The \texttt{oprobit\_predict} package, version 1.3

Allin Cottrell

December 29, 2018

1 Introduction

This package computes outcome probabilities and predictions based on results from an ordered response model (probit or logit). To set the scene we begin with a brief review of such models. (See section 35.2 of the \textit{Gretl User’s Guide} for more details.)

Let $y$ be an ordered response variable that can take on any of the $J+1$ values $0, 1, 2, \ldots, J$. We suppose that underlying the observed response is a latent variable,

$$y^* = X\beta + \varepsilon = z + \varepsilon$$

(1)

Now define “cut points”, $\alpha_1 < \alpha_2 < \cdots < \alpha_J$, such that

\begin{align*}
  y &= 0 \text{ if } y^* \leq \alpha_1 \\
  y &= 1 \text{ if } \alpha_1 < y^* \leq \alpha_2 \\
  & \vdots \\
  y &= J \text{ if } y^* > \alpha_J
\end{align*}

(2)

For example, if the response takes on three values there will be two such cut points, $\alpha_1$ and $\alpha_2$.

The probability that individual $i$ exhibits response $j$, conditional on the characteristics $x_i$, is then given by

\begin{align*}
  P(y_i = j \mid x_i) &= \begin{cases} 
    P(y^* \leq \alpha_1 \mid x_i) = F(\alpha_1 - z_i) & \text{for } j = 0 \\
    P(\alpha_j < y^* \leq \alpha_{j+1} \mid x_i) = F(\alpha_{j+1} - z_i) - F(\alpha_j - z_i) & \text{for } 0 < j < J \\
    P(y^* > \alpha_J \mid x_i) = 1 - F(\alpha_J - z_i) & \text{for } j = J
  \end{cases}
\end{align*}

(3)

where $F(\cdot)$ denotes a cumulative distribution function: either the normal CDF (in the case of probit) or the logistic CDF (logit). The unknown parameters $\alpha_j$ are estimated jointly with the $\beta$ terms via maximum likelihood.

In \texttt{gretl} the \texttt{probit} or \texttt{logit} command can be used to estimate such a model. After estimation the accessor \texttt{$yhat} can be used to retrieve the estimates $X\hat{\beta} = \hat{z}$ (see equation 1). This package can then be used to compute the per-outcome probabilities as shown in (3), with inputs $\hat{z}$ (labeled \texttt{zhat} below) and the vector of cut-point estimates (labeled \texttt{cut} below). The latter can be obtained by selecting the appropriate trailing rows of the \texttt{$coeff} vector: if the specification includes $k$ independent variables then one wants

$$\text{matrix cut = $coeff[k+1:]$}$$

since $\texttt{$coeff}$ holds $\hat{\beta}$ followed by $\hat{\alpha}$.

2 Usage via script

Two main public functions are offered for scripting purposes; they each take two required arguments, namely a cut-point vector and a $\hat{z}$ series (plus optional trailing arguments as described below).
Note that for out-of-sample prediction one can supply a \( \hat{z} \) other than that obtained via the \$yhat\ accessor.

For example, suppose one has 500 observations and one wishes to estimate a model on the first 400 and obtain predictions for the remaining 100. Then after ordered estimation one can do

\[
\text{smpl 401 500} \\
\text{series zhat = lincomb(Xlist, $coeff[1:nelem(Xlist)])}
\]

where Xlist is the list of regressors in the specification.

### 2.1 Obtaining the matrix of probabilities

To obtain an outcome-probability matrix use the following function:

\[
\text{matrix ordered_Pmat (const matrix cut,} \\
\text{series zhat,} \\
\text{bool lgt[0])}
\]

It returns an \( n \times m \) matrix, where \( n \) is the number of observations and \( m \) is the number of distinct response values (that is, the number of cut points plus one). Row \( i \) of this matrix holds the estimated probability of each of the \( m \) responses at observation \( i \). Give a non-zero value for the third, optional argument to select the logit variant, the default being probit.

### 2.2 Obtaining point predictions

In some cases one may wish to ask: which particular outcome should be predicted for case \( i \) (integer-valued \( \hat{y}_i \)) conditional on the characteristics \( x_i \) along with \( \hat{\beta} \) and \( \hat{\alpha} \)? The package provides means of answering that question, but it is not as straightforward as it may seem.

