Bayesian Ordinal Models

Nengjun Yi

This vignette describes Bayesian ordinal regression models for analyzing ordinal outcomes (e.g. disease severity or multi-level drug response) and the quasi-Newton algorithm, and then the methods for assessing the fitted model and its predictive values. Finally, it explains how to set up and fit Bayesian ordinal models using the `bpolr` function in the **BhGLM** package.

Bayesian ordinal regression models

Let y_i be the ordinal outcome and x_{ij} the value of the jth predictor for the ith individual, where $i = 1, \cdots, n$ and $j = 1, \cdots, J$. The property of ordinal outcomes is that there exists a clear ordering of the response categories, but no underlying interval scale between them. Thus, it does not make sense to treat ordinal responses as numeric values. For notational convenience, we code the ordinal outcome as the integers 1, 2, $\cdots$, K, with K being the number of categories. The commonly used method for analyzing the ordinal outcome is the ordinal logistic regression:

$$\Pr(y_i = k) = \begin{cases} 1 - \text{logit}^{-1}(X_i\beta - c_1) & \text{if } k = 1 \\ \text{logit}^{-1}(X_i\beta - c_{k-1}) - \text{logit}^{-1}(X_i\beta - c_k) & \text{if } 1 < k < K \\ \text{logit}^{-1}(X_i\beta - c_{K-1}) & \text{if } k = K \end{cases} \quad (1)$$

where $\beta = (\beta_1, \cdots, \beta_J)^{\mathrm{T}}$ is a vector of the coefficients, and the parameters c_k are the cut-points or thresholds and are constrained to increase, because the probabilities defined in (1) are non-negative. The transformation *logit* can be replaced by other transformations, leading to other ordinal regressions, e.g. probit, loglog, cloglog, and cauchit, as specified by the argument `method` in the `bpolr` function.

The ordinal regressions can be fitted by the maximum likelihood procedure as implemented by the function `polr` in the R core package **MASS**. However, the classical ordinal regressions are not appropriate for jointly analyzing a large number of and/or highly-correlated predictor variables, due to the problems of non-identifiability and overfitting.

Bayesian ordinal models specify priors on the parameters, which can constraint the coefficients in reasonable ranges and thus allows the model to be reliably fitted and to identify important predictors. For specifying appropriate priors for all the coefficients, it is important to transform all variables to have a common scale. The function `covariates` in **BhGLM** can be used to transform both continuous and categorical predictors.

The function `bpolr` employs uniform priors on the cut-points c_k, and Student-t priors on the coefficients β_j:

$$\beta_j \sim t_\nu(\mu_j, s_j^2) \quad (2)$$

where the preset scale parameter s_j controls the shrinkage on the coefficient β_j; smaller scale s_j induces stronger shrinkage on β_j. The Student-t distribution includes normal (e.g. $\nu = +\infty$) and

Cauchy (e.g. v = 1) distributions as special cases. Different scale values s_j can be specified for different predictors. This is very useful for incorporating prior information about the predictors into the analysis. For example, we can set a large scale for some relevant predictors and a small scale for others. If we have no prior information, we use a common scale s for all predictors. We recommend fit models with several scales, for example, s = 0.03, 0.05, 0.1, 0.5, 1, 2, and then choose an optimal model from them.

The quasi-Newton algorithm

The function `bpolr` employs a quasi-Newton algorithm, called BFGS, to fit the above Bayesian ordinal models by finding the posterior modes of the parameters (β, c), i.e. estimating the parameters by maximizing the posterior density. We implement the BFGS algorithm by altering the commonly used function `polr` in the R package **`MASS`**, which fits classical ordinal regressions using the quasi-Newton algorithm.

Summarizing the fitted model and inferences

The quasi-Newton algorithm returns the estimates of the coefficients and their standard deviations. The p-values for testing the hypotheses H_0: $\beta_j = 0$ can be calculated similarly as the classical framework. The `summary.bh` function in **`BhGLM`** provides the estimates of the coefficients, the standard deviations, and the p-values. The `plot.bh` function graphically displays these estimates.

