# Pset3

March 2, 2017

## 1 Table of Contents



## 2 Question 1

(25pt + 10pt) In this question, we use the file `brader.csv` which contains data from Brader, Valentino and Suhay (2008). The file includes the following variables for $n = 265$ observations:

- the outcome of interest – a four-point scale in response to Do you think the number of immigrants from foreign countries should be increased or decreased?
- tone of the story treatment (positive or negative)
- ethnicity of the featured immigrant treatment (Mexican or Russian)
- respondents' age
- respondents' income

Consider the following ordered logit model for an ordered outcome variable with four levels:

$$\Pr(Y_i \leq j \mid X_i) \;=\; \frac{\exp(\psi_j - X_i^\top \beta)}{1 + \exp(\psi_j - X_i^\top \beta)}$$

for $j = 1, 2, 3, 4$ and $i = 1, ..., n$ where $\psi_4 = \infty$ and $X_i = [\texttt{tone}_i \; \texttt{eth}_i \; \texttt{ppage}_i \; \texttt{ppincimp}_i]^\top$ (i.e. no intercept).

**a) (5pt)** Write down the likelihood function.

To simplify the notation, the indexs $i, j, k$ are used consistently, $i$ to iterate over the number of observations,$j$ to iterate over the number of outcomes, and $k$ * to iterate over the predictors.*

The log-likelihood can be easily seen to be:

$$l = \prod_{i=1}^{n} \frac{\exp(\psi_{Y_i} - X_i^\top \beta)}{1 + \exp(\psi_{Y_i} - X_i^\top \beta)} - \frac{\exp(\psi_{Y_i - 1} - X_i^\top \beta)}{1 + \exp(\psi_{Y_i - 1} - X_i^\top \beta)}$$

To simplify, from here on $\phi$ will be the inverse link function and $\phi'$ its derivative. This allows us to write the above expression in a more Matricial form as:

$$\prod_{i=1}^{n} \sum_{j=1}^{4} \tilde{M}_{i,j} \phi \left( \psi_j - \sum_{k=1}^{m} X_{ik} \beta_k \right)$$

where

$$\tilde{M} = MK = M \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

And $M$ is the OneHot matrix for the $Y_i$, i.e. a matrix that has $n$ rows, and each row is the row vector with zeros everywhere but at the position $j$, where the the observed $Y$ is $Y_j$.

In practice, however, we are interested in the log-likelihood, which is:

$$L = \sum_{i=1}^{n} \log \sum_{j=1}^{4} \tilde{M}_{i,j} \phi \left( \psi_j - \sum_{k=1}^{m} X_{ik}\beta_k \right)$$

Warning: We will use a great deal of abstraction because the functions are a mess otherwise, here we go:

```
In [6]: log_likelihood = wrapper(
            function(X,Y,beta,psi,M,phi,dphi){
                sum(log(rowSums(M*phi)))
            }
        )
```

The wrapper function is just an abstaction (see it at the end of the file, in the annex), that computes the variables `M`, `phi`= $\phi\left(\psi_j - \sum_{k=1}^{m} X_{ik}\beta_k\right)$, `dphi`= $\phi'\left(\psi_j - \sum_{k=1}^{m} X_{ik}\beta_k\right)$. We use it because everything is easier in those variables. When a function goes trough the "wrapper", its arguments change to become (X,Y,beta,psi), and the wrapper computes M, phi and dphi.

b) (10pt) Derive the score functions for $\beta$ and $\psi_j$.

To simplify the computations, let $\phi_{ij} = \phi(\psi_j - X_i^\top \beta)$, and $\phi'_{ij} = \phi'(\psi_j - X_i^\top \beta)$. Then we will have:

$$L = \sum_{i=1}^{n} \log \sum_{j=1}^{4} \tilde{M}_{i,j} \phi_{i,j}$$

Since deriveatives and sums commute freely, we can compute the score easily:

$$\frac{\partial L}{\partial \beta_k} = -\sum_{i=1}^{n} \left( \sum_{j=0}^{4} \tilde{M}_{i,j}\phi_{i,j} \right)^{-1} \left( \sum_{j=0}^{4} \tilde{M}_{i,j}\phi'_{i,j} \right) X_{ik}$$

in `R` that is:

```
In [7]: beta_score =  wrapper(
            function(X,Y,beta,psi,M,phi,dphi){
                -(rowSums(M*dphi)/rowSums(M*phi))%*%X
            }
        )
```

For the $\psi$, the result is equally immediate:

$$\frac{\partial L}{\partial \psi_j} = \sum_{i=1}^{n} \left( \sum_{j=0}^{4} \tilde{M}_{i,j}\phi_{i,j} \right)^{-1} \tilde{M}_{i,j}\phi'_{i,j}$$

which in `R` can be written as:

```
In [95]: psi_score =  wrapper(
             function(X,Y,beta,psi,M,phi,dphi){
                 r = (rowSums(M*phi)**-1%*%(M*dphi))
                 r[1:(length(r)-1)]
             }
         )
```

(10pt) Using (a) and (b), calculate the maximum likelihood estimates of $\beta$ and $\psi_j$ and their standard errors via the `optim` function in R. Confirm your results by comparing them to outputs from the `polr` function in the `MASS` package.