The most “natural” way of arriving at a point prediction is, perhaps, to choose the outcome with the greatest estimated probability. We’ll call this the \text{pmax} method; it is easily implemented given the outcome-probability matrix, as in

\[
\text{matrix Pij = ordered_Pmat(cut, zhat)} \\
\text{matrix pmax = imaxr(Pij) - 1}
\]

The built-in \text{imaxr} function returns a column vector holding the row indices of the maximum values on each row of the input matrix, so in this case an \( n \)-vector with elements in the range 1 to \( J + 1 \) (using the notation of section 1, and where \( n \) is the length of \text{zhat}). We subtract 1 to get the zero-based value of the maximum-probability outcome—again, as per section 1, but see also section 2.3 below.

However, there are (at least) two other ways of arriving at a point prediction.

- Under the assumption that \( E(\varepsilon|X) = 0 \), \( E(\hat{z}) \) should equal \( z \) and one could assign a prediction based on where \( \hat{z} \) falls in relation to the estimated cut points. That is, predict \( y_i = 0 \) when \( \hat{z}_i \leq \alpha_1 \), \( y_i = 1 \) when \( \alpha_1 < \hat{z}_i \leq \alpha_2 \), and so on—see (2). Call this the \text{cuts} method.

- Use all the probability information for case \( i \) by forming an expected value, as in

\[
E(y_i|x_i) = \sum_{j=0}^{J} \hat{P}(y_i = j | x_i) \cdot j
\]

which one could then round to the nearest integer. Call this the \text{expectation} method.

These three methods may well yield the same specific prediction in many cases but they don’t have to; they are not logically equivalent and in fact can diverge quite substantially, with the extent of the divergence depending on, among other things, the variance of the error term, \( \varepsilon \). So is one of the methods clearly better than the others? It depends on the criterion for success in prediction (or its flipside, the loss function). Here are our findings in brief (see the Appendix for more details):
To maximize the number of correct predictions, use the \texttt{pmax} method.

To minimize the sum of absolute prediction errors, use the \texttt{cuts} method.

To minimize the number of cases where the prediction differs from the observed value by more than one step, use the \texttt{expectation} method.

All that said, here is the signature of the function for obtaining point predictions.

\begin{verbatim}
series ordered_prediction (const matrix cut, 
    series zhat, 
    int ymin[0], 
    bool lgt[0], 
    int ptype[1:3:1])
\end{verbatim}

The return value is a series containing a predicted response at each observation based on \texttt{cut} and \texttt{zhat}. The optional arguments have the following effects:

- \texttt{ymin} can be used to adjust the minimum value of the response variable. By default this is taken to be zero, but if (for example) you give \texttt{ymin = 3} the responses are labeled 3, 4, 5, ...

- \texttt{lgt} can be used to specify that the logit estimator should be assumed (the default being probit).

- \texttt{ptype} can be used to specify a prediction method (1 = \texttt{pmax}, the default; 2 = \texttt{cuts}; 3 = \texttt{expectation}).

In addition, for purposes of comparison, the package offers a variant on \texttt{ordered_prediction()} which returns a matrix containing predictions from all three methods, in the order indicated above. The signature of this function is as follows,

\begin{verbatim}
matrix all_ordered_predictions (const matrix cut, 
    series zhat, 
    int ymin[0], 
    bool lgt[0])
\end{verbatim}

and the returned matrix is $n \times 3$, where $n$ is the number of values of \texttt{zhat} in the current sample range.

\section*{2.3 More on ymin}

It may be helpful to further explain the \texttt{ymin} argument to \texttt{ordered_prediction}. In the theoretical presentation of the ordered-outcome model in section \[\text{(square brackets represents section number)}\] we took the minimum value of $y$ to be zero, by construction, and this is also what gretl does internally. In some applications, however, the actual $y$ data may have a positive minimum. For example, suppose our dependent variable is a rating on a scale of 0 to 10 (in principle) but in our sample there are no values below 3 and none higher than 8. In effect, therefore, we have six levels of $y$ (the integers 3 to 8) which will be recoded internally as 0 to 5.

If we were to generate predictions of $y$ on the 0 to 5 scale and check them for equality with observed $y$ to find the number of correct predictions, we’d likely get a spurious answer of none correct. For comparability we either have to subtract 3 from observed $y$ or add 3 to the predictions. The \texttt{ymin} argument lets you do the latter.

\section*{2.4 Compatibility functions}

For the sake of backward compatibility with the previous release of this package (version 1.2, dated 2016-01-26), the following two functions are included under their original names.

\begin{verbatim}
matrix oprobit_Pmat (const matrix cut, series zhat)
\end{verbatim}
This is a variant of `ordered_Pmat` with the choice of probit hard-wired.

```plaintext
series oprobit_pred (const matrix cut, series zhat)
```

This is a variant of `ordered_prediction` with its optional arguments hard-wired as `ymin = 0, lgt = 0` (probit estimates assumed) and `ptype = 1` (pmax predictions).