Evaluating the fitted model and the predictive performance

After fitting the model, we obtain the estimate $(\hat{\beta}, \hat{c})$ and then can estimate the probabilities: $p_{ik} = \Pr(y_i = k \mid X_i\hat{\beta}, \hat{c}),\ i = 1, \cdots, n;\ k = 1, \cdots, K$. Denote y_{ik} = I(y_i = k) as the binary indicator response for the k-th category. With the estimated probabilities p_{ik}, we can evaluate the fitted model using several measures.

The function `predict.bh` returns several measures:

(1) *Deviance*: $d = -2\sum_{i=1}^{n} \log p_{ik}$. Deviance measures the overall quality of a fitted model;

(2) *AUC* (area under the ROC curve). We can calculate AUC for the k-th category using y_{ik} and p_{ik}, i = 1, ⋯, n, as usual. Then the AUC for all the categories is defined as $\frac{1}{K}\sum_{k=1}^{K} AUC_k$;

(3) *MSE* (mean squared error). MSE is defined as $MSE = \frac{1}{K}\sum_{k=1}^{K}\left[\frac{1}{n}\sum_{i=1}^{n}(y_{ik} - p_{ik})^2\right]$;

(4) *Misclassification*. The misclassification is defined as: $MIS = \frac{1}{K}\sum_{k=1}^{K}\left[\frac{1}{n}\sum_{i=1}^{n} I(|\, y_{ik} - p_{ik} \,| > 0.5)\right]$, where $I(|\, y_{ik} - p_{ik} \,| > 0.5) = 1$ if $|\, y_{ik} - p_{ik} \,| > 0.5$, and $I(|\, y_{ik} - p_{ik} \,| > 0.5) = 0$ if $|\, y_{ik} - p_{ik} \,| \le 0.5$.

The package `BhGLM` provides two ways to evaluate the predictive performance of the model. If we have a training data set and a validation data set, we can use the training data to fit a Bayesian

model and then evaluate the predictive performance on the validation data (treated as new data) using the function `predict.bh`. If we have only one data set, we can perform K-fold cross-validation using the function `cv.bh`, which returning all the measures described above.

Examples

We first use the functions `sim.x` and `sim.y` to simulate data, and then explain how to use `bpolr` to analyze Bayesian ordinal models. We simulate 1000 individuals, 200 predictors, five non-zero coefficients, and a three-level ordinal outcome.

```
library(BhGLM)

set.seed(1234)
N = 1000
K = 200
# simulate K correlated variables
x = sim.x (n=N, m=K, corr=0.6)
# assign 5 non-zero coefficients, and all others are zero
h = rep(0.1, 5)
nz = as.integer(seq(5, K, by=K/length(h)))
# simulate responses
yy = sim.y (x=x[, nz], mu=10, herit=h, p.neg=0.5, sigma=1.6, quantiles=c(0.3, 0.6))

# show the non-zero coefficients
yy$coefs

#         x5         x45         x85        x125        x165
# 0.7247270 -0.7323255  0.7387685 -0.7297424  0.7287910

y = as.factor(yy$y.ordinal)
table(y)
# y
#   0   1   2
# 300 300 400
```

Conventional ordinal regression models

We first analyze the simulated data using conventional ordinal logistic regression with the function `polr` in **`MASS`**:

```
library(MASS)

f1 = polr(y ~ ., data = x, method = "logistic", Hess = T)
```

We can summarize the fitted model using the function `summary.bh`:

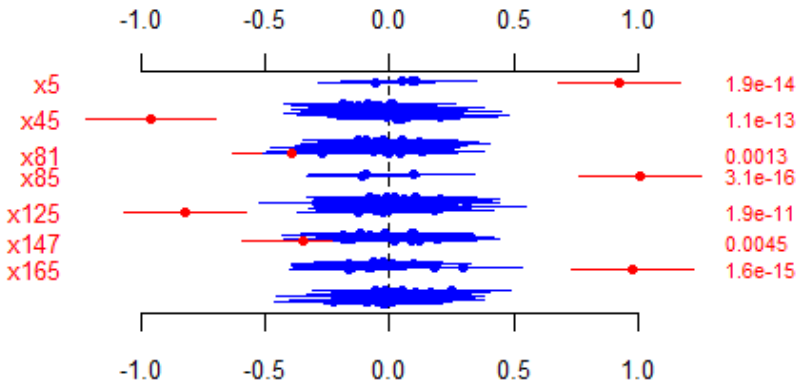
```
out = summary.bh (f1, digits = 3)
```

```
head(out)

#       coef se(coef) exp(coef) lower.95 upper.95   pvalue
# x1   0.103    0.122     1.109    0.869    1.414 3.96e-01
# x2   0.089    0.119     1.094    0.863    1.386 4.51e-01
# x3   0.050    0.122     1.052    0.824    1.342 6.78e-01
# x4  -0.054    0.116     0.948    0.751    1.195 6.43e-01
# x5   0.924    0.121     2.518    1.978    3.205 1.94e-14
# x6   0.005    0.123     1.005    0.786    1.284 9.68e-01
```