Load the data:

```
In [96]: brader = read.csv('Data/brader.csv')
```

Prepare a Handler function for `optim`

```
In [104]: likelihood_handler = function (x,data){
              X = data[[1]]
              Y = data[[2]]
              o = ncol(X)
              beta = x[1:o]
              psi = x[(o+1):length(x)]
              l = length(psi)
              diff_psi = psi[2:l]-psi[1:(l-1)]
              if(min(diff_psi)<=0)
                  return (1E5)
              return(-log_likelihood(X,Y,beta,psi))
          }
```

```
In [114]: gradient_handler = function(x,data){
              X = data[[1]]
              Y = data[[2]]
              o = ncol(X)
              beta = x[1:o]
              psi = x[(o+1):length(x)]
              l = length(psi)
              diff_psi = psi[2:l]-psi[1:(l-1)]
              if(min(diff_psi)<=0)
                  return(x*0);
              sbeta = (-beta_score(X,Y,beta,psi))
              spsi  = (-psi_score(X,Y,beta,psi))
              append(sbeta,spsi)
          }
```

Then we prepare the data as our function needs it

```
In [111]: Y = brader$immigr
          X = data.matrix(brader)[,2:5]
          data = list(X,Y)
          beta = c(0.5, 0, 0, 0)
          psi = c(-1, 0, 1)
          betapsi = append(beta,psi)
          betapsi
```

0.5

0

0

0

-1

0

1

```
In [112]: psi_score(X,Y,beta,psi)
```

-48.8333052835874

-6.56482749873738

-11.6548684369225

And we run the optimization

```
In [113]: r = optim(betapsi,likelihood_handler, gr = gradient_handler, data = data,
                    hessian = 1, control = (reltol = 1E-12))
          r$par[1:4]
          r$par[5:7]
          -log_likelihood(X,Y,r$par[1:4],r$par[5:7])
```

0.75057472585271

0.182514633025783

0.00954246157357417

0.00453801085115179

-1.64987523554466

0.153712692971585

1.37459144576628

325.94791194349

```
In [31]: library(MASS)

         plr <- polr( factor(immigr) ~ tone + eth + ppage + ppincimp,
                           data = brader, method = "logistic", Hess = 1)
         summary(plr)
         -log_likelihood(X,Y,plr$coefficients,plr$z)
```

```
Call:
polr(formula = factor(immigr) ~ tone + eth + ppage + ppincimp,
    data = brader, Hess = 1, method = "logistic")

Coefficients:
            Value Std. Error t value
tone     0.749446   0.230241  3.2550
eth      0.166225   0.227063  0.7321
ppage    0.009345   0.007131  1.3105
ppincimp 0.004149   0.029511  0.1406

Intercepts:
    Value   Std. Error t value
1|2 -1.6671  0.5664    -2.9434
2|3  0.1318  0.5401     0.2441
3|4  1.3535  0.5466     2.4761

Residual Deviance: 651.8892
AIC: 665.8892
```

325.944618379387

** d. (10pt) Bonus question. ** The standard ordered logit model is sometimes called the proportional odds model because it assumes the effect of $X_i$ is constant across levels on the odds ratio scale. One approach to relax this assumption is to allow the coefficients to vary across levels, i.e.,

$$\Pr(Y_i \leq j \mid X_i) \;=\; \frac{\exp(\psi_j - X_i^\top \beta_j)}{1 + \exp(\psi_j - X_i^\top \beta_j)}$$

for $j = 1, 2, 3, 4$ and $i = 1, ..., n$ where $\beta_4 = 0$ (i.e. the fourth group is a reference group), and $\psi_4 = \infty$. For this model, derive the likelihood and score functions, and use `optim` to obtain the maximum likelihood estimates of $\beta_j$ and $\psi_j$ and their standard errors for the `brader` data.

A direct modification of the above exercise, substituting $\beta_k$ for $\beta_{k,j}$ works: We will have:

$$L = \sum_{i=1}^{n} \log \sum_{j=1}^{4} \tilde{M}_{i,j} \phi \left( \psi_j - \sum_{k=1}^{m} X_{ik} \beta_{kj} \right)$$

therefore, since the formula is essentialy the same (and thanks to `R` not really caring much about the shape of the objects – which is nice!), the same `log_likelihood` coded above still works!