### 3 GUI access

On installing the `oprobit_predict` package you should get the choice of letting it attach to the Analysis menu in gretl model windows under the item `Outcome probabilities`. In that case when you estimate an ordered model via the graphical interface you can view the matrix of per-outcome probabilities by clicking on this item. The output also identifies the outcome with the greatest individual probability.
Appendix

We present here some background for the statements made above concerning the relative merits of three prediction methods for ordered models (see section 2.2).

We ran several Monte Carlo exercises (differing with respect to the variance of the error term and the relative frequency of occurrence of the discrete $y$ values) and obtained similar results on all of them, at least with regard to the ordering of the three prediction methods on each of the four criteria considered, which were:

1. Proportion of cases correctly predicted.
2. Proportion of cases where the prediction misses actual by more than one step.
3. Mean Absolute Error (MAE) of the prediction.
4. Mean Squared Error (MSE) of the prediction.

With a dependent variable that is not on an interval scale one might have doubts about the applicability of MAE and MSE (although it’s presumably “worse” even on an ordinal scale for the prediction to miss the actual value by more steps); that’s why we included the relatively robust criterion 2.

The table below shows typical results for a simulation with a single normally distributed independent variable and a normally distributed error, where $y$ is discretized into 4 levels, with a sample size of 200 at each iteration and 10,000 iterations. The error variance was calibrated to give disagreement between the methods in about 5 percent of cases. The figures of merit in the table are based solely on the (102,559) cases where the three methods were not unanimous, and they are normalized to a value of 1.0 for best performance on each row (that is, the greatest value for “Correct predictions” and the least value for the other criteria).

```
Method        1    2    3
Correct predictions  1.000 0.984 0.909
Cases off by more than one  6.802 3.995 1.000
Mean Absolute Error  1.024 1.000 1.027
Mean Squared Error     1.147 1.052 1.000
```

Methods: 1 = \texttt{pmax}, 2 = \texttt{cuts}, 3 = \texttt{expectation}

Not surprisingly, when the simulation is calibrated to produce fewer cases of disagreement the results tend to become more uniform (e.g. the counts of correct predictions for methods 2 and 3 get closer to that of method 1). But in all our simulations the rank-order of the methods on each criterion was invariant.

So it’s noteworthy that while the \texttt{pmax} method tends to give the greatest number of correct predictions, it also tends to give the greatest number of substantially wrong predictions (off by more than one). Seriously wrong predictions are minimized by the \texttt{expectation} method, while the \texttt{cuts} method seems to occupy the middle ground, with the virtue (if it be such) of tending to minimize the MAE.

The hansl script which generated the above results is shown on the following page. Note that with a large number of Monte Carlo iterations some defensive programming is required: we need to handle cases where the estimator fails to converge, and we also exclude cases datasets where the dependent variable takes on fewer than its maximum of 4 levels.
# Monte Carlo script for comparing prediction methods

set verbose off
include oprobit_predict.gfn

nulldata 200
set seed 1544461131 # or change this

# artificial independent (latent) variable
series x = normal()

iters = 10000
K = iters * $nobs
scalar N = 0
scalar m = 0

# allow sloppier than usual BFGS convergence criterion
set bfgs_maxgrad 8.0

# 0.2 * K should be more than enough rows
matrix R = zeros(0.2*K, 3)

loop while N < K --quiet
    series z = x + normal()/1.6
    series y = z < -1.1 ? 0 : z < 0 ? 1 : z < 1.1 ? 2 : 3
    catch probit y x --quiet
    if $error
        nc = 0
    else
        c = $coeff[2:]
        nc = nelem(c)
    endif
    if nc == 3
        # successful estimation with 3 cuts
        N += $T
        matrix P = all_ordered_predictions(c, $yhat, 0, 0)
        matrix d = selifr(abs(y .- P), sel)
        nd = rows(d)
        if nd > 0
            R[m+1:m+nd,] = d
            m += nd
        endif
    endif
endloop

R = R[1:m,] # restrict R to the m non-unanous cases
printf "Differing predictions: %d out of %d (%.1f%%)
In cases of differing predictions --
count of correct predictions:
%8.3f", r * 1/r[imaxr(r)]
printf "count of predictions off by more than one:
%8.3f", r * 1/r[iminr(r)]
printf "sum of absolute errors:
%8.3f", r * 1/r[iminr(r)]
printf "sum of squared errors:
%8.3f", r * 1/r[iminr(r)]