We graphically display the fitted model using the function `plot.bh`:

```
plot.bh (f1, threshold = 0.01, gap = 50, col.pts = c("red", "blue") )
```



Bayesian ordinal regression models

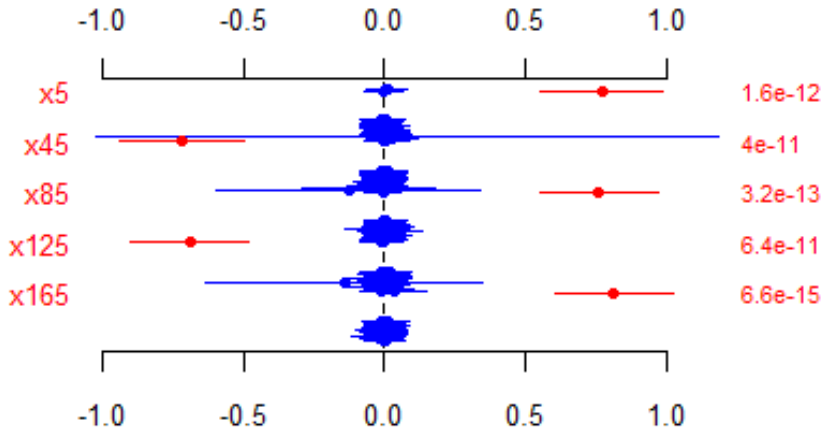
We then analyze the simulated data using Bayesian ordinal logistic regression with Cauchy prior, which is special case of Student-t prior (i.e. `prior.df = 1`). If the prior scale is set to be infinite, the Bayesian model is equivalent to the conventional model. Instead, we set the prior scale to be small:

```
f2 = bpolr (y ~ ., data = x, method = "logistic", Hess = T, prior.scale = 0.05)

out = summary.bh (f2, digits = 3)
head(out)

#       coef se(coef) exp(coef) lower.95 upper.95   pvalue
# x1   0.011    0.036     1.011    0.942    1.086 7.54e-01
# x2   0.010    0.035     1.010    0.941    1.083 7.85e-01
# x3   0.009    0.035     1.009    0.941    1.082 8.04e-01
# x4  -0.004    0.034     0.996    0.930    1.065 8.96e-01
# x5   0.772    0.109     2.163    1.739    2.691 1.62e-12
# x6   0.006    0.034     1.006    0.939    1.077 8.69e-01
```

plot.bh (f2, threshold = 0.01, gap = 50, col.pts = c("red", "blue"))



It can be seen that Bayesian analysis shrinks the estimates of zero-coefficients close to zero.

Assessing the fitted models and the predictive values

We can use the function `predict.bh` to calculate measures for assessing the fitted model:

predict.bh(f1)$measures

```
# deviance                auc                mse misclassification
# 1684.629              0.785              0.166              0.246
```

predict.bh(f2)$measures

```
# deviance                auc                mse misclassification
# 1819.739              0.733              0.181              0.267
```

If we have new data new.x and new.y, we can use predict.bh(f1, new.x, new.y)$measures to evaluate the predictive performance of the model f1. We also can use cross-validation to assess the predictive performance:

cv.bh(f1, nfolds=10, ncv=1)$measures

```
# deviance                auc                mse misclassification
# 2356.179              0.631              0.225              0.320
```

cv.bh(f2, nfolds=10, ncv=1)$measures

```
# deviance                auc                mse misclassification
# 1900.467              0.702              0.190              0.278
```

The cross-validation can be used to choose optimal prior scale and to compare different models.