0.001.

7. Repeat steps 4 to 6 until the difference between two successive log-likelihoods is smaller than a pre-specified tolerance, τ.

The resulting model will be sparse, meaning the vast majority of the coefficient estimates will be zero. Thus, only the unpenalized predictors and the penalized predictors deemed important by the algorithm will have an estimated effect on the outcome. As a result, we can assess how well the model predicts a future observation as well as how well the model performed feature selection. The former can be estimated by applying the model to an

independent validation set or by utilizing resampling techniques (e.g. bootstrap resampling, cross-validation) and the latter can be assessed with a simulation study.

4.2.4 Performance measures

We measured the performance of the ordinal response models in the simulation and data analysis with Somer's D, a measure of association between two ordinal variables. Before giving the equation for Somer's D, it is necessary to define concordance and discordance. Let y_a and $\hat{y}_a$ be the observed and the predicted ordinal response variables, respectively, for subject a. Given a pair of subjects, if the subject that has a larger observed value also has the larger predicted value, the pair of subjects is *concordant*. If the subject that has a larger observed value has the smaller predicted value, the pair of subjects is *discordant*. Now, let C represent the number of concordant pairs in a sample and D represent the number of discordant pairs. Define T_y as the number of pairs in which the two subjects had the same observed response value (i.e. they were tied with respect to y) and n as the number of subjects. Somer's D is an asymmetric measure of association, meaning we cannot treat the two variables interchangeably. We used Somer's D_{XY}, which measures how well X serves as a predictor of Y [17]. Here, X represents the predicted value, or $\hat{y}_a$, and Y represents the observed outcome, y_a. The sample version of Somer's D_{XY} is given by:

$$D_{XY} = \frac{C - D}{n(n-1)/2 - T_Y},$$

where the numerator represents the difference in the number of concordant and discordant pairs, and the denominator represents the total number of pairs that are untied on Y. D_{XY} ranges from -1 to 1, where $D_{XY} = -1$ indicates a perfect negative association, $D_{XY} = 1$ indicates a perfect positive association, and $D_{XY} = 0$ indicates no association.

Somer's D has been extended to account for censored observations in survival data [17]. Define the censored sign difference, csign, for two outcomes, y_a and y_b, and their respective

censoring indicators, δ_a and δ_b, as

$$\text{csign}(y_a, \delta_a, y_b, \delta_b) = \begin{cases} 1 & \text{if } y_a > y_b \text{ and } \delta_a \leq 1 = \delta_b \\ -1 & \text{if } y_a < y_b \text{ and } \delta_a = 1 \geq \delta_b \\ 0 & \text{otherwise} \end{cases}$$

For the predicted values, we will use

$$\text{sign}(\hat{y}_a, \hat{y}_b) = \begin{cases} 1 & \text{if } \hat{y}_a > \hat{y}_b \\ -1 & \text{if } \hat{y}_a < \hat{y}_b \\ 0 & \text{otherwise} \end{cases}$$

Given the set $\{y_a, \delta_a, y_b, \delta_b, \hat{y}_a, \hat{y}_b\}$, the product $\text{csign}(y_a, \delta_a, y_b, \delta_b) * \text{sign}(\hat{y}_a, \hat{y}_b)$ is calculated and the pair of subjects is declared to be concordant if the product is 1, discordant if the product is -1, and "tied" if the product is 0. This product is calculated for all $\binom{n}{2}$ pairs of subjects and then Somer's D_{XY} can be calculated as usual, where T_Y is the number of times that $\text{csign}(y_a, \delta_a, y_b, \delta_b) = 0$.

4.2.5 Simulation

The aim of the simulation study was two-fold. First, we were interested in determining how well our model predicts event times in an independent dataset. Second, we wanted to determine how well our model performs feature selection. We simulated data from 3 discrete survival time points (e.g. short-, intermediate-, and long-term survival), and we set the number of penalized predictors to $p = 1,000$. Within each time point ($k = 1, 2, \text{and } 3$), 50 multivariate Gaussian observations were generated, where the $\{w, z\}^{th}$ element of the covariance structure was set to $\sigma_{wz} = 0.5^{|w-z|}$ and the variance of each predictor was set to $\sigma_{zz} = 1$. We simulated the first ten predictors to have means equal to their event time and the remaining 990 to have a mean of 0 for each of the three classes, so $\boldsymbol{\mu} = (\boldsymbol{k}_{10}^T, \mathbf{0}_{990}^T)$ for $k = 1, 2, 3$. In order to assess the effect of censoring on performance, we randomly sampled 10%, 20%, and 30% of the observations to be censored. We were also interested

in determining which of the four assumptions is most valid, so we fit models using each assumption. Thus, there were 12 simulation scenarios, and each simulation was run 2,401 times.

Using stratified random sampling, we allocated 2/3 of the data to a training set and 1/3 to a validation set. We fit a discrete survival time model to the training set using the GMIFS algorithm and predicted the event times in the validation set. We assessed the predictive performance of the algorithm with Somer's D_{XY} measure of association between the observed and predicted values. Next, in order to assess how well the model performed feature selection, we fit a discrete survival time model using the combined training and validation set. We calculated the sensitivity, which assesses the ability of the model to correctly identify features that were simulated to have different means across the outcome classes, and we calculated the specificity, which assesses the ability of the model to correctly identify features that were simulated to have equal means across the outcome classes. Furthermore, the model resulting from the convergence of the GMIFS algorithm is often overfit, resulting in poor test set error rates, so we examined the results of the models at the steps that minimized the AIC and BIC.

We used the normal approximation for the confidence interval of a proportion to arrive at the number of simulations. We set the estimated accuracy at 0.5 because it gives the most conservative (i.e. largest) sample size estimate, and we wanted to be within 2% of the true error rate with 95% confidence. Thus, setting the margin of error to 0.02, and using $Z_{1-\alpha} = 1.96$, we concluded that we needed to conduct 2,401 simulations, as shown below.

$$0.02 = 1.96 * \sqrt{\frac{0.5 * 0.5}{n}}$$
$$\Rightarrow n = 0.25 * \left(\frac{1.96}{0.02}\right)^2 = 2,401$$

Table 9. Distribution of T (% censored within interval) for the Extended Phase of the AML DREAM Challenge data.

Interval 1	Interval 2	Interval 3
71 (16.9%)	16 (0%)	44 (68.2%)

4.2.6 Data analysis

4.2.6.1 Exploratory data analysis

The dataset provided for the Extended Phase of the AML DREAM Challenge contained a censoring indicator and $p = 265$ predictor variables (34 clinical and cytogenetic predictors and 231 proteomic predictors) on 187 subjects. However, we only retained the $n = 131$ subjects who achieved a complete response and thus had a nonmissing response variable (remission duration). Of the 131 subjects, 89 (67.9%) relapsed and 42 (32.1%) were censored, but the censored observations were not uniformly distributed across the three intervals (Table 9). The distribution of T and C reflect the difficulties in analyzing this particular dataset. First, there is a noticeable class imbalance, which will likely cause any model to overemphasize the classes with larger sample sizes. Second, a ubiquitous assumption in survival analysis is that the censoring times are independent of the event times. This assumption appears to be invalid in this particular dataset. Nevertheless, these issues are common in biomedical data.

The structure of the chromosomes (i.e. cytogenetics) is a known predictor of risk for AML patients and can be used to categorize a patient into one of three risk groups: favorable, intermediate, or adverse [65]. No subjects in the dataset had a favorable karyotypic feature, so we dichotomized the variable as "adverse" or "other". Next, we checked for any variables with zero variance or "near-zero variance." Near-zero variance predictors are those in which the number of unique values divided by the sample size is small and the ratio of the frequency of the most prevalent predictor value to the frequency of the second most prevalent value is large [74]. After ensuring that no predictors met the criteria, we centered and scaled the

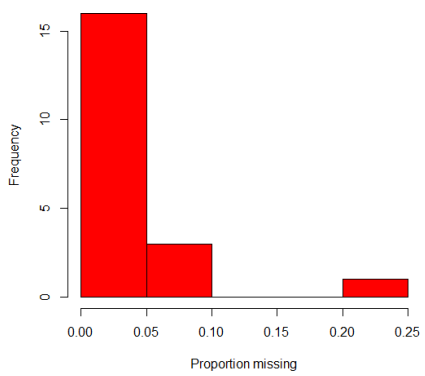

Fig. 22. Proportion missing among predictors with at least one missing value in the AML dataset.

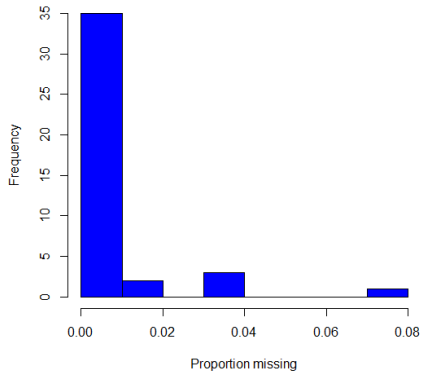

Fig. 23. Proportion missing among samples with at least one missing value in the AML dataset.

Table 10. Percent censored within each interval in the AML dataset, by RAS mutation status.

	Interval 1	Interval 2	Interval 3
Missing RAS status	60.7%	10.7%	28.6%
Not missing RAS status	52.4%	12.6%	35.0%

continuous predictors. Then we examined the data for missing values. Of the 131 subjects, 41 (31.3%) had at least one missing value, and of the 265 predictors, 20 (15.3%) had at least one missing value. Among the predictors with at least one missing value, only one (a variable indicating whether or not a patient had a particular Ras mutation) had greater than 10% of its values missing (Figure 22). We empirically examined that predictor to assess whether the pattern of missingness was related to the outcome (Table 10) and concluded that the pattern of missingness appeared to be random. We also looked at the subjects with at least one missing value (Figure 23) and noticed that the amount of missingness was small. Thus, we imputed the missing values using the K-nearest neighbors technique, which is fairly robust to the tuning parameter, K, and the amount of missing data [75]. We used the `R` function, `knnImputation` in the `DMwR` package with the default value of $K = 10$.

Unpenalized Subset Recall that the unpenalized subset refers to clinical predictors that have a known association with the response. We do not penalize these variables in the fitting process, so they are effectively forced into the model. To choose the predictors to include in the unpenalized subset, we first separately fit each of the 34 clinical variables to the discrete survival time response using logistic regression models, as previously described (Table 11). We combined the results of this analysis with a literature search of known risk factors among AML patients and decided to include 4 predictors in the unpenalized subset: cytogenetic category[65, 76], age at diagnosis[65, 63], Fms-like tyrosine kinase 3 receptor-internal tandem duplication (FLT3/ITD mutation)[77, 78, 79], and hemoglobin count[76]. The remaining clinical and proteomic predictors were included in the penalized subset.

4.2.6.2 Analysis

We used the GMIFS algorithm to fit a FCR model to the AML dataset for each of the four assumptions regarding the censoring information. We evaluated the results of the models at the steps that minimized the AIC and BIC using the leave-one-out (N-fold) cross-validation estimate of Somer's D_{XY} for each of the four models.

4.3 Results

4.3.1 Simulation results

For each of the 2,401 simulations, 2/3 of the data was used to train a model, and 1/3 of the data was set aside to test the fitted model's prediction performance. Figure 24 shows the distributions of Somer's D_{XY}, calculated using the predicted observations in the validation set, for each of the models fit using different assumptions and for varying percentages of censoring. For all models, performance deteriorated as the percentage of censored observations in the data increased. This is to be expected since less information is provided by a censored observation than an observation with an event time. When 10% of the observations were censored, models fit using assumptions 2 and 4 resulted in slightly larger

Table 11. Extended Phase of the AML DREAM Challenge univariate feature selection for the unpenalized subset.

Feature	Unadjusted p-value
Hemoglobin count	0.0005
Whether the patient was found to have a ITD FLT3 mutation	0.0028
Levels of cell surface marker CD13 detected	0.0054
Cytogenetic category	0.0126
Whether the patient was found to have a Ras/STAT mutation	0.0285
Age at time of diagnosis	0.0325
Fibrinogen levels	0.0420
Whether the patient had been diagnosed with a prior cancer	0.0533
Whether the patient was found to have a D835 FLT3 mutation	0.0664
Whether the patient had had prior chemotherapy	0.0768
Levels of cell surface marker CD7 detected	0.0889
Whether the patient was diagnosed with an infection	0.0925
Sex	0.1576
White blood cell count	0.1938
Whether the patient had prior radiation therapy	0.2295
Prior antecedent hematologic disorder	0.2601
Albumin levels	0.2658
Lactate dehydrogenase levels	0.3120
Levels of cell surface marker HLA-DR detected	0.3235
Number of myeloid blast cells (bone marrow)	0.3443
Total number of myeloid blast cells (blood)	0.4051
Number of monocytes measured (blood)	0.4287
Levels of cell surface marker CD33 detected	0.4560
Creatinine levels measured	0.6376
Number of promegakaryocytes measured (blood)	0.6894
Levels of cell surface marker CD34 detected	0.6951
Number of monocytes measured (bone marrow)	0.7156
Bilirubin levels measured	0.7744
Levels of cell surface marker CD19 detected	0.7979
Levels of cell surface marker CD20 detected	0.8334
Number of myeloid blast cells measured (blood)	0.9305
Platelet count	0.9474
Levels of cell surface marker CD10 detected	0.9501
Number of promegakaryocytes measured (bone marrow)	0.9541

values of Somer's D_{XY} than models fit using assumptions 1 and 3. This trend continued as 20% and then 30% of observations were censored. However, the differences in performance became more apparent as the percentage of censored observations increased. When 20% of observations were censored, assumption 1 was worst, followed by assumption 3. Models fit using assumption 4 and assumption 2 performed relatively equivalently. In the simulation with 30% of observations censored, assumption 1 performed poorly and by far the worst, followed by assumption 3. Assumptions 2 and 4 performed best, but assumption 4 was the slightly better of the two. In all three cases with different levels of censoring, assumptions 2 and 4 performed much better than assumptions 1 and 3. Models fit using assumption 4, which underestimates survival times for censored observations by assuming their event times are equal to the times at which the observations were censored, consistently achieved the largest values of Somer's D_{XY}. This result echos Kuhn and Thompson's statement that incorporating censoring times may not be necessary when prediction, as opposed to inference, is the ultimate goal [74]. It appears there is a bias-variance tradeoff, as is the case in most statistical learning problems. Assumption 4 biases results by underestimating survival times for censored observations but reduces the variance by incorporating into the likelihood an event time as opposed to a *bound* on an event time, as the other assumptions do. Furthermore, the selection criteria did not seem to make a difference. That is, the AIC-selected and BIC-selected models performed approximately equivalently.

Aside from developing a useful predictive model, the GMIFS algorithm also performs feature selection. When $p >> n$, this corresponds to selecting a subset of the p features to jointly predict the response. Of the $p = 1,000$ features in the simulations, 10 were simulated to differ between classes, while the remaining 990 were not. Figure 25 shows the distribution of the sensitivity for the AIC-selected and BIC-selected models, separately by assumption. Once again, the models employing assumptions 2 and 4 performed best, assumption 1 performed poorly, and assumption 3 performed slighlty worse than 2 and 4. As the percentage of censoring increased, assumption 4 performed better than assumption 2. This is a surprising

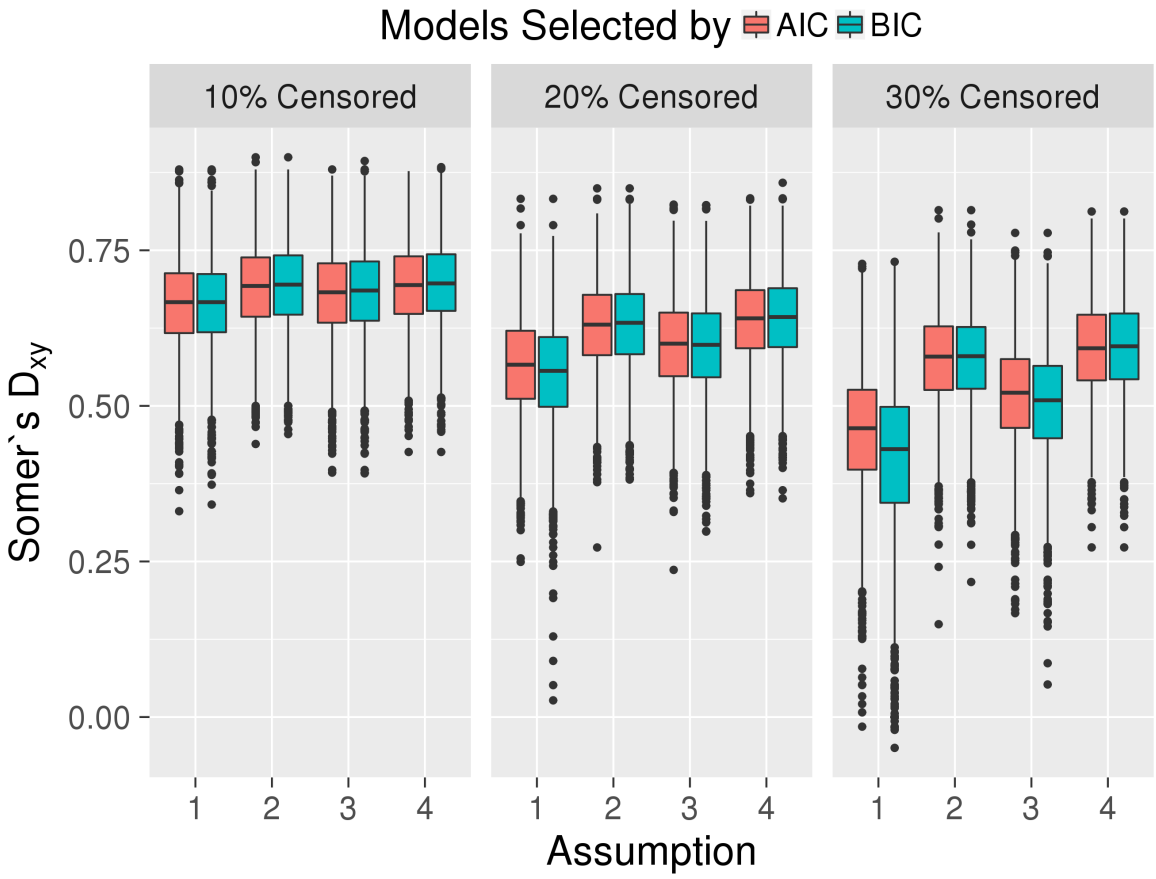

Fig. 24. Validation set estimates of Somer's D_{XY} from the simulation study for models fit using each of the four assumptions and for different proportions of censoring. Results from both the AIC-selected (red) and BIC-selected (blue) models are shown.

result. It has been hypothesized that assumption 4 might be reasonable when prediction is the primary research task [74], but assumption 4 also seems to perform feature selection well. Further, the percentage of censoring does not affect the assumption 4 models because the models in the three cases are the same. That is, since assumption 4 effectively ignores the censoring information, the proportion of censoring does not affect the predictors selected by the models. Increasing the percentage of censored observations slightly negatively affected models fit with assumption 2. Furthermore, the variation of sensitivity estimates is relatively large for all cases. This is a result of there being only 10 features that were simulated to have nonzero coefficients. Thus, the addition or withdrawal of a single predictor from a model has a large impact on the estimated sensitivity.

Figure 26 shows the distribution of the specificity estimates. As with sensitivity, models fit using assumptions 2 and 4 performed best, and assumption 1 performed poorly. Additionally, it seems that using BIC to select the final models resulted in better specificity but slightly worse sensitivity.

4.3.2 Data analysis results

The results from the Extended Phase of the AML DREAM Challenge data analysis are presented in Figure 27. Aside from assumption 2 in which the AIC and BIC performed equivalently, the BIC seemed to have slightly larger Somer's D_{XY} values, which suggests that the AIC-selected models were overfit. The models for assumption 2 achieved the highest estimate of Somer's D_{XY} (Figure 27). However, the AIC-selected and BIC-selected models fit using assumption 3 were well within one standard error of the models fit using assumption 2. Thus, it appears that there were two levels of performance. Assumptions 2 and 3 performed best while models fit using assumptions 1 and 4 performed equivalently to one another but worse than 2 and 3.

In addition to prediction accuracy, we were also interested in determining which variables the models selected among the 261 penalized predictors. Thus, we fit a model utilizing the entire

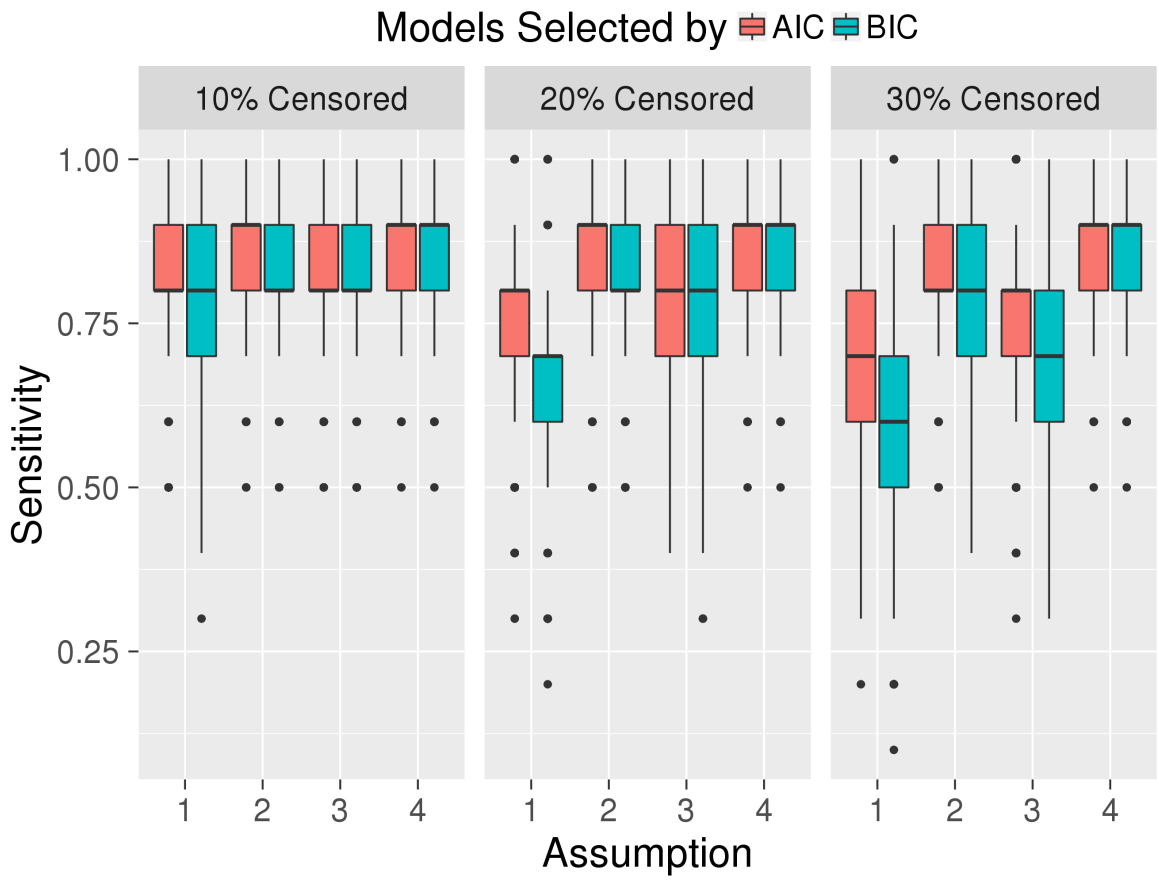

Fig. 25. The distribution of sensitivity estimates from the simulation study for models fit using each of the four assumptions and for different proportions of censoring. Results from both the AIC-selected (red) and BIC-selected (blue) models are shown.

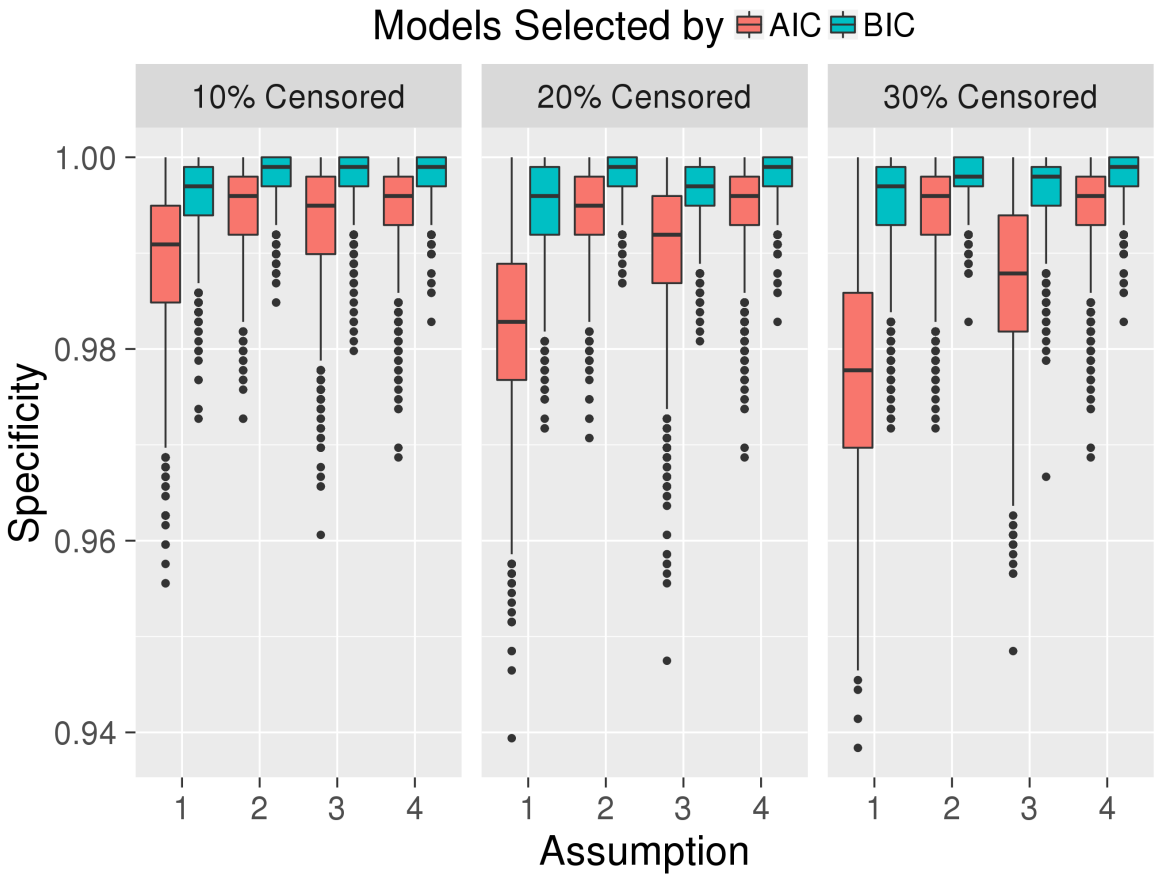

Fig. 26. The distribution of specificity estimates from the simulation study for models fit using each of the four assumptions and for different proportions of censoring. Results from both the AIC-selected (red) and BIC-selected (blue) models are shown.

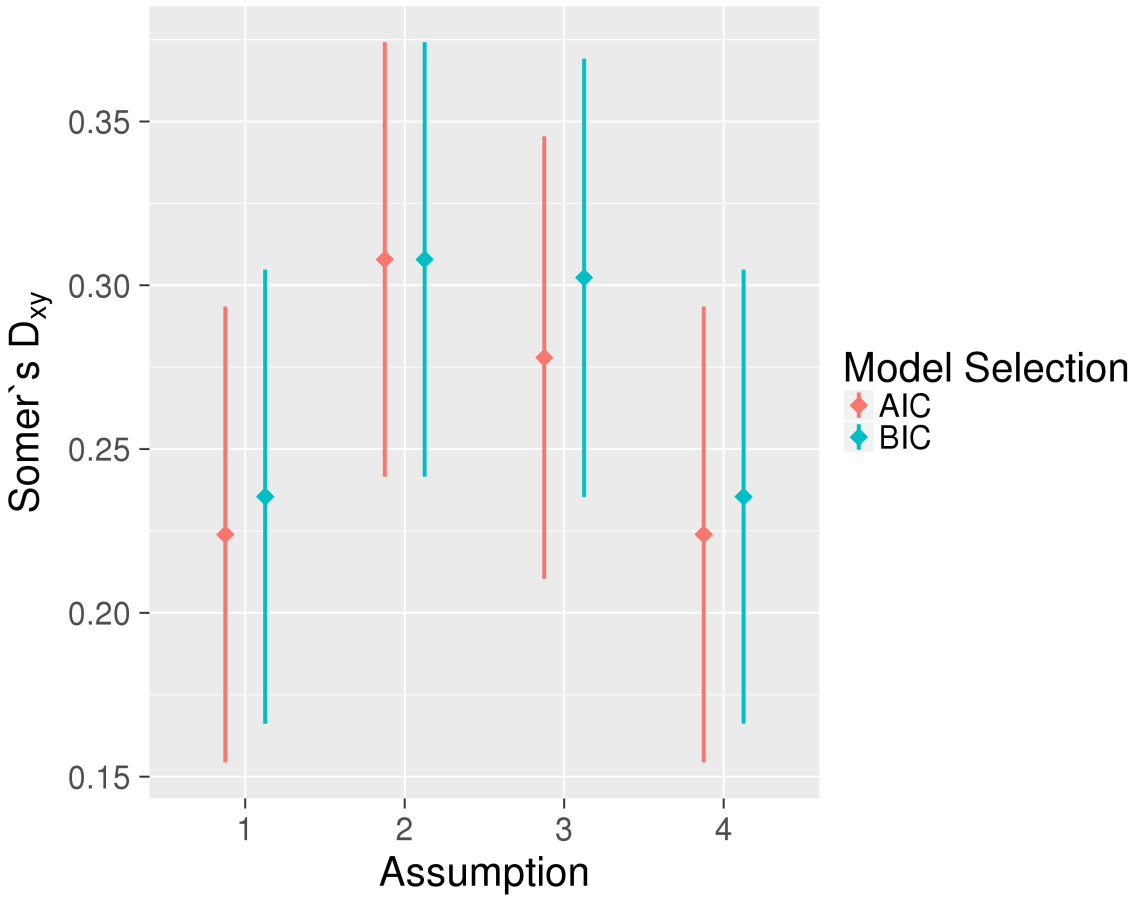

Fig. 27. Leave-one-out cross-validation estimates of Somer's D_{XY} $\pm$ one standard error for models fit using the AML data and for each of the four assumptions. Results from both the AIC-selected (red) and BIC-selected (blue) models are shown.

Table 12. Number of nonzero coefficient estimates among the 261 penalized predictors when using the entire AML dataset. The GMIFS iteration of the selected model is shown in parentheses.

Selection Criterion	Assumption 1	Assumption 2	Assumption 3	Assumption 4
AIC	4 (207)	2 (115)	3 (118)	26 (2,371)
BIC	4 (207)	2 (115)	1 (29)	1 (41)

dataset for each assumption, as opposed to N-fold CV models, in which one observation is left out each time a model is fit. The BIC-selected models, which seemed to achieve slightly better prediction accuracy as measured by N-fold CV (Figure 27), were more parsimonious than the AIC-selected models for assumptions 3 and 4 (Table 12). AIC and BIC selected the same models for assumptions 1 and 2. We compared the features with nonzero coefficient estimates in each AIC-selected and BIC-selected model. Only one feature, MCL1, had a nonzero coefficient estimate in all models. MCL1, a member of the Bcl-2 family, is the anti-apoptotic protein encoded by the MCL1 (myeloid cell leukemia 1) gene. Although the gene doesn't play a direct role in cell proliferation and differentiation, MCL1 and other genes in the Bcl-2 family can either permit or prevent these programs from proceeding [80]. The longest gene product (isoform) inhibits apoptosis, but alternatively spliced shorter isoforms promote apoptosis. Furthermore, drug resistance in AML has been linked to high levels of MCL1 [81]. Glaser et al. showed that removal of MCL1 could cure mice with AML and that MCL1 is critical for survival of human AML cells [81].

4.4 Summary

In this chapter, we developed a method for modeling discrete survival times when the number of predictors is much larger than the number of observations. Specifically, we showed that the discrete times can be modeled using a forward continuation ratio model with a complementary log-log link function. When the discrete response measures are intervals in time, there is no clear best way of incorporating censoring times into the likelihood. Therefore, we examined four possible assumptions. The first treats the beginning of the time interval

following the interval in which an observation was censored as a lower bound on the event time for that observation. The second assumption treats the beginning of the time interval in which the censoring occurred as the lower bound. The third assumes that the hazard rate is constant over the period in which the censoring occurred. Finally, we can simply, and naïvely assume that the censoring times are equal to the event times.

The ordinal Generalized Monotone Incremental Forward Stagewise (GMIFS) algorithm allowed us to model both an unpenalized subset of predictors, that are forced into the model with nonzero coefficient estimates, and a penalized subset of predictors, many of which will end up having coefficient estimates equal to zero. Thus, in addition to developing a predictive model, the algorithm also performed feature selection. GMIFS models run until convergence are usually overfit, so we examined the performance of the models stopped at the steps that minimized the AIC and BIC criteria.

Among the four assumptions, the AIC-selected and BIC-selected models fit using assumption 2 ($y_i > c_i - 1$) resulted in the best performance in the data analysis. However, in the simulation studies, models fit using assumption 2 and assumption 4 (the event times are set equal to the censoring times) seemed to perform best among both AIC-selected and BIC-selected models. Specifically, models fit using assumption 4 achieved the highest Somer's D_{XY} measures and were about equivalent to models fit using assumption 2 with regard to feature selection. Based on the combined results of the AML data analysis and the simulation studies, no matter if the purpose of a study is to develop a predictive model or to gain a sense of which features are driving the model, we recommend employing assumption 2 when modeling discrete survival times in high dimensions. Assumption 1 ($y_i > c_i$), which is frequently used in low-dimensional discrete survival time models, performed the worst according to all three criteria (Somer's D_{XY}, sensitivity, and specificity) .

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this dissertation, we developed new methods for predicting an ordinal response in cases when the number of predictors, p, is much larger than the sample size, n. In Chapter 2, we extended the binary classification method, Feature Augmentation via Nonparametrics and Selection (FANS), to the ordinal response setting. Our motivation to develop this novel method arose from the fact that, in the binary response setting, FANS was highly competitive when compared with popular high-dimensional classification methods and outperformed the other methods in many cases. FANS is unique in that it fits a model of augmented features defined in terms of class-conditional kernel density estimates using an ensemble scheme of data splitting and prediction averaging. We implemented two approaches to extend FANS to the ordinal response setting, namely, aggregating $K-1$ penalized binary response models and proportional odds boosting augmented features. We described the two methods in Chapter 2 and compared their performances using a simulation study and an analysis of a gene expression dataset. In the simulation study, we examined the effects of the number of ordinal classes, K, the sample size, n, the number of iterations in the FANS algorithm, L, and the linearity of the decision boundary. Because Ordinal FANS fits a decision boundary that is nonlinear in the original features, both approaches achieved better predictive performance when the simulated decision boundary was nonlinear. When compared against one another, the aggregating penalized binary response models approach had a lower misclassification rate overall when the true decision boundary was linear, while the proportional odds boosting approach performed better when the true decision boundary was nonlinear. With respect to selecting truly important features from the high-dimensional feature space, the Ordinal

FANS proportional odds boosting approach performed better overall. In our data analysis, we showed that both Ordinal FANS approaches accurately utilized high-throughput gene expression data to classify tissues as normal cervical, high-grade intraepithelial lesion, or cervical carcinoma, but the Ordinal FANS proportional odds boosting approach performed better.

In Chapter 3, we compared Ordinal FANS to other competing methods, namely Ordinal GMIFS, P/O Boosting, and k-nearest neighbors. We assessed the models' performances in a simulation study and in three analyses of high-throughput genomic data using a variety of metrics. To assess predictive accuracy, we utilized Somers' D_{XY}, a measure of ordinal association between the predicted and observed outcomes, as well as the misclassification rate. To assess feature selection, we examined the sensitivity and specificity of the fitted models. We showed that both Ordinal FANS approaches performed competitively in the simulation study in terms of predictive accuracy when the true decision boundary was nonlinear, the number of classes in the ordinal outcome was small ($K = 3$ in the simulations), and the sample size was 100 or greater. With respect to feature selection, we demonstrated that the Ordinal FANS proportional odds boosting approach was overall the most specific method and one of the most sensitive as well. Furthermore, in the data analyses, it was apparent that the best method was dependent on the dataset. That is, one method was not best overall, and one method was not worst overall. The Ordinal FANS approaches performed best on one of the datasets, but they were also competitive in all three data analyses.

In Chapter 4, we developed a novel method for predicting a discrete survival time outcome using high-throughput genomic data. We accomplished this by extending the Ordinal GMIFS method to accommodate censoring information and to allow for the use of the complementary log-log link function. We implemented four approaches for accommodating the censoring information, each of which uses a unique assumption. We compared the results of models fit using each assumption in a simulation study and in an analysis of acute myeloid leukemia (AML) patients. The results suggested that the lower bound of a censored pa-

tient's event time should be the discrete time point prior to the time point in which they were censored.

5.2 Future work

In our Ordinal FANS extension, we used kernel density estimation to augment the original features. Specifically, we used a normal kernel function with a normal optimal bandwidth/smoothing parameter [82], given by

$$\left(\frac{4}{3n}\right)^{\frac{1}{5}} * \sigma,$$

where σ denotes the standard deviation. The choice of kernel function is not overly important, but we would like to evaluate different bandwidths. One possibility is to modify the bandwidth to accommodate extreme values and long tailed distributions by utilizing a more robust version of σ, such as the median absolute deviation estimator, given by [82]

$$\tilde{\sigma} = \frac{\text{median}(|y_i - \tilde{\mu}|)}{0.6745},$$

where $\tilde{\mu}$ denotes the median of the observations.

We also saw in Chapter 2 that Ordinal FANS performs much better when the true decision boundary is nonlinear. Fan et al. described a variant of FANS, called FANS2, that includes the original features in addition to the augmented features in the model fitting procedure, which improves performance when a linear decision boundary is reasonable [25]. We plan on implementing FANS2 in the ordinal response setting and comparing it to the Ordinal FANS approaches described in Chapter 2.

Furthermore, it is well known that class imbalance can lead to predictions that favor the class(es) with more observations [74]. Methods such as upsampling, in which the classes with fewer observations are sampled with replacement such that all class-specific sample sizes are equal, have been criticized as unscientific. We plan on examining the impact of class imbalance on the Ordinal FANS algorithm through a simulation study.

Finally, key demographic or clinical predictors that have a known association with the ordinal response should be included in the model in addition to the important genomic features. For instance, age and sex are known to be associated with many diseases. These predictors should not be subject to feature selection by a penalization technique. In other words they should be forced into the model as *unpenalized* predictors. As we mentioned in Chapter 3, the ability to model both penalized and unpenalized predictors is important for any high-dimensional data modeling method. Therefore, we plan on extending the Ordinal FANS method described in Chapter 2 to allow for the inclusion of an unpenalized subset of predictors.

Appendix A

CODE FOR CHAPTER 1

A.1 Ordinal FANS, approach 1

```
#### Ordinal FANS: Approach #1 ############

### Model Fitting

binFANS <- function(x, ...) UseMethod("binFANS") # Generic function

#Function for running ordinal FANS algorithm
binFANS.default <- function(x, y, newx=NULL, niter=1, seed=2468,
                            scale=TRUE, parallel=FALSE) {

  if (class(y)[1] != "ordered") {
    stop("y must be an ordered factor.")
  }
  orig.y <- y
  y <- NULL
  # For each feature, add a small amount of noise to the duplicate values.
  # Avoids issues estimating densities.
  orig.x <- apply(x, 2, function(d) {
    set.seed(seed)
    d[duplicated(d)] <- jitter(d[duplicated(d)], factor=0.5)
    return(d)
```

```
22  })
23  if (scale) {
24    trans <- preProcess(as.data.frame(orig.x), method=c("center", "scale"))
25    if (!is.null(newx)) newx <- predict(trans, as.data.frame(newx))
26    orig.x <- predict(trans, as.data.frame(orig.x))
27  } else {
28    trans <-NULL
29  }
30  x <- NULL
31  K <- length(unique(orig.y))
32  n <- dim(orig.x)[1]
33  levels <- unique(orig.y)
34  set.seed(seed)
35  partitions <- createDataPartition(orig.y, p=0.5,
36                                    times=ceiling(niter / 2))
37  call <- match.call()
38  matrix.sum <- function(mat1, mat2) {
39    return(as.matrix(mat1) + as.matrix(mat2))
40  }
41  # Part of the algorithm that is repeated niter times
42  if (parallel) {
43    `%fun%` <- `%dopar%`
44  } else {
45    `%fun%` <- `%do%`
46  }
47  fitted.model <- foreach(i=1:niter, .combine='rbind',
48                          .packages=c('sm', 'caret')) %fun% {
```

```
49  #Create the data partitions, i.e. (D1, D1^c), (D2, D2^c), ..., (DL, DL^c
       )
50  if(i%%2!=0) {
51    D.train.x <- orig.x[partitions[[(i - 1) / 2 + 1]], ] #x in D1, D3, ...
52    D.train.y <- orig.y[partitions[[(i - 1) / 2 + 1]]] #y in D1, D3, ...
53    D.test.x <- orig.x[-partitions[[(i - 1) / 2 + 1]], ] #x in D1^c, D3^c,
          ...
54    D.test.y <- orig.y[-partitions[[(i - 1) / 2 + 1]]] #y in D1^c, D3^c,
         ...
55  } else {
56    D.train.x <- orig.x[-partitions[[i / 2]], ] #x in D2, D4, D6, ...
57    D.train.y <- orig.y[-partitions[[i / 2]]] #y in D2, D4, D6, ...
58    D.test.x <- orig.x[partitions[[i / 2]], ] #x in D2^c, D4^c, D6^c, ...
59    D.test.y <- orig.y[partitions[[i / 2]]] #y in D2^c, D4^c, D6^c, ...
60  }
61  D.train.y.bin <- matrix(nrow=length(D.train.y), ncol=K - 1)
62  D.test.y.bin <- matrix(nrow=length(D.test.y), ncol=K - 1)
63  for(k in 1:(K - 1)) {
64    D.train.y.bin[, k] <- ifelse(as.numeric(D.train.y) <= k, 1, 0)
65    D.test.y.bin[, k] <- ifelse(as.numeric(D.test.y) <= k, 1, 0)
66  }
67  f.marginals <- array(dim=c(dim(D.test.x)[1], dim(orig.x)[2], K - 1))
68  g.marginals <- array(dim=c(dim(D.test.x)[1], dim(orig.x)[2], K - 1))
69  #if newx is specified, fit model and predict outcome of new observations
70  if(!is.null(newx)) {
71    if(class(newx) == "numeric") {newx <- t(as.matrix(newx))}
72    test.marginals.f <- array(dim=c(dim(newx)[1], dim(orig.x)[2], K - 1))
```

```
73      test.marginals.g <- array(dim=c(dim(newx)[1], dim(orig.x)[2], K - 1))
74      for(k in 1:(K - 1)) {
75        for(j in 1:dim(orig.x)[2]) {
76          f.estimates <- sm.density(D.train.x[as.numeric(D.train.y) <= k, j],
77                                    eval.points=c(D.test.x[, j], newx[, j]),
78                                    display="none")$estimate
79          g.estimates <- sm.density(D.train.x[as.numeric(D.train.y) > k, j],
80                                    eval.points=c(D.test.x[, j], newx[, j]),
81                                    display="none")$estimate
82          #Marginal density estimates for observations in D^c
83          f.marginals[, j, k] <- f.estimates[1:length(D.test.x[, j])]
84          g.marginals[, j, k] <- g.estimates[1:length(D.test.x[, j])]
85          #Marginal density estimates for newx
86          test.marginals.f[, j, k] <- f.estimates[(length(D.test.x[, j]) + 1)
                :
87                                                  length(f.estimates)]
88          test.marginals.g[, j, k] <- g.estimates[(length(D.test.x[, j]) + 1)
                :
89                                                  length(g.estimates)]
90        }
91      }
92      f.marginals[f.marginals < 0.001] <- 0.001
93      g.marginals[g.marginals < 0.001] <- 0.001
94      test.marginals.g[test.marginals.g < 0.001] <- 0.001
95      test.marginals.f[test.marginals.f < 0.001] <- 0.001
96      Z <- log(f.marginals / g.marginals) #array of augmented features.
97      Z.test <- log(test.marginals.f / test.marginals.g)
```

```
98      for (g in 1:dim(Z)[3]) {
99        colnames(Z[, , g]) <- paste(colnames(orig.x), g, sep=".")
100     }
101     #Fit the K - 1 binary response models
102     p.mat <- array(dim=c(K, dim(Z.test)[1], K - 1))
103     create.mat <- function(w) {
104       c(rep(w, k), rep(1 - w, K - k))
105     }
106     coefs <- matrix(nrow=K - 1, ncol=dim(orig.x)[2] + 1)
107     colnames(coefs) <- c("Intercept", colnames(orig.x))
108     rownames(coefs) <- sapply(as.character(1:nrow(coefs)),
109                               function(x) {
110                                 paste("Modeling logit[P(Y <= ", ")]", sep=x)
111                               })
112     fits <- list()
113     min.AIC <- c()
114     for(k in 1:(K - 1)) {
115       fits[[k]] <- glmpath(x=Z[, , k], y=D.test.y.bin[, k],
116                                    family=binomial)
117       min.AIC[k] <- as.numeric(gsub("Step ", "",
118                                rownames(summary(fits[[k]])
119                                [which.min(summary(fits[[k]])$AIC), ])))
120       p.hat <- predict(fits[[k]], newx=Z.test[, , k],
121                        s=min.AIC[k], type="response") #predict y<=k or y>k
122       coefs[k, ] <- predict(fits[[k]], s=min.AIC[k], type="coefficients")
123       p.mat[, , k] <- sapply(p.hat, create.mat)
124     }
```

```
125     scores <- apply(p.mat, c(1, 2), sum) #aggregate binary predictions
126     return(list(fits=fits, minAIC=min.AIC, scores=t(scores), coefs=coefs,
127                 D.train.x=D.train.x, D.train.y=D.train.y, trans=trans))
128   #if newx is NULL, just fit K - 1 binary response models
129   } else {
130     for(k in 1:(K - 1)) {
131       for(j in 1:dim(orig.x)[2]) {
132         #Marginal density estimates for observations in D^c
133         f.marginals[, j, k] <- sm.density(D.train.x[as.numeric(D.train.y)
              <= k, j],
134                                           eval.points=c(D.test.x[, j]),
135                                           display="none")$estimate
136         g.marginals[, j, k] <- sm.density(D.train.x[as.numeric(D.train.y) >
               k, j],
137                                           eval.points=c(D.test.x[, j]),
138                                           display="none")$estimate
139       }
140     }
141     f.marginals[f.marginals < 0.001] <- 0.001
142     g.marginals[g.marginals < 0.001] <- 0.001
143     Z <- log(f.marginals / g.marginals)
144     #Fit the K - 1 binary response models
145     fits <- list()
146     coefs <- matrix(nrow=K - 1, ncol=dim(orig.x)[2] + 1)
147     colnames(coefs) <- c("Intercept", colnames(orig.x))
148     rownames(coefs) <- sapply(as.character(1:nrow(coefs)),
149                               function(x) {
```

```
150                               paste("Modeling logit[P(Y <= ", ")]", sep=x)
151                             })
152     min.AIC <- c()
153     for(k in 1:(K - 1)) {
154       fits[[k]] <- glmpath(x=Z[, , k], y=D.test.y.bin[, k], family=
              binomial)
155       min.AIC[k] <- as.numeric(gsub("Step ", "",
156                                rownames(summary(fits[[k]])
157                                [which.min(summary(fits[[k]])$AIC), ])))
158       coefs[k, ] <- predict(fits[[k]], s=min.AIC[k], type="coefficients")
159     }
160     return(list(fits=fits, x=orig.x, D.train.x=D.train.x, coefs=coefs,
161                 D.train.y=D.train.y, K=K, minAIC=min.AIC, niter=niter,
162                 call=call, trans=trans))
163   }
164 }
165 if (niter == 1) {
166   if(!is.null(newx)) {
167     output <- list(coefs=fitted.model$coefs, niter=niter, K=K, x=orig.x,
168                    fits=fitted.model$fits, minAIC=fitted.model$minAIC,
169                    D.train.x=fitted.model$D.train.x, y=orig.y,
170                    D.train.y=fitted.model$D.train.y, call=call,
171                    trans=trans, newx=newx, scores=fitted.model$scores,
172                    pred.class=ordered(unique(orig.y)[apply(fitted.model$
                          scores,
173                                                           1, which.max)],
174                                       levels=levels(orig.y)))
```

```
175   } else {
176     output <- list(coefs=fitted.model$coefs, niter=niter, K=K, x=orig.x,
177                    fits=fitted.model$fits, minAIC=fitted.model$minAIC,
178                    D.train.x=fitted.model$D.train.x, y=orig.y,
179                    D.train.y=fitted.model$D.train.y, call=call,
180                    trans=trans)
181   }
182 } else {
183   if(!is.null(newx)) {
184     fits <- lapply(fitted.model[, 1], as.matrix)
185     minAIC <- fitted.model[, 2]
186     scores <- lapply(fitted.model[, 3], as.matrix)
187     agg.scores <- Reduce('+',lapply(fitted.model[, 3], as.matrix))
188     coefs <- lapply(fitted.model[, 4], as.matrix)
189     D.train.x <- lapply(fitted.model[, 5], as.matrix)
190     D.train.y <- lapply(fitted.model[, 6], unlist)
191     pred.class <- ordered(unique(orig.y)[apply(agg.scores, 1, which.max)],
192                           levels=levels(orig.y))
193     output <- list(coefs=coefs, niter=niter, K=K, x=orig.x, fits=fits,
194                minAIC=minAIC, D.train.x=D.train.x, y=orig.y,
195                D.train.y=D.train.y, call=call, trans=trans, newx=newx,
196                scores=scores, agg.scores=agg.scores, pred.class=pred.class
                      )
197   } else {
198     fits <- lapply(fitted.model[, 1], as.matrix)
199     D.train.x <- lapply(fitted.model[, 3], as.matrix)
200     D.train.y <- lapply(fitted.model[, 5], unlist)
```

```
201       minAIC <- fitted.model[, 7]
202       coefs <- lapply(fitted.model[, 4], as.matrix)
203       output <- list(coefs=coefs, niter=niter, K=K, x=orig.x, fits=fits,
204                      minAIC=minAIC, D.train.x=D.train.x, y=orig.y,
205                      D.train.y=D.train.y, call=call, trans=trans)
206     }
207   }
208   class(output) <- "binFANS"
209   return(output)
210 }
```

```
 1
 2 ### Extract coefficient estimates
 3
 4 coef.binFANS <- function(object, model.select="mean", only.nonzero=FALSE) {
 5   if(is.numeric(model.select) & model.select > object$niter) {
 6     stop(paste("model.select cannot be greater than ", object$niter, sep="")
         )
 7   }
 8   if (object$niter > 1) {
 9     if (model.select=="mean") {
10       mean.coefs <- Reduce('+', object$coefs) / object$niter
11       if (only.nonzero) {
12         beta <- mean.coefs[, which(apply(mean.coefs, 2,
13                                          function(x) abs(sum(x)))!=0)]
14       }
15     } else if (is.numeric(model.select)) {
16       beta <- object$coefs[[model.select]]
```

```
      if (only.nonzero) {
        beta <- beta[, which(apply(beta, 2, function(x) abs(sum(x)))!=0)]
      }
    } else if (model.select=="all") {
      beta <- object$coefs
      if (only.nonzero) {
        nonzero <- function(x) {
          x[, which(apply(x, 2, function(y) abs(sum(y)))!=0)]
        }
        beta <- lapply(beta, function(l) nonzero(l))
      }
    }
  } else {
    beta <- object$coefs
    if (only.nonzero) {
      beta <- beta[, which(apply(beta, 2, function(x) abs(sum(x)))!=0)]
    }
  }
  beta
}
```

```
### Plot coefficient paths of individual binary response models

plot.binFANS <- function(object, type="coefficients", xlab=NULL,
                         ylab=NULL, main=NULL, ...) {
  num.plots <- object$niter * (object$K - 1)
  if (object$niter > 1) {
    for (l in 1:object$niter) {
```

```
    for (k in 1:(object$K - 1)) {
      if(is.null(main)) {
        title <- paste(paste("FANS Iteration: ", l),
                       paste("Modeling Augmented Feature: ", "",
                       sep=as.character(k)), sep="\n")
      }
      x11()
      par(oma=c(0,0,1,1))
      plot(object$fits[[l]][[k]], main="", breaks=FALSE, type=type, ...)
      title(main=title)
    }
  }
} else {
  for (k in 1:num.plots) {
    if (is.null(main)) {
      title <- paste(paste("FANS Iteration: ", object$niter),
                     paste("Modeling Augmented Feature: ", "",
                     sep=as.character(k)), sep="\n")
    }
    x11()
    par(oma=c(0,0,1,1))
    plot(object$fits[[k]], main="", breaks=FALSE, type=type, ...)
    title(main=title)
  }
}
}
```

```
### Predict the outcome of a new observation
```

```
2
3 predict.binFANS <- function(object, newx) {
4   newx <- predict(object$trans, newx) # Scale newx with mean and sd of X
5   if (!is.null(object$newx)) {
6     if (!is.null(object$pred.class) & all.equal(as.matrix(object$newx),
7                                                 newx)[1]) {
8       #if (type=="class") {
9       # return(object$pred.class)
10      #} else {
11      # return(object$scores)
12      #}
13      if (object$niter == 1) {
14        return(list(scores=object$scores, pred.class=object$pred.class))
15      } else {
16        return(list(scores=object$scores, agg.scores=object$agg.scores,
17                    pred.class=object$pred.class))
18      }
19    }
20  }
21  K <- object$K
22  x <- object$x
23  niter <- object$niter
24  predicted <- foreach(i=1:niter, .combine='rbind',
25                       .packages=c('sm', 'caret')) %dopar% {
26    if (niter==1) {
27      D.train.x <- object$D.train.x
28      D.train.y <- object$D.train.y
```

```
29      fits <- object$fits
30      min.AIC <- object$minAIC
31    } else {
32      D.train.x <- object$D.train.x[[i]]
33      D.train.y <- object$D.train.y[[i]]
34      fits <- object$fits[[i]]
35      min.AIC <- object$minAIC[[i]]
36    }
37    test.marginals.f <- array(dim=c(dim(newx)[1],
38                              dim(newx)[2], K - 1))
39    test.marginals.g <- array(dim=c(dim(newx)[1],
40                              dim(newx)[2], K - 1))
41    for(k in 1:(K - 1)) {
42      for(j in 1:dim(x)[2]) {
43          test.marginals.f[, j, k] <-
44            sm.density(D.train.x[as.numeric(D.train.y) <= k, j],
45                                          eval.points=c(newx[, j]),
46                                          display="none")$estimate
47          test.marginals.g[, j, k] <-
48            sm.density(D.train.x[as.numeric(D.train.y) > k, j],
49                                          eval.points=c(newx[, j]),
50                                          display="none")$estimate
51      }
52    }
53    test.marginals.g[test.marginals.g < 0.001] <- 0.001
54    test.marginals.f[test.marginals.f < 0.001] <- 0.001
55    Z.test <- log(test.marginals.f / test.marginals.g)
```

```
56    p.mat <- array(dim=c(K, dim(Z.test)[1], K - 1))
57    create.mat <- function(w) {
58      c(rep(w, k), rep(1 - w, K - k))
59    }
60    for(k in 1:(K - 1)) {
61      p.hat <- predict(fits[[k]], newx=Z.test[, , k],
62                       s=min.AIC[k], type="response") #predict y<=k or y>k
63      p.mat[, , k] <- sapply(p.hat, create.mat)
64    }
65    scores <- t(apply(p.mat, c(1, 2), sum)) #aggregate binary predictions
66    class <- apply(scores, 1, which.max)
67    return(list(scores=scores, class=class))
68  }
69  if(niter > 1) {
70    scores <- lapply(predicted[, 1], as.matrix)
71    agg.scores <- Reduce('+', scores)
72   # if (type=="class") {
73      class <- apply(agg.scores, 1, which.max)
74   # return(pred.class=class)
75   # } else {
76    return(list(scores=scores,
77                agg.scores=agg.scores,
78                pred.class=levels(object$y)[class]))
79    #}
80  } else {
81    #if (type=="class") {
82    # return(pred.class=predicted$class)
```

```
    #} else {
    return(list(scores=predicted$scores,
                pred.class=levels(object$y)[predicted$class]))
    #}
  }
}
```

```
### Another function that does the same thing as predict.binFANS()

fitted.binFANS <- function(object, newx) {
  predict.binFANS(object=object, newx=newx)
}
```

```
### Print a summary of the model fitting

print.binFANS <- function(object, ...) {
  cat("Call:\n")
  print(object$call)
  cat("\nNumber of FANS iterations: niter = ", object$niter)
  cat("\n")
}
```

```
### Summary of the model fitting

summary.binFANS <- function(object) {
  features <- colnames(coef.binFANS(object, model.select="mean",
                      only.nonzero=TRUE))[-1]
  return(list(object=object, features=features))
}
```

A.2 Ordinal FANS, approach 2

```
#### Ordinal FANS: Approach #2 ########################################

### Model Fitting

sum.across.L <- function(Li, Lj) {
  # Sums the results of the L fitted models
  #
  # Args:
  # Li: list of results of a fitted model
  # Lj: list of results of another fitted model
  #
  # Returns:
  # A list composed of the sums of the elements of the L
  # lists produced from the L fitted models.
  tmp<-rbind(Li, Lj)
  apply(tmp, 2, function(x) Reduce('+', x))
}

ordinalFANS <- function(x, ...) UseMethod("ordinalFANS") # Generic function

ordinalFANS.default<-function(x, y, newx=NULL, eps=0.1, niter=1,
                              seed=2468, mstop=100, scale=TRUE,
                              parallel=FALSE) {
  # Runs the ordinal FANS algorithm
  #
```

```
26  # Args:
27  # x: the n x p design matrix
28  # y: the response vector
29  # newx: (optional) design matrix with which to predict outcomes
30  # niter: number of times to repeat the algorithm with new data partitions
31  # seed: sets the seed for the initial partitioning of the data
32  # mstop: Stopping iteration
33  #
34  call <- match.call()
35  if (class(y)[1] != "ordered") {
36    stop("y must be an ordered factor.")
37  }
38  orig.y <- y
39  y <- NULL
40  # For each feature, add a small amount of noise to the duplicate values.
41  # Avoids issues estimating densities.
42  orig.x <- apply(x, 2, function(d) {
43    set.seed(seed)
44    d[duplicated(d)] <- jitter(d[duplicated(d)], factor=0.5)
45    return(d)
46  })
47  if (scale) {
48    trans <- preProcess(as.data.frame(orig.x), method=c("center", "scale"))
49    if (!is.null(newx)) newx <- predict(trans, as.data.frame(newx))
50    orig.x <- predict(trans, as.data.frame(orig.x))
51  } else {
52    trans <-NULL
```

```
53  }
54  x <- NULL
55  K <- length(unique(orig.y))
56  n <- dim(orig.x)[1]
57  levels <- unique(orig.y)
58  set.seed(seed)
59  # Stratified random sampling
60  partitions <- createDataPartition(orig.y, p=0.5,
61                                    times=ceiling(niter / 2))
62
63  # Part of the algorithm that is repeated niter times
64  if (parallel) {
65    `%fun%` <- `%dopar%`
66  } else {
67    `%fun%` <- `%do%`
68  }
69  if (niter > 1) {
70    fitted.model <- foreach(i=1:niter, .combine='rbind',
71                            .packages=c('sm', 'caret')) %fun% {
72      # Create the data partitions, (D1, D1^c), (D2, D2^c), ..., (DL, DL^c)
73      if (i%%2 != 0) {
74        D.train.x <- orig.x[partitions[[(i - 1) / 2 + 1]], ]
75        D.train.y <- orig.y[partitions[[(i - 1) / 2 + 1]]]
76        D.test.x <- orig.x[-partitions[[(i - 1) / 2 + 1]], ]
77        D.test.y <- orig.y[-partitions[[(i - 1) / 2 + 1]]]
78      } else {
79        D.train.x <- orig.x[-partitions[[i / 2]], ]
```

```
80        D.train.y <- orig.y[-partitions[[i / 2]]]
81        D.test.x <- orig.x[partitions[[i / 2]], ]
82        D.test.y <- orig.y[partitions[[i / 2]]]
83      }
84      # Estimate marginals using data in Di, evaluate using data in Di^c,
85      # and calculate augmented features (log ratios)
86      # If newx is supplied, fit model, then find predicted values for new
          obs
87      if (!is.null(newx)) {
88        f.estimates <- array(dim=c(dim(D.test.x)[1] + dim(newx)[1],
89                             K - 1, dim(orig.x)[2]))
90        g.estimates <- array(dim=c(dim(D.test.x)[1] + dim(newx)[1],
91                             K - 1, dim(orig.x)[2]))
92        for (k in 1:(K - 1)) {
93          for (j in 1:dim(orig.x)[2]) {
94            f.estimates[, k, j] <-
95              sm.density(D.train.x[which(as.numeric(D.train.y) <= k), j],
96                         eval.points=c(D.test.x[, j], newx[, j]),
97                         display="none")$estimate
98            g.estimates[, k, j] <-
99              sm.density(D.train.x[which(as.numeric(D.train.y) > k), j],
100                        eval.points=c(D.test.x[, j], newx[, j]),
101                        display="none")$estimate
102         }
103       }
104       # Marginal density estimates for observations in D^c
105       f.marginals <- f.estimates[1:dim(D.test.x)[1], , ]
```

```
106       g.marginals <- g.estimates[1:dim(D.test.x)[1], , ]
107       # Marginal density estimates for newx
108       test.marginals.f <- f.estimates[(dim(D.test.x)[1] + 1):
109                                       dim(f.estimates)[1], , ]
110       test.marginals.g <- g.estimates[(dim(D.test.x)[1] + 1):
111                                       dim(g.estimates)[1], , ]
112
113       # Winsorization to improve stability of estimates as
114       # suggested in FANS manuscript
115       f.marginals[f.marginals < 0.001] <-
116         g.marginals[g.marginals < 0.001] <-
117         test.marginals.g[test.marginals.g < 0.001] <-
118         test.marginals.f[test.marginals.f < 0.001] <- 0.001
119       # Array of augmented features for x. Within the array:
120       # n x (K - 1) matrices of augmented features .
121       # array dimensions are n x (K - 1) x p
122       Z <- log(f.marginals / g.marginals)
123       full <- data.frame(Z)
124       colnames(full) <- paste("z", paste(rep(1:dim(Z)[3], each=K - 1),
125                                rep(1:(K - 1), times=dim(Z)[3]),
126                                sep="."), sep="")
127       # Array of augmented features for newx. Within the array:
128       # n x (K - 1) matrices of augmented features .
129       # array dimensions are n x (K - 1) x p
130       Z.test <- log(test.marginals.f / test.marginals.g)
131       full.test <- data.frame(Z.test)
132       colnames(full.test) <- paste("z", paste(rep(1:dim(Z)[3], each=K - 1)
```

```
    ,
                              rep(1:(K - 1),
                              times=dim(f.marginals)[3]),
                              sep="."), sep="")
  orig.vars.index <- rep(1:dim(Z)[3], each=K - 1)
  # Fit proportional odds boosting model
  fit <- PO.boost(x=full, y=D.test.y, eps=eps, mstop=mstop)
  # Variable importance measure = absolute value of sum of the (K - 1)
  # augmented features coefficient estimates
  var.importance <- apply(fit$coefs, 1, function(x) {
    tapply(x, orig.vars.index, function(y) {
      abs(sum(y))
    })
  })
  var.importance <- data.frame(t(var.importance))
  colnames(var.importance) <- colnames(orig.x)
  # function estimates for newx
  offset <- as.numeric(-1 * (matrix(apply(full, 2, mean), nrow=1) %*%
           matrix(fit$coefs[mstop, ], ncol=1)))
  f.newx <- offset + (as.matrix(full.test) %*% fit$coefs[mstop, ])
  # estimated posterior probabilities for newx
  post.probs <- response(f.newx, fit$theta[mstop, ])
  risk <- apply(fit$loss, 1, sum)
  return(list(coefs=fit$coefs, theta=fit$theta, risk=risk,
              post.probs=post.probs, D.train.x=D.train.x,
              D.train.y=D.train.y, trans=trans,
              var.importance=var.importance,
```

```
159                 aug.newX=as.matrix(full.test), aug.X=as.matrix(full)))
160
161
162       # If newx is NOT supplied
163     } else {
164       # Initialize array of f marginals
165       f.marginals <- array(dim=c(dim(D.test.x)[1],
166                                  K - 1, dim(orig.x)[2]))
167       # Initialize array of g marginals
168       g.marginals <- array(dim=c(dim(D.test.x)[1],
169                                  K - 1, dim(orig.x)[2]))
170       for (k in 1:(K - 1)) {
171         for (j in 1:dim(orig.x)[2]) {
172           # Marginal density estimates for observations in D^c
173           f.marginals[, k, j] <-
174             sm.density(D.train.x[which(as.numeric(D.train.y) <= k), j],
175                        eval.points=D.test.x[, j],
176                        display="none")$estimate
177           g.marginals[, k, j] <-
178             sm.density(D.train.x[which(as.numeric(D.train.y) > k), j],
179                        eval.points=D.test.x[, j],
180                        display="none")$estimate
181         }
182       }
183       # Winsorization to improve stability of estimates
184       # as suggested in FANS
185       f.marginals[f.marginals < 0.001] <-
```

```
186       g.marginals[g.marginals < 0.001] <- 0.001
187       # Array of augmented features for x. Within the array:
188       # n x (K - 1) matrices of augmented features.
189       # Array dimensions are n x (K - 1) x p
190       Z <- log(f.marginals / g.marginals)
191       full <- data.frame(Z)
192       colnames(full) <- paste("z",
193                               paste(rep(1:dim(Z)[3], each=K - 1),
194                                     rep(1:(K - 1), times=dim(Z)[3]),
195                                     sep="."), sep="")
196       orig.vars.index <- rep(1:dim(Z)[3], each=K - 1)
197       # Fit proportional odds boosting model
198       fit <- PO.boost(x=full, y=D.test.y, mstop=mstop)
199       var.importance <- apply(fit$coefs, 1, function(x) {
200         tapply(x, orig.vars.index, function(y) {
201           abs(sum(y))
202         })
203       })
204       var.importance <- data.frame(t(var.importance))
205       colnames(var.importance) <- colnames(orig.x)
206       risk <- apply(fit$loss, 1, sum)
207       return(list(coefs=fit$coefs, theta=fit$theta, risk=risk,
208                   D.train.x=D.train.x, D.train.y=D.train.y, trans=trans,
209                   var.importance=var.importance, aug.X=as.matrix(full)))
210     }
211   }
212   if (!is.null(newx)) {
```

```
213     coefs <- lapply(fitted.model[, 1], as.matrix)
214     mean.coefs <- Reduce('+', fitted.model[, 1]) / niter
215     theta <- lapply(fitted.model[, 2], as.matrix)
216     mean.theta <- Reduce('+', fitted.model[, 2]) / niter
217     risk <- lapply(fitted.model[, 3], as.numeric)
218     mean.risk <- as.numeric(Reduce('+', risk) / niter)
219     if (dim(newx)[1] == 1) {
220       posteriors <- lapply(fitted.model[, 4], function(q) t(as.matrix(q)))
221     } else {
222       posteriors <- lapply(fitted.model[, 4], as.matrix)
223     }
224     mean.posteriors <- Reduce('+', posteriors) / niter
225     D.train.x <- lapply(fitted.model[, 5], as.matrix)
226     D.train.y <- lapply(fitted.model[, 6], unlist)
227     pred.class <-
228       factor(levels(orig.y)[apply(mean.posteriors, 1, which.max)],
229              levels=levels(orig.y), ordered=TRUE)
230     var.importance <- lapply(fitted.model[, 8], data.frame)
231     mean.var.importance <-
232       data.frame(Reduce('+', fitted.model[, 8]) / niter)
233     aug.newX <- lapply(fitted.model[, 9], as.matrix)
234     aug.X <- lapply(fitted.model[, 10], as.matrix)
235     output <- list(call=call, eps=eps, mstop=mstop,
236                    niter=niter, K=K, coefs=coefs,
237                    mean.coefs=mean.coefs, theta=theta,
238                    mean.theta=mean.theta,
239                    risk=risk, mean.risk=mean.risk, y=orig.y,
```

```
240                 D.train.x=D.train.x, D.train.y=D.train.y,
241                 trans=trans, var.importance=var.importance,
242                 aug.X=aug.X,
243                 mean.var.importance=mean.var.importance,
244                 newx=newx, aug.newX=aug.newX,
245                 posteriors=posteriors,
246                 mean.posteriors=mean.posteriors,
247                 pred.class=pred.class)
248   }
249   else {
250     coefs <- lapply(fitted.model[, 1], as.matrix)
251     mean.coefs <- Reduce('+', fitted.model[, 1]) / niter
252     theta <- lapply(fitted.model[, 2], as.matrix)
253     mean.theta <- Reduce('+', fitted.model[, 2]) / niter
254     risk <- lapply(fitted.model[, 3], as.numeric)
255     mean.risk <- as.numeric(Reduce('+', risk) / niter)
256     D.train.x <- lapply(fitted.model[, 4], as.matrix)
257     D.train.y <- lapply(fitted.model[, 5], unlist)
258     var.importance <- lapply(fitted.model[, 7], data.frame)
259     aug.X <- lapply(fitted.model[, 8], as.matrix)
260     mean.var.importance <-
261       data.frame(Reduce('+', fitted.model[, 7]) / niter)
262     output <- list(call=call, eps=eps, mstop=mstop,
263                 niter=niter, K=K, coefs=coefs,
264                 mean.coefs=mean.coefs, theta=theta,
265                 mean.theta=mean.theta,
266                 risk=risk, mean.risk=mean.risk, y=orig.y,
```

```
                  D.train.x=D.train.x, D.train.y=D.train.y,
                  trans=trans, var.importance=var.importance,
                  aug.X=aug.X, mean.var.importance=mean.var.importance)
  }


# if niter=1
} else {
  # Create the data partitions, i.e. (D1, D1^c)
  D.train.x <- orig.x[partitions[[1]], ] #x in D1
  D.train.y <- orig.y[partitions[[1]]] #y in D1
  D.test.x <- orig.x[-partitions[[1]], ] #x in D1^c
  D.test.y <- orig.y[-partitions[[1]]] #y in D1^c
  # Estimate marginals using data in Di, evaluate using data in Di^c,
  # and calculate augmented features (log ratios)
  # If newx is supplied, fit model, then find predicted values for new obs
  if (!is.null(newx)) {
    f.estimates <- array(dim=c(dim(D.test.x)[1] + dim(newx)[1],
                               K - 1, dim(orig.x)[2]))
    g.estimates <- array(dim=c(dim(D.test.x)[1] + dim(newx)[1],
                               K - 1, dim(orig.x)[2]))
    for (k in 1:(K - 1)) {
      for (j in 1:dim(orig.x)[2]) {
        f.estimates[, k, j] <-
          sm.density(D.train.x[which(as.numeric(D.train.y) <= k), j],
                     eval.points=c(D.test.x[, j], newx[, j]),
                     display="none")$estimate
```

```
294         g.estimates[, k, j] <-
295           sm.density(D.train.x[which(as.numeric(D.train.y) > k), j],
296                      eval.points=c(D.test.x[, j], newx[, j]),
297                      display="none")$estimate
298       }
299     }
300     # Marginal density estimates for observations in D^c
301     f.marginals <- f.estimates[1:dim(D.test.x)[1], , ]
302     g.marginals <- g.estimates[1:dim(D.test.x)[1], , ]
303     # Marginal density estimates for newx
304     test.marginals.f <- f.estimates[(dim(D.test.x)[1] + 1):
305                                     dim(f.estimates)[1], , ]
306     test.marginals.g <- g.estimates[(dim(D.test.x)[1] + 1):
307                                     dim(g.estimates)[1], , ]
308     # Winsorization to improve stability of estimates
309     # as suggested in FANS manuscript
310     f.marginals[f.marginals < 0.001] <-
311       g.marginals[g.marginals < 0.001] <-
312       test.marginals.g[test.marginals.g < 0.001] <-
313       test.marginals.f[test.marginals.f < 0.001] <- 0.001
314       # Array of augmented features for x. Within the array:
315       # n x (K - 1) matrices of augmented features.
316       # Array dimensions are n x (K - 1) x p
317     Z <- log(f.marginals / g.marginals)
318     full <- data.frame(Z)
319     colnames(full) <- paste("z",
320                             paste(rep(1:dim(Z)[3], each=K - 1),
```

```
                              rep(1:(K - 1), times=dim(Z)[3]),
                              sep="."), sep="")
  # Array of augmented features for newx. Within the array:
  # n x (K - 1) matrices of augmented features .
  # array dimensions are n x (K - 1) x p
Z.test <- log(test.marginals.f / test.marginals.g)
full.test <- data.frame(Z.test)
colnames(full.test) <- paste("z",
                             paste(rep(1:dim(Z)[3], each=K - 1),
                             rep(1:(K - 1),
                             times=dim(f.marginals)[3]),
                             sep="."), sep="")
orig.vars.index <- rep(1:dim(Z)[3], each=K - 1)
# Fit proportional odds boosting model
fit <- PO.boost(x=full, y=D.test.y, eps=eps, mstop=mstop)
# Variable importance measure = absolute value of sum of the (K - 1)
# augmented features coefficient estimates
var.importance <- apply(fit$coefs, 1, function(x) {
  tapply(x, orig.vars.index, function(y) {
    abs(sum(y))
  })
})
var.importance <- data.frame(t(var.importance))
colnames(var.importance) <- colnames(orig.x)
# function estimates for newx
offset <- as.numeric(-1 * (matrix(apply(full, 2, mean), nrow=1) %*%
         matrix(fit$coefs[mstop, ], ncol=1)))
```

```
348     f.newx <- offset + (as.matrix(full.test) %*% fit$coefs[mstop, ])
349     # estimated posterior probabilities for newx
350     post.probs <- response(f.newx, fit$theta[mstop, ])
351     risk <- apply(fit$loss, 1, sum)
352
353
354     # If newx is NOT supplied
355   } else {
356     # Initialize array of f marginals
357     f.marginals <- array(dim=c(dim(D.test.x)[1],
358                          K - 1, dim(orig.x)[2]))
359     # Initialize array of g marginals
360     g.marginals <- array(dim=c(dim(D.test.x)[1],
361                          K - 1, dim(orig.x)[2]))
362     for (k in 1:(K - 1)) {
363       for (j in 1:dim(orig.x)[2]) {
364         # Marginal density estimates for observations in D^c
365         f.marginals[, k, j] <-
366           sm.density(D.train.x[which(as.numeric(D.train.y) <= k), j],
367                                eval.points=D.test.x[, j],
368                                display="none")$estimate
369         g.marginals[, k, j] <-
370           sm.density(D.train.x[which(as.numeric(D.train.y) > k), j],
371                                eval.points=D.test.x[, j],
372                                display="none")$estimate
373       }
374     }
```

```
375     # Winsorization to improve stability of estimates
376     # as suggested in the FANS manuscript
377     f.marginals[f.marginals < 0.001] <-
378     g.marginals[g.marginals < 0.001] <- 0.001
379     # Array of augmented features for x. Within the array:
380     # n x (K - 1) matrices of augmented features .
381     # array dimensions are n x (K - 1) x p
382     Z <- log(f.marginals / g.marginals)
383     full <- data.frame(Z)
384     colnames(full) <- paste("z",
385                             paste(rep(1:dim(Z)[3], each=K - 1),
386                                   rep(1:(K - 1), times=dim(Z)[3]),
387                                   sep="."), sep="")
388     orig.vars.index <- rep(1:dim(Z)[3], each=K - 1)
389     # Fit proportional odds boosting model
390     fit <- PO.boost(x=full, y=D.test.y, mstop=mstop)
391     var.importance <- apply(fit$coefs, 1, function(x) {
392       tapply(x, orig.vars.index, function(y) {
393         abs(sum(y))
394       })
395     })
396     var.importance <- data.frame(t(var.importance))
397     colnames(var.importance) <- colnames(orig.x)
398     risk <- apply(fit$loss, 1, sum)
399   }
400   if (!is.null(newx)) {
401     coefs <- as.matrix(fit$coefs)
```

```
402     theta <- as.matrix(fit$theta)
403     risk <- as.numeric(risk)
404     if (dim(newx)[1] == 1) {
405       posteriors <- t(as.matrix(post.probs))
406     } else {
407       posteriors <- as.matrix(post.probs)
408     }
409     pred.class <-
410       factor(levels(orig.y)[apply(posteriors, 1, which.max)],
411              levels=levels(orig.y), ordered=TRUE)
412     D.train.x <- as.matrix(D.train.x)
413     D.train.y <- as.matrix(D.train.y)
414     var.importance <- data.frame(var.importance)
415     aug.newX <- as.matrix(full.test)
416     output <- list(call=call, eps=eps, mstop=mstop, niter=niter,
417                    K=K, coefs=coefs, theta=theta, risk=risk,
418                    y=orig.y, D.train.x=D.train.x,
419                    D.train.y=D.train.y, trans=trans,
420                    var.importance=var.importance, aug.X=full,
421                    newx=newx, aug.newX=aug.newX,
422                    posteriors=posteriors, pred.class=pred.class)
423   } else {
424     coefs <- as.matrix(fit$coefs)
425     theta <- as.matrix(fit$theta)
426     risk <- as.numeric(risk)
427     D.train.x <- as.matrix(D.train.x)
428     D.train.y <- as.matrix(D.train.y)
```

```
429       var.importance <- data.frame(var.importance)
430       output <- list(call=call, eps=eps, mstop=mstop,
431                      niter=niter, K=K, coefs=coefs,
432                      theta=theta, risk=risk, y=orig.y,
433                      D.train.x=D.train.x,
434                      D.train.y=D.train.y, trans=trans,
435                      var.importance=var.importance, aug.X=full)
436     }
437   }
438   class(output) <- "ordinalFANS"
439   return(output)
440 }
```

```
1 ### Proportional Odds Boosting
2
3 PO.boost<-function(x, y, eps=0.1, w=1, mstop) {
4   # Proportional odds (P/O) boosting algorithm
5   #
6   # Args:
7   # x: Design matrix
8   # y: Response vector
9   # eps: A real-valued step length factor
10  # w: Vector of weights
11  # K: Number of levels in the response
12  # mstop: Stopping iteration
13  # Returns:
14  # coefs: Vector of coefficient estimates
15  # f.hat: Vector of function estimate
```

```
16  # theta: Vector of threshold estimates
17  ###
18  # Step 1: Initialize the n-dimensional vector f_hat[0] and the
19  # K1 threshold parameter estimates with offset values.
20  #y <- factor(y, ordered=TRUE)
21  K <- length(unique(y))
22  p <- dim(x)[2] / (K - 1)
23  x <- scale(x, center=T, scale=F)
24  f.hat <- rep(0, length(y))
25  pi.0 <- table(y) / length(y)
26  theta <- matrix(0, nrow=mstop, ncol=K - 1)
27  theta[1, ] <- delta <- log(cumsum(pi.0) / (1 - cumsum(pi.0)))[1:(K - 1)]
28  colnames(theta) <- paste("theta", 1:(K - 1), sep="")
29  # Matrix of coefficient estimates (in all m steps)
30  coefs <- matrix(0, nrow=mstop, ncol=dim(x)[2])
31  # Matrix of latest, updated vector of coefficient estimates
32  coefs.latest <- rep(0, dim(x)[2])
33  colnames(coefs) <- names(coefs.latest) <- colnames(x)
34  all.loss <- matrix(0, nrow=mstop, ncol=length(y))
35  # Step 2: Specify base-learners, set m = 0.
36  vars <- list()
37  for (w in 1:p) {
38    vars[[w]] <- colnames(x)[(w + (w-1)):(w + (w-1) + (K-2))]
39  }
40  m <- 0
41  # Iterate mstop times.
42  for (j in 1:mstop) {
```

```
43    # Step 3: Increase m by 1.
44    m <- m + 1
45    # Step 4a: Calculate negative gradient vector with current estimate of
46    # theta and f
47    U <- neg.grad(y=y, f=f.hat, theta = theta[m, ])
48    # Step 4b: Fit the negative gradient vector U[m] using each of the p
49    # base learners. This yields p vectors of predicted values, where each
50    # vector is an estimate of the negative gradient vector U[m].
51    # Use R^2 to determine best base learner.
52    fit.rsq <- function(aug.vars) {
53      # Fit a linear model for a given base learner
54      #
55      # Args:
56      # aug.vars: Names of the augmented features used to fit the model
57      # Returns:
58      # R^2 for given model.
59      # fit.i <- glmnet(x=data.matrix(x[, aug.vars]), y=U,
60      # family="gaussian", alpha = 0)
61      fit.i <- lm(U ~ 0 + ., data=data.frame(U, x[, aug.vars]))
62      return(summary(fit.i)$r.squared)
63      #return(max(fit.i$dev.ratio)) # Return max R^2 of all lambda values
            used
64    }
65    r.sq <- vapply(vars, fit.rsq, 1) # R^2 for all p base learners
66    # Step 4c: Select the base-learner that fits U[m] best according to the
         Rsq
67    # goodness-of-fit criterion. Set U_hat[m] equal to the fitted values of
```

```
        the
    # best model.
    max.rsq <- which.max(r.sq) # Base learner with largest R^2
    fit.maxrsq <- lm(U ~ 0 + ., data=data.frame(U, x[, vars[[max.rsq]]]))
    u.hat <- predict(fit.maxrsq, newdata = data.frame(x[, vars[[max.rsq]]]),
                     interval="none")
    vars.update <- names(fit.maxrsq$coefficients)
    coefs.latest[vars.update] <- coefs.latest[vars.update] +
                                 eps * fit.maxrsq$coefficients
    coefs[m, ] <- coefs.latest
    # Step 4d: Update f_hat[m]  f_hat[m1] + eps * U_hat[m], where 0 < eps  1
    # is a real-valued step length factor.
    f.hat <- f.hat + eps * u.hat
    # Step 5: Plug f_hat[m] into the empirical risk function and minimize
        the
    # empirical risk over . Set _hat[m] equal to the newly obtained estimate
    # of . (Convert delta (unconstrained) to theta, minimize risk over theta
        ,
    # return optimum delta).
    delta <- optim(par = delta, fn = riskS, y = y,
              fit = f.hat, w = w, method = "BFGS")$par
    # Convert optimum delta (above) to theta (constrained).
    theta[m, ] <- d2t(delta)
    all.loss[m, ] <- t(plloss(theta=theta[m, ], y=y, f=f.hat))
  }
  return(list=list(coefs=coefs, f.hat=f.hat, theta=theta, loss=all.loss))
}
```

```
1
2 ### Functions used in P/O Boosting
3
4 d2t <- function(delta) {
5   # Used for constraining the threshold estimates to be nondecreasing
6   #
7   # Args:
8   # delta: the vector of unconstrained threshold values
9   #
10  # Returns:
11  # Theta, the vector of constrained threshold values
12  delta[1] + cumsum(c(0, exp(delta[-1])))
13 }
14
15 plloss <- function(theta, y, f, w = w) {
16  # Calculates the value of the loss (negative logL)
17  #
18  # Args:
19  # theta: the vector of constrained threshold values
20  # y: the response vector
21  # f: the current function estimate
22  # w: the vector of weights
23  #
24  # Returns:
25  # The value of the loss function at the current step
26  if (length(f) == 1) f <- rep(f, length(y))
27  tmp <- lapply(1:(length(theta) + 1), function(i) {
```

```
28    if (i == 1) return(1 + exp(f - theta[i]))
29    if (i == (length(theta) + 1)) {
30        return(1 - 1 / (1 + exp(f - theta[i - 1])))
31    }
32    return(1 / (1 + exp(f - theta[i])) -
33             1 / (1 + exp(f - theta[i - 1])))
34  })
35  loss <- log(tmp[[1]]) * (y == levels(y)[1])
36  for (i in 2:nlevels(y)) {
37    loss <- loss - log(tmp[[i]]) * (y == levels(y)[i])
38  }
39  return(loss)
40 }
41
42 riskS <- function(delta, y, fit, w = w) {
43  # Calculates the value of the empirical risk.
44  # Converts delta to theta (constrained), then sums Loss over i=1, ..., n.
45  #
46  # Args:
47  # delta: the vector of unconstrained threshold values
48  # y: the response vector
49  # fit: the current function estimate
50  # w: the vector of weights
51  #
52  # Returns:
53  # The value of the risk function at the current step
54  sum(w * plloss(y = y, f = fit, theta = d2t(delta)))
```

```
}

neg.grad <- function(y, f, theta = theta, w = w) {
  # Calculates the negative gradient at current estimate of f and theta
  #
  # Args:
  # y: the response vector
  # f: the current function estimate
  # theta: the vector of constrained threshold values
  # w: the vector of weights
  #
  # Returns:
  # The value of the risk function at the current step
  if (length(f) == 1) f <- rep(f, length(y))
  # Calculates negative gradient for each subject, then sums over i=1...n
  ng <- sapply(1:(length(theta) + 1), function(i) {
    if (i > 1 & i < (length(theta) + 1)) {
      ret <- (1 - exp(2 * f - theta[i - 1] - theta[i])) /
            (1 + exp(f - theta[i - 1]) +
                exp(f - theta[i]) +
                exp(2 * f - theta[i - 1] - theta[i]))
    } else {
        if (i == 1) {
          ret <- -1 / (1 + exp(theta[i] - f))
        } else {
          ret <- 1 / (1 + exp(f - theta[i - 1]))
        }
```

```
82     }
83     return(ret * (y == levels(y)[i]))
84   })
85   rowSums(ng)
86  }
87
88 #Empirical risk function, treat theta as fixed, optimize over f - used to
       initialize f
89 # risk <- function(y, f, w = w)
90 # sum(w * plloss(y = y, f = f, theta = theta))
91
92 #Initialize delta / theta and f (maybe just initialize f to 0)
93 #initial.f <- function(y, w = w) {
94 # delta<-log(cumsum(pi.0) / (1 - cumsum(pi.0)))[1:(K - 1)]
95 # optimize(risk, interval = c(-5, 5), y = y, w = w)$minimum
96 # }
97
98 response <- function(f, theta) {
99    # Calculates posterior probabilities
100   #
101   # Args:
102   # f: the current function estimate
103   #
104   # Returns:
105   # Vector of posterior probabilities (of length K)
106   ret <- sapply(1:(length(theta) + 1), function(i) {
107     if (i == 1) return(1 / (1 + exp(f - theta[i]))) #P(Y=1|X)
```

```
    if (i == (length(theta) + 1)) {
      return(1 - 1 / (1 + exp(f - theta[i - 1]))) #P(Y=K|X)
    }
    return(1 / (1 + exp(f - theta[i])) - #P(Y=2or3or...orK-1|X)
       1 / (1 + exp(f - theta[i - 1])))
    })
    ret
   }
```

```
### Select stopping iteration by cross validation

cv.modelselect <- function(x, y, parallel=TRUE, num.folds=5, seed=2468,
                           mstop.seq=floor(seq(from=10, to=100, by=10))) {
  set.seed(seed)
  folds <- createFolds(y=y, k=num.folds, list=F)
  if (parallel) {
    `%fun%` <- `%dopar%`
  } else {
    `%fun%` <- `%do%`
  }
    modelselect <- foreach (m=mstop.seq, .combine=rbind) %fun% {
      dxy <- rep(0, num.folds)
      for (i in 1:num.folds) {
        fit <- ordinalFANS(x=x[-which(folds == i), ], y=y[-which(folds == i)
           ],
                           newx=x[which(folds == i), ], niter=1, mstop=m)
        dxy[i] <- rcorr.cens(as.numeric(fit$pred.class),
                             y[which(folds == i)])[2]
```

```
    }
    CV.estimate <- sum((table(folds) / length(y)) * dxy)
    dxy.sd <- sqrt(sum((table(folds) / length(y)) * (dxy - mean(dxy))^2))
    return(c(m, CV.estimate, dxy.sd))
  }
 colnames(modelselect) <- c("mstop", "CV Somer's Dxy Estimate", "SE")
 return(modelselect)
}
```

```
### Extract Model Coefficients

coef.ordinalFANS <- function(object, model.select="mean", m=NULL,
                             only.nonzero=FALSE) {
 if (object$niter > 1) {
   if (is.null(m)) {
     m <- dim(object$coefs[[1]])[1]
   }
   if (model.select=="mean") {
       beta <- object$mean.coefs[m, ]
       theta <- object$mean.theta[m, ]
       if (only.nonzero) {
         beta <- object$mean.coefs[m, ][object$mean.coefs[m, ]!=0]
       }
   } else if (is.numeric(model.select)) {
       beta <- object$coefs[[model.select]][m, ]
       theta <- object$theta[[model.select]][m, ]
       if (only.nonzero) {
         beta <- beta[beta!=0]
```

```
      theta <- theta[theta!=0]
    }
  } else if (model.select=="all") {
    beta <- lapply(object$coefs, function(x) {x[m, ]})
    theta <- lapply(object$theta, function(x) {x[m, ]})
    if (only.nonzero) {
      beta <- lapply(beta, function(x) {x[x!=0]})
      theta <- lapply(theta, function(x) {x[x!=0]})
    }
  }
} else {
  if (is.null(m)) {
    m <- dim(object$coefs)[1]
  }
  beta <- object$coefs[m, ]
  theta <- object$theta[m, ]
  if (only.nonzero) {
    beta <- beta[beta!=0]
  }
  }
  c(theta, beta)
}
```

```
### Plot the model output

plot.ordinalFANS <- function(object, type="coefficients", xlab=NULL,
                             ylab=NULL, main=NULL) {
  if (is.null(xlab)) xlab="Step"
```

```
6  if (is.null(ylab)) {
7    if (type == "coefficients") {
8      ylab <- expression(hat(beta))
9      if (is.null(main)) {
10      main <- "Coefficient Path"
11     }
12   } else if (type == "risk") {
13     ylab <- "Empirical Risk"
14     if (is.null(main)) {
15       main <- "Empirical Risk Path"
16     }
17   } else if (type=="var.importance") {
18     ylab <- "Variable Importance"
19     if (is.null(main)) {
20       main <- "Variable Importance Path"
21     }
22   }
23 }
24 if (type=="coefficients") {
25   if (object$niter > 1) {
26     mean.beta <- object$mean.coefs
27   } else {
28     mean.beta <- object$coefs
29   }
30   last.row <- nrow(mean.beta)
31   y.positions <- mean.beta[last.row, ][which(mean.beta[last.row, ] != 0)]
32   varorder <- names(y.positions)
```

```
33    matplot(mean.beta, type="l", xlab=xlab, ylab=ylab, main=main)
34    axis(side=4, labels=varorder, at=y.positions, cex.axis=0.69)
35  } else if (type=="risk") {
36      if (object$niter > 1) {
37        mean.risk <- object$mean.risk
38      } else {
39        mean.risk <- object$risk
40      }
41      plot(mean.risk, type="b", xlab=xlab, ylab=ylab, main=main)
42  } else if (type=="var.importance") {
43      if (object$niter > 1) {
44        mean.importance <- object$mean.var.importance
45      } else {
46        mean.importance <- object$var.importance
47      }
48      last.row <- nrow(mean.importance)
49      y.positions <- mean.importance[last.row, ][
50                      which(mean.importance[last.row, ] != 0)]
51      varorder <- names(y.positions)
52      matplot(mean.importance, type="l", xlab=xlab, ylab=ylab, main=main)
53      axis(side=4, labels=varorder, at=y.positions, cex.axis=0.69)
54    }
55 }
```

```
1 ### Predict the outcome of a new observation
2
3 predict.ordinalFANS <- function(object, newx, mstop=NULL) {
4   newx <- predict(object$trans, newx) # Scale newx with mean and sd of X
```

```
5
6  if (is.null(mstop)) mstop <- object$mstop
7  if (!is.null(object$newx)) {
8    if (object$niter > 1) {
9      if (!is.null(object$mean.posteriors) &
10         all.equal(as.matrix(object$newx), as.matrix(newx))) {
11       return(list=list(mean.posteriors=object$mean.posteriors,
12              pred.class=object$pred.class))
13     }
14   } else {
15     if (!is.null(object$posteriors) &
16         all.equal(as.matrix(object$newx), as.matrix(newx))) {
17       return(list=list(posteriors=object$posteriors,
18              pred.class=object$pred.class))
19     }
20   }
21 }
22 K <- object$K
23 niter <- object$niter
24
25 niter.preds <- foreach(l = 1:niter, .combine='rbind',
26                  .packages=c('sm', 'caret')) %do% {
27   if (object$niter > 1) {
28     D.train.x <- object$D.train.x[[l]]
29     D.train.y <- object$D.train.y[[l]]
30     coefs <- object$coefs[[l]]
31     theta <- object$theta[[l]]
```

```
32      aug.X <- object$aug.X[[1]]
33    } else {
34      D.train.x <- object$D.train.x
35      D.train.y <- object$D.train.y
36      coefs <- object$coefs
37      theta <- object$theta
38      aug.X <- object$aug.X
39    }
40    test.marginals.f <- array(dim=c(dim(newx)[1], K - 1, dim(newx)[2]))
41    test.marginals.g <- array(dim=c(dim(newx)[1], K - 1, dim(newx)[2]))
42    for (k in 1:(K - 1)) {
43      for (j in 1:dim(newx)[2]) {
44        test.marginals.f[, k, j] <- sm.density(D.train.x[D.train.y <= k, j],
45                                 eval.points=newx[, j],
46                                 display="none")$estimate
47        test.marginals.g[, k, j] <- sm.density(D.train.x[D.train.y > k, j],
48                                 eval.points=newx[, j],
49                                 display="none")$estimate
50      }
51    }
52    # Winsorization to improve stability of estimates as suggested in FANS
53    # manuscript
54    test.marginals.g[test.marginals.g < 0.001] <-
55    test.marginals.f[test.marginals.f < 0.001] <- 0.001
56    # Array of augmented features for newx. Within the array: p matrices of
57    # augmented features of dimension n x (K - 1)
58    Z.test <- log(test.marginals.f / test.marginals.g)
```

```
59    full.test <- data.frame(Z.test)
60    colnames(full.test) <- paste("z", paste(rep(1:dim(Z.test)[3], each=K -
          1),
61                                            rep(1:(K - 1),
62                                            times=dim(test.marginals.f)[3]),
63                                            sep="."),
64                                sep="")
65    # function estimates for newx
66    offset <- as.numeric(-1 * (matrix(apply(aug.X, 2, mean), nrow=1) %*%
67                         matrix(coefs[mstop, ], ncol=1)))
68    f.newx <- offset + (as.matrix(full.test) %*% as.matrix(coefs[mstop, ]))
69    # estimated posterior probabilities for newx
70    post.probs <- response(f.newx, theta[mstop, ])
71    return(list=list(post.probs))
72  }
73  if (object$niter > 1) {
74    if (ncol(as.matrix(niter.preds[[1]])) == 1) {
75      mean.posteriors <- Reduce('+', niter.preds) / niter
76      pred.class <- factor(levels(object$y)[apply(t(as.matrix(mean.
            posteriors)), 1, which.max)],
77                           levels=levels(object$y), ordered=TRUE)
78      posteriors <- lapply(1:niter, function(w) t(as.matrix(niter.preds))[[w
            ]])
79    } else {
80      mean.posteriors <- Reduce('+', niter.preds) / niter
81      pred.class <- factor(levels(object$y)[apply(mean.posteriors, 1, which.
            max)],
```

```
                        levels=levels(object$y), ordered=TRUE)
    posteriors <- lapply(1:niter, function(w) niter.preds[[w]])
  }
  return(list=list(posteriors=posteriors,
                   mean.posteriors=mean.posteriors, pred.class=pred.class))
} else {
  if (ncol(as.matrix(niter.preds[[1]])) == 1) {
    posteriors <- t(as.matrix(niter.preds[[1]]))
  } else {
    posteriors <- as.matrix(niter.preds[[1]])
  }
  pred.class <- factor(levels(object$y)[apply(posteriors, 1, which.max)],
                       levels=levels(object$y), ordered=TRUE)
  return(list=list(posteriors=posteriors, pred.class=pred.class))
}
}
```

```
### Does the same thing as predict()

fitted.ordinalFANS <- function(object, newx, mstop=NULL) {
  predict.ordinalFANS(object=object, newx=newx, mstop=mstop)
}
```

```
### Print the model output

print.ordinalFANS <- function(object, ...) {
  cat("Call:\n")
  print(object$call)
```

```
  cat("\nNumber of boosting iterations: mstop = ", object$mstop)
  cat("\nNumber of FANS iterations: niter = ", object$niter)
  cat("\nStep size: ", object$eps, "\n")
}

### Summarize the fitted model

summary.ordinalFANS <- function(object, m=NULL) {
  if (object$L > 1) {
    if (is.null(m)) {
      m <- dim(object$coefs[[1]])[1]
    }
    rownames(object$mean.var.importance) <- NULL
    var.importance <- object$mean.var.importance[m,
                                  which(object$mean.var.importance[m, ]!=0)]
  } else {
      if (is.null(m)) {
        m <- dim(object$coefs)[1]
      }
      rownames(object$var.importance) <- NULL
      var.importance <- object$var.importance[m,
                                       which(object$var.importance[m, ]!=0)]
  }
  return(list(object=object, var.importance=var.importance))
}
```

Appendix B

CODE FOR CHAPTER 3

B.1 Extended phase of the AML DREAM Challenge analysis

```
### Select clinical predictors to include in the
### no penalty subset

rm(list=ls())
#setwd("C:\\Users\\Kyle\\Dropbox\\Dissertation\\Aim 3\\AML\\Data")
#Fit univariate coxPH models for each predictor using continuous response
library(survival)
load("AML.RData")



### Logistic regression approach from Allison
aml.train$y<-as.numeric(aml.train$y)
aml.train$censor<-as.numeric(aml.train$censor)

X<-clinical
classes<-c()
for(i in 1:ncol(X)) {
  classes[i]<-class(X[,i])[1]
}
```

```
22
23
24 yij<-matrix(nrow=dim(X)[1],ncol=3)
25 for(i in 1:dim(X)[1]) {
26   for(j in 1:3) {
27     yij[i,j]<-ifelse(aml.train$y[i]==j &
28                      aml.train$censor[i]==1,1,0)
29   }
30 }
31
32 newX<-matrix(nrow=1,ncol=dim(X)[2])
33 for(i in 1:dim(X)[1]) {
34   for(j in 1:dim(X)[2]) {
35     if(class(X[,j])=="factor") X[,j]<-as.character(X[,j])
36   }
37   new.rows<-matrix(rep(X[i,],aml.train$y[i])
38                    nrow=aml.train$y[i],ncol=dim(X)[2],
39                    byrow=TRUE)
40   newX<-rbind(newX,new.rows)
41 }
42 newX<-newX[-1,]
43 newX.mat<-newX
44 newX<-data.frame(newX)
45 colnames(newX)<-colnames(clinical)
46 newX<-apply(newX,2,as.numeric)
47
48 new.y<-numeric()
```

```
for(i in 1:dim(X)[1]) {
  indiv<-c(rep(0,aml.train$y[i]-1),aml.train$censor[i])
  new.y<-c(new.y,indiv)
}

#Univariate models
glm.univFit<-function(pred) {
      data<-data.frame(new.y,unlist(newX[,pred]))
      colnames(data)<-c("y","x")
      form<-as.formula(paste("y","x",sep=" ~ "))
      fit<-summary(glm(form,data=data,family=binomial))
      return(coef(fit)[2,4])
}
p.values<-c()
j<-0
#p-values for numeric variables or factors with only 2 levels
for(i in 1:ncol(clinical)) {
  j<-j+1
  p.values[j]<-glm.univFit(i)
}

#p-values with variable names
vars1<-colnames(clinical)[c(1:ncol(clinical))]
p.vals<-data.frame(vars1,p.values)

#significant at alpha=0.1
lt.10 <-p.vals[which(p.vals[,2]<0.1),]
```

```
lt.10[order(lt.10[,2]), ]
# vars1 p.values
# 20 HGB 0.0005222901
# 9 ITD 0.0027507695
# 27 CD13 0.0054240144
# 8 cyto.cat 0.0125850298
# 11 Ras.Stat 0.0285030529
# 2 Age.at.Dx 0.0325419998
# 26 FIBRINOGEN 0.0419517112
# 4 PRIOR.MAL 0.0532993564
# 10 D835 0.0664101054
# 5 PRIOR.CHEMO 0.0767917172
# 30 CD7 0.0889254091
# 7 Infection 0.0924734909


#significant at alpha=0.05
lt.05 <- p.vals[which(p.vals[,2]<0.05),]
lt.05[order(lt.05[,2]), ]
# vars1 p.values
# 20 HGB 0.0005222901
# 9 ITD 0.0027507695
# 27 CD13 0.0054240144
# 8 cyto.cat 0.0125850298
# 11 Ras.Stat 0.0285030529
```

```
# 2 Age.at.Dx 0.0325419998
# 26 FIBRINOGEN 0.0419517112

#Combine results of univariate models and literature search:
#choose as unpenalized subset: Age, cytogenetics, ITD (FLT3), HGB
```

```
#N-fold cross validation (CV) code

library(doMC)
library(survival)
library(foreach)
library(Hmisc)

### READ IN DATA ###
load("AML.RData")
univ.sig <- c("Age.at.Dx", "cyto.cat", "ITD", "HGB")
y <- as.numeric(as.character(aml.train$y))
AML <- data.frame(y=Surv(y, aml.train$censor), protein, clinical)
X <- AML[, setdiff(colnames(AML), univ.sig)][, -1] #penalized predictors
unpen <- AML[, univ.sig] #unpenalized predictors

source("..//discSurvFCR.R")
registerDoMC(cores=10)

NfoldCV.1 <- foreach(i=1:nrow(AML), .combine='rbind') %dopar% {
  print(i)
  fit.1 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB,
                              data=AML[-i, ], x=X[-i, ], scale=TRUE,
```

```
                            epsilon=0.001, tol=1e-04, assumption=1)
  pred.AIC.1 <- predict.forwardCR(fit.1, newx=X[i, ], neww=AML[i, univ.sig],
                                  model.select="AIC")$class #minAIC model
  pred.BIC.1 <- predict.forwardCR(fit.1, newx=X[i, ], neww=AML[i, univ.sig],
                                  model.select="BIC")$class #minBIC model
  return(c(i,pred.AIC.1, pred.BIC.1))
}


somer.aic.1 <- rcorr.cens(NfoldCV.1[, 2], AML[, 1])[2:3] # Somer's Dxy (AIC)
somer.bic.1 <- rcorr.cens(NfoldCV.1[, 3], AML[, 1])[2:3] # Somer's Dxy (BIC)
somer.1 <- data.frame(assumption=c(1, 1),
                      somer=c(somer.aic.1[1], somer.bic.1[1]),
                      std.err=c(somer.aic.1[2], somer.bic.1[2]),
                      model.selection=c("AIC", "BIC"))

#
    ##############################################################################


registerDoMC(cores=10)

NfoldCV.2 <- foreach(i=1:nrow(AML), .combine='rbind') %dopar% {
  print(i)
  fit.2 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB,
                              data=AML[-i, ], x=X[-i, ], scale=TRUE,
                              epsilon=0.001, tol=1e-04, assumption=2)
```

```
48   pred.AIC.2 <- predict.forwardCR(fit.2, newx=X[i, ], neww=AML[i, univ.sig],
49                                   model.select="AIC")$class #minAIC model
50   pred.BIC.2 <- predict.forwardCR(fit.2, newx=X[i, ], neww=AML[i, univ.sig],
51                                   model.select="BIC")$class #minBIC model
52   return(c(i,pred.AIC.2, pred.BIC.2))
53 }
54
55
56 somer.aic.2 <- rcorr.cens(NfoldCV.2[, 2], AML[, 1])[2:3] # Somer's Dxy (AIC)
57 somer.bic.2 <- rcorr.cens(NfoldCV.2[, 3], AML[, 1])[2:3] # Somer's Dxy (BIC)
58 somer.2 <- data.frame(assumption=c(2, 2),
59                       somer=c(somer.aic.2[1], somer.bic.2[1]),
60                       std.err=c(somer.aic.2[2], somer.bic.2[2]),
61                       model.selection=c("AIC", "BIC"))
62
63 #
     ##############################################################################


64
65 registerDoMC(cores=10)
66
67 NfoldCV.3 <- foreach(i=1:nrow(AML), .combine='rbind') %dopar% {
68   print(i)
69   fit.3 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB,
70                               data=AML[-i, ], x=X[-i, ], scale=TRUE,
71                               epsilon=0.001, tol=1e-04, assumption=3)
72   pred.AIC.3 <- predict.forwardCR(fit.3, newx=X[i, ], neww=AML[i, univ.sig],
```

```
                              model.select="AIC")$class #minAIC model
  pred.BIC.3 <- predict.forwardCR(fit.3, newx=X[i, ], neww=AML[i, univ.sig],
                              model.select="BIC")$class #minBIC model
  return(c(i,pred.AIC.3, pred.BIC.3))
}


somer.aic.3 <- rcorr.cens(NfoldCV.3[, 2], AML[, 1])[2:3] # Somer's Dxy (AIC)
somer.bic.3 <- rcorr.cens(NfoldCV.3[, 3], AML[, 1])[2:3] # Somer's Dxy (BIC)
somer.3 <- data.frame(assumption=c(3, 3),
                    somer=c(somer.aic.3[1], somer.bic.3[1]),
                    std.err=c(somer.aic.3[2], somer.bic.3[2]),
                    model.selection=c("AIC", "BIC"))

#
    ##############################################################################


registerDoMC(cores=10)

NfoldCV.4 <- foreach(i=1:nrow(AML), .combine='rbind') %dopar% {
  print(i)
  fit.4 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB,
                            data=AML[-i, ], x=X[-i, ], scale=TRUE,
                            epsilon=0.001, tol=1e-04, assumption=1)
  pred.AIC.4 <- predict.forwardCR(fit.4, newx=X[i, ], neww=AML[i, univ.sig],
                              model.select="AIC")$class #minAIC model
```

```
  pred.BIC.4 <- predict.forwardCR(fit.4, newx=X[i, ], neww=AML[i, univ.sig],
                                  model.select="BIC")$class #minBIC model
  return(c(i,pred.AIC.4, pred.BIC.4))
}


somer.aic.4 <- rcorr.cens(NfoldCV.4[, 2], AML[, 1])[2:3] # Somer's Dxy (AIC)
somer.bic.4 <- rcorr.cens(NfoldCV.4[, 3], AML[, 1])[2:3] # Somer's Dxy (BIC)
somer.4 <- data.frame(assumption=c(4, 4),
                      somer=c(somer.aic.4[1], somer.bic.4[1]),
                      std.err=c(somer.aic.4[2], somer.bic.4[2]),
                      model.selection=c("AIC", "BIC"))
to.plot <- rbind(somer.1, rbind(somer.2, rbind(somer.3, somer.4)))
to.plot$assumption <- as.factor(to.plot$assumption)
to.plot$model.selection <- as.factor(to.plot$model.selection)

save(list=c("NfoldCV.1", "somer.aic.1", "somer.bic.1",
            "NfoldCV.2", "somer.aic.2", "somer.bic.2",
            "NfoldCV.3", "somer.aic.3", "somer.bic.3",
            "NfoldCV.4", "somer.aic.4", "somer.bic.4",
            "AML", "to.plot"),
     file="AML_NfoldCV.RData")
```

```
### Fit a model with the entire dataset,
### examine which features are included

### READ IN DATA ###
library(survival)
```

```
load("AML.RData")
univ.sig <- c("Age.at.Dx", "cyto.cat", "ITD", "HGB")
y <- as.numeric(as.character(aml.train$y))
AML <- data.frame(y=Surv(y, aml.train$censor), protein, clinical)
X <- AML[, setdiff(colnames(AML), univ.sig)][, -1] #penalized predictors
unpen <- AML[, univ.sig] #unpenalized predictors

source("..//discSurvFCR.R")

fit.1 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB, data=AML,
                            x=X, scale=TRUE, epsilon=0.001, tol=1e-04,
                            assumption=1)
fit.2 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB, data=AML,
                            x=X, scale=TRUE, epsilon=0.001, tol=1e-04,
                            assumption=2)
fit.3 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB, data=AML,
                            x=X, scale=TRUE, epsilon=0.001, tol=1e-04,
                            assumption=3)
fit.4 <- forwardcr.stepwise(y ~ Age.at.Dx + cyto.cat + ITD + HGB, data=AML,
                            x=X, scale=TRUE, epsilon=0.001, tol=1e-04,
                            assumption=4)


sAIC.1 <- which.min(fit.1$AIC)
sBIC.1 <- which.min(fit.1$BIC)
sAIC.1
sBIC.1
```

```
beta.nonzero.AIC.1 <- fit.1$beta[sAIC.1, which(fit.1$beta[sAIC.1, ] != 0)]
beta.nonzero.BIC.1 <- fit.1$beta[sBIC.1, which(fit.1$beta[sBIC.1, ] != 0)]
nonzero.AIC.1 <- names(beta.nonzero.AIC.1)
nonzero.BIC.1 <- names(beta.nonzero.BIC.1)

sAIC.2 <- which.min(fit.2$AIC)
sBIC.2 <- which.min(fit.2$BIC)
sAIC.2
sBIC.2
beta.nonzero.AIC.2 <- fit.2$beta[sAIC.2, which(fit.2$beta[sAIC.2, ] != 0)]
beta.nonzero.BIC.2 <- fit.2$beta[sBIC.2, which(fit.2$beta[sBIC.2, ] != 0)]
nonzero.AIC.2 <- names(beta.nonzero.AIC.2)
nonzero.BIC.2 <- names(beta.nonzero.BIC.2)

sAIC.3 <- which.min(fit.3$AIC)
sBIC.3 <- which.min(fit.3$BIC)
sAIC.3
sBIC.3
beta.nonzero.AIC.3 <- fit.3$beta[sAIC.3, which(fit.3$beta[sAIC.3, ] != 0)]
beta.nonzero.BIC.3 <- fit.3$beta[sBIC.3, which(fit.3$beta[sBIC.3, ] != 0)]
nonzero.AIC.3 <- names(beta.nonzero.AIC.3)
nonzero.BIC.3 <- names(beta.nonzero.BIC.3)

sAIC.4 <- which.min(fit.4$AIC)
sBIC.4 <- which.min(fit.4$BIC)
sAIC.4
sBIC.4
```

```
beta.nonzero.AIC.4 <- fit.4$beta[sAIC.4, which(fit.4$beta[sAIC.4, ] != 0)]
beta.nonzero.BIC.4 <- fit.4$beta[sBIC.4, which(fit.4$beta[sBIC.4, ] != 0)]
nonzero.AIC.4 <- names(beta.nonzero.AIC.4)
nonzero.BIC.4 <- names(beta.nonzero.BIC.4)

data.frame(fit.1$AIC[sAIC.1], fit.2$AIC[sAIC.2], fit.3$AIC[sAIC.3],
          fit.4$AIC[sAIC.4])
data.frame(fit.1$BIC[sBIC.1], fit.2$BIC[sBIC.2], fit.3$BIC[sBIC.3],
          fit.4$BIC[sBIC.4])

save.image("AMLfull.RData")
```

REFERENCES

[1] W Fraser Symmans et al. "Measurement of residual breast cancer burden to predict survival after neoadjuvant chemotherapy". In: *Journal of Clinical Oncology* 25.28 (2007), pp. 4414–4422.

[2] Ian Ayres. *Super Crunchers: Why Thinking-By-Numbers is the New Way to be Smart*. Bantam, 2008.

[3] Otto Metzger Filho, Michail Ignatiadis, and Christos Sotiriou. "Genomic Grade Index: An important tool for assessing breast cancer tumor grade and prognosis". In: *Critical Reviews in Oncology/Hematology* 77.1 (2011), pp. 20–29.

[4] Khawaja Afzal Ammar et al. "Prevalence and prognostic significance of heart failure stages application of the American College of Cardiology/American Heart Association heart failure staging criteria in the community". In: *Circulation* 115.12 (2007), pp. 1563–1570.

[5] Reisa A Sperling et al. "Toward defining the preclinical stages of Alzheimers Disease: recommendations from the National Institute on Aging-Alzheimer's Association workgroups on diagnostic guidelines for Alzheimer's Disease". In: *Alzheimer's & Dementia* 7.3 (2011), pp. 280–292.

[6] Andrew S Levey et al. "National Kidney Foundation practice guidelines for chronic kidney disease: evaluation, classification, and stratification". In: *Annals of Internal Medicine* 139.2 (2003), pp. 137–147.

[7] Kellie J Archer et al. "ordinalgmifs: An R package for ordinal regression in high-dimensional data settings". In: *Cancer Informatics* 13 (2014), p. 187.

[8] John W Pratt. "Concavity of the log likelihood". In: *Journal of the American Statistical Association* 76.373 (1981), pp. 103–106.

[9] J Burridge. "A note on maximum likelihood estimation for regression models using grouped data". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1981), pp. 41–45.

[10] Sorin Drăghici. *Statistics and Data Analysis for Microarrays Using R and Bioconductor*. CRC Press, 2011.

[11] Danh V Nguyen et al. "DNA microarray experiments: biological and technological aspects". In: *Biometrics* 58.4 (2002), pp. 701–717.

[12] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.

[13] Trevor Hastie et al. "Forward stagewise regression and the monotone lasso". In: *Electronic Journal of Statistics* 1 (2007), pp. 1–29.

[14] Peter Bühlmann and Torsten Hothorn. "Boosting algorithms: regularization, prediction and model fitting". In: *Statistical Science* (2007), pp. 477–505.

[15] Trevor Hastie, Rob Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc.; 2001.

[16] Matthias Schmid et al. "Geoadditive regression modeling of stream biological condition". In: *Environmental and Ecological Statistics* 18.4 (2011), pp. 709–733.

[17] Roger Newson. "Confidence intervals for rank statistics: Somers' D and extensions". In: *Stata Journal* 6.3 (2006), p. 309.

[18] Lars Kai Hansen and Peter Salamon. "Neural network ensembles". In: *IEEE transactions on pattern analysis and machine intelligence* 12 (1990), pp. 993–1001.

[19] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[20] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140.

[21] L Breiman. *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth International Group, 1984.

[22] Kevin J Cherkauer. "Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks". In: *Working notes of the AAAI workshop on integrating multiple learned models*. Citeseer. 1996, pp. 15–21.

[23] Thomas G. Dietterich and Ghulum Bakiri. "Solving multiclass learning problems via error-correcting output codes". In: *Journal of Artificial Intelligence Research* 2 (1995), pp. 263–286.

[24] Leo Breiman. "Random forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[25] Jianqing Fan et al. "Feature Augmentation via Nonparametrics and Selection (FANS) in high-dimensional classification". In: *Journal of the American Statistical Association* 111.513 (2016), pp. 275–287.

[26] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2014. URL: `http://www.R-project.org/`.

[27] Avrum Spira et al. "Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer". In: *Nature Medicine* 13.3 (2007), pp. 361–366.

[28] Gerhard Tutz and Klaus Hechenbichler. "Aggregating classifiers with ordinal response structure". In: *Journal of Statistical Computation and Simulation* 75.5 (2005), pp. 391–408.

[29] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of Statistics* (2001), pp. 1189–1232.

[30] Benjamin Hofner et al. "Model-based boosting in R: a hands-on tutorial using the R package mboost". In: *Computational Statistics* 29.1-2 (2014), pp. 3–35.

[31] Christine Desmedt et al. "Strong time dependence of the 76-gene prognostic signature for node-negative breast cancer patients in the TRANSBIG multicenter independent validation series". In: *Clinical Cancer Research* 13.11 (2007), pp. 3207–3214.

[32] Ron Edgar, Michael Domrachev, and Alex E Lash. "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository". In: *Nucleic Acids Research* 30.1 (2002), pp. 207–210.

[33] Rafael A Irizarry et al. "Summaries of Affymetrix GeneChip probe level data". In: *Nucleic Acids Research* 31.4 (2003), e15–e15.

[34] Kellie J Archer and Sarah E Reese. "Detection call algorithms for high-throughput gene expression microarray data". In: *Briefings in Bioinformatics* (2009), bbp055.

[35] Yoav Benjamini and Daniel Yekutieli. "The control of the false discovery rate in multiple testing under dependency". In: *Annals of Statistics* (2001), pp. 1165–1188.

[36] Xiaochun Li. *ALL: A data package*. R package version 1.14.0. 2009.

[37] Yali Zhai et al. "Gene expression analysis of preinvasive and invasive cervical squamous cell carcinomas identifies HOXC10 as a key mediator of invasion". In: *Cancer research* 67.21 (2007), pp. 10163–10172.

[38] Jeanette N McClintick and Howard J Edenberg. "Effects of filtering by present call on analysis of microarray experiments". In: *BMC Bioinformatics* 7.1 (2006), p. 1.

[39] Lin-Jing Yuan et al. "SPAG5 upregulation predicts poor prognosis in cervical cancer patients and alters sensitivity to taxol treatment via the mTOR signaling pathway". In: *Cell Death & Disease* 5.5 (2014), e1247.

[40] De Jun Sun et al. "Endothelin-3 growth factor levels decreased in cervical cancer compared with normal cervical epithelial cells". In: *Human Pathology* 38.7 (2007), pp. 1047–1056.

[41] Martin Koch and Michael Wiese. "Gene expression signatures of angiocidin and darapladib treatment connect to therapy options in cervical cancer". In: *Journal of Cancer Research and Clinical Oncology* 139.2 (2013), pp. 259–267.

[42] Klaus Schliep and Klaus Hechenbichler. *kknn: Weighted k-nearest neighbors*. R package version 1.0. 2010.

[43] National Cancer Institute. *SEER Cancer Statistics Factsheets: Cervix Uteri Cancer. National Cancer Institute. Bethesda, MD.* `http://seer.cancer.gov/statfacts/html/cervix.html`. Accessed: 2016-08-30.

[44] Virginia A Moyer. "Screening for cervical cancer: US Preventive Services Task Force recommendation statement". In: *Annals of Internal Medicine* 156.12 (2012), pp. 880–891.

[45] National Cancer Institute. *SEER Cancer Statistics Factsheets: Melanoma of the Skin. National Cancer Institute. Bethesda, MD.* `http://seer.cancer.gov/statfacts/html/melan.html`. Accessed: 2016-08-30.

[46] Darrell S Rigel, Julie Russak, and Robert Friedman. "The evolution of melanoma diagnosis: 25 years beyond the ABCDs". In: *CA: A Cancer Journal for Clinicians* 60.5 (2010), pp. 301–316.

[47] Agnessa Gadeliya Goodson and Douglas Grossman. "Strategies for early melanoma detection: approaches to the patient with nevi". In: *Journal of the American Academy of Dermatology* 60.5 (2009), pp. 719–735.

[48] Dmitri Talantov et al. "Novel genes associated with malignant melanoma but not benign melanocytic lesions". In: *Clinical Cancer Research* 11.20 (2005), pp. 7234–7242.

[49] Arun J Sanyal, Seung Kew Yoon, and Riccardo Lencioni. "The etiology of hepatocellular carcinoma and consequences for treatment". In: *The Oncologist* 15.Supplement 4 (2010), pp. 14–22.

[50] Kellie J Archer et al. "High-throughput assessment of CpG site methylation for distinguishing between HCV-cirrhosis and HCV-associated hepatocellular carcinoma". In: *Molecular Genetics and Genomics* 283.4 (2010), pp. 341–349.

[51] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), pp. 67–82.

[52] R Dummer, A Hauschild, and G Pentheroudakis. "Cutaneous malignant melanoma: ESMO clinical recommendations for diagnosis, treatment and follow-up". In: *Annals of Oncology* 20.suppl 4 (2009), pp. iv129–iv131.

[53] Guido Reifenberger et al. "Molecular characterization of long-term survivors of glioblastoma using genome-and transcriptome-wide profiling". In: *International Journal of Cancer* 135.8 (2014), pp. 1822–1831.

[54] US Department of Health and Human Services et al. *Guidance regarding methods for de-identification of protected health information in accordance with the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule*. 2015.

[55] Robin Henderson, Margaret Jones, and Janez Stare. "Accuracy of point predictions in survival analysis". In: *Statistics in Medicine* 20.20 (2001), pp. 3083–3096.

[56] C Murray Parkes. "Accuracy of predictions of survival in later stages of cancer". In: *British Medical Journal* 2.5804 (1972), p. 29.

[57] Enrico A Colosimo, Liciana VAS Chalita, and Clarice GB Demétrio. "Tests of proportional hazards and proportional odds models for grouped survival data". In: *Biometrics* 56.4 (2000), pp. 1233–1240.

[58] David R Cox. "Regression models and life-tables". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1972), pp. 187–220.

[59] David R Cox. "Partial likelihood". In: *Biometrika* 62.2 (1975), pp. 269–276.

[60] Ross L Prentice and Lynn A Gloeckler. "Regression analysis of grouped survival data with application to breast cancer data". In: *Biometrics* (1978), pp. 57–67.

[61] John P Klein and Melvin L Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer Science & Business Media, 2003.

[62] Jianguo Sun. "Regression analysis of interval-censored failure time data". In: *Statistics in Medicine* 16.5 (1997), pp. 497–504.

[63] Bob Lowenberg, James R Downing, and Alan Burnett. "Acute myeloid leukemia". In: *New England Journal of Medicine* 341.14 (1999), pp. 1051–1062.

[64] National Cancer Institute. *SEER Cancer Statistics Factsheets: Acute Myeloid Leukemia*. `http://seer.cancer.gov/statfacts/html/amyl.html`. Accessed: 2015-12-02. 2015.

[65] Elihu Estey and Hartmut Döhner. "Acute myeloid leukaemia". In: *The Lancet* 368.9550 (2006), pp. 1894–1907.

[66] David Noren et al. "DREAM 9: An Acute Myeloid Leukemia Prediction Big Data Challenge". In: *ICBO*. 2014, pp. 108–109.

[67] Paul D Allison. "Discrete-time methods for the analysis of event histories". In: *Sociological Methodology* 13.1 (1982), pp. 61–98.

[68] Judith D Singer and John B Willett. "Its about time: Using discrete-time survival analysis to study duration and the timing of events". In: *Journal of Educational and Behavioral Statistics* 18.2 (1993), pp. 155–195.

[69] Janine Austin Clayton. "Studying both sexes: a guiding principle for biomedicine". In: *The FASEB Journal* (2015), fj–15.

[70] Kyle Ferber and Kellie J Archer. "Modeling discrete survival time using genomic feature data". In: *Cancer Informatics* 14.Suppl 2 (2015), pp. 37–43.

[71] Amanda Elswick Gentry et al. "Penalized ordinal regression methods for predicting stage of cancer in high-dimensional covariate spaces". In: *Cancer Informatics* 14 (Suppl 2) (2015), p. 201.

[72] KJ Archer and AAA Williams. "L1 penalized continuation ratio models for ordinal response prediction using high-dimensional datasets". In: *Statistics in Medicine* 31.14 (2012), pp. 1464–1474.

[73] WA Thompson Jr. "On the treatment of grouped observations in life studies". In: *Biometrics* (1977), pp. 463–470.

[74] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013.

[75] Olga Troyanskaya et al. "Missing value estimation methods for DNA microarrays". In: *Bioinformatics* 17.6 (2001), pp. 520–525.

[76] P Morel et al. "Cytogenetic analysis has strong independent prognostic value in de novo myelodysplastic syndromes and can be incorporated in a new scoring system: a report on 408 cases." In: *Leukemia* 7.9 (1993), pp. 1315–1323.

[77] Panagiotis D Kottaridis et al. "The presence of a FLT3 internal tandem duplication in patients with acute myeloid leukemia (AML) adds important prognostic information to cytogenetic risk group and response to the first cycle of chemotherapy: analysis of 854 patients from the United Kingdom Medical Research Council AML 10 and 12 trials". In: *Blood* 98.6 (2001), pp. 1752–1759.

[78] Luis Villela and Javier Bolaños-Meade. "Acute Myeloid Leukaemia". In: *Drugs* 71.12 (2011), pp. 1537–1550.

[79] Susan P Whitman et al. "Absence of the wild-type allele predicts poor prognosis in adult de novo acute myeloid leukemia with normal cytogenetics and the internal tandem duplication of FLT3: A Cancer and Leukemia Group B Study". In: *Cancer Research* 61.19 (2001), pp. 7233–7239.

[80] RW Craig. "MCL1 provides a window on the role of the BCL2 family in cell proliferation, differentiation and tumorigenesis." In: *Leukemia* 16.4 (2002), pp. 444–454.

[81] Stefan P Glaser et al. "Anti-apoptotic Mcl-1 is essential for the development and sustained growth of acute myeloid leukemia". In: *Genes & Development* 26.2 (2012), pp. 120–125.

[82] Adrian W Bowman and Adelchi Azzalini. *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Vol. 18. OUP Oxford, 1997.

VITA

Kyle Ferber was born on July 2, 1990 in Virginia Beach, VA. He received his Bachelor of Science degree in May of 2012 from the College of William and Mary in Williamsburg, VA. He began work towards a PhD in Biostatistics at Virginia Commonwealth University in August of 2012. While pursuing his PhD, he has worked as a Teaching Assistant (August 2012 - June 2013), a Statistical Intern for PharPoint Research, Inc. (June 2013 - June 2014), a Research Assistant under Dr. Kellie J. Archer (June 2014 - April 2016), and as a Consulting Biostatistician for ImmunArray, Ltd. (April 2016 - November 2016).