Since deriveatives and sums commute freely, we can compute the score easily:

$$\frac{\partial L}{\partial \beta_{kj}} = -\sum_{i=1}^{n} \left( \sum_{j=0}^{4} \tilde{M}_{i,j} \phi_{i,j} \right)^{-1} \tilde{M}_{i,j} \phi'_{i,j} X_{ik}$$

in this case we must modify the function, to create a new function:

```
In [69]: extra_beta_score = wrapper(
             function(X,Y,beta,psi,M,phi,dphi){
                 #this corresponds to a matrix that has the same elements in every
                 #row, and are the (sum **)^{-1} in the formula above
                 M1 = matrix(rep(rowSums(M*phi),ncol(M)),ncol=ncol(M),byrow=0)
                 -t(M*dphi/M1)%*%X
             }
         )
```

For the $\psi$, the result is equally immediate:

$$\frac{\partial L}{\partial \psi_j} = \sum_{i=1}^{n} \left( \sum_{j=0}^{4} \tilde{M}_{i,j} \phi_{i,j} \right)^{-1} \tilde{M}_{i,j} \phi'_{i,j}$$

As for the likelihood, the same function does the job

# 3 Question 4

Cross Validation for Polynomial Regression. (18 points) Consider the following four data generating processes:

- DGP 1: $Y = -2 * 1_{\{X<-3\}} + 2.55 * 1_{\{X>-2\}} - 2 * 1_{\{X>0\}} + 4 * 1_{\{X>2\}} - 1 * 1_{\{X>3\}} + \epsilon$
- DGP 2: $Y = 6 + 0.4X - 0.36X^2 + 0.005X^3 + \epsilon$
- DGP 3: \$Y = 2.83 * sin($\frac{\pi}{2} \times X) + \epsilon$\$$DGP4 : Y = 4 * \sin(3\pi \times X) * 1_{\{X>0\}} + \epsilon$

$X$ is drawn from the uniform distribution in [-4,4] and \$$\epsilon$\$$isdrawnfromastandardnormal(\mu = 0, \sigma^2 =$ 1). \begin{enumerate}

```
In [ ]: DGP1 = function(X){
            -2   *(X < -3)
            +2.55*(X > -2)
            -2   *(X >  0)
            +4   *(X >  2)
            -1   *(X >  3)}
        DGP2 = function(X){
            6+0.4*X-0.36*X^2+0.005*X^3
        }
        DGP3 = function(X){
            2.83*sin(pi/2*X)
        }
        DGP4 = function(X){
            4*sin(3*pi*x)*(X>0)
        }
        ERR = rnorm
        DGP = c(DGP1,DGP2,DGP3,DGP4)
```

(5 pts.) Write a function to estimate the generalization error of a polynomial by $k$-fold cross-validation. It should take as arguments the data, the degree of the polynomial, and the number of folds $k$. It should return the cross-validation mean squared error.

```
In [ ]: n = nrow(data);
        data = data[sample(n),]
        folds = cut(seq(1,data),breaks=N,labels=FALSE)
        for(i in 1:N)
        {
            ind = which(folds==i,arr.ind=TRUE)
            test = yourData[ind,]
            train = yourData[-ind,]
        }
```

```
In [ ]: s = matrix(c(1,2,3,4,1,2,3,4,1,2,3,4),nrow = 4,byrow = 1)
```

```
In [ ]: rowSums(s)
```

```
In [ ]: s[,3] = 1
```

# 4 Heavy lifting for 2.

This is where dirty things happen so that the inline code looks beautiful

This is the `wrapper` function used in exercise 2, it precomputes everything the equations need, and thus it greatly simplifies the calculations!

```
In [3]: wrapper = function(expression){
            function(X,Y,beta,psi,
                     phi = function(x){exp(x)/(1+exp(x))},
                     dphi = function(x){exp(x)/(1+exp(x))**2})
            {
                psi = append(psi,10); #add the 'infinity'
                m = length(psi);
                M = compute_M(Y,m);
                l = linear_combination(X,Y,beta,psi);

                 phix = phi(l);   phix[,m]=1; #impose the 'infinity'
```

```
            dphix = phi(l);  dphix[,m]=0; #impose the 'infinity'

             #this is where everything happens
            expression(X,Y,beta,psi,M,phix,dphix);
        }
    }
```

In order for the code to run, we need the `compute_M` function that creates the `M` matrix defined above:

```
In [4]: compute_M = function (Y,m){

            #This is just a One Hot encoder:
            n = length(Y)
            M1 = t(matrix(rep(c(1:m),n),m))
            M2 = matrix(rep(c(Y),m),n)
            M = (M1==M2)+0

            #Create the matrix K
            K = diag(m+1)
            K = -K[1:m,]+K[2:(m+1),]
            K = K[1:m,1:m+1]

            #Return the product
            M%*%K
        }
```

We also used - without declaration - a function "linear combination", that should return the linear combination inside the inverse link

```
In [13]: linear_combination = function(X,Y,beta,psi){
             m = length(psi) #possible results, from 1 to m.
             n = length(Y) #number of observatons
             if(TRUE){
                 beta2 = matrix(rep(beta,m),ncol=m,byrow=0)
             }
             else{
                 beta2 = beta
             }
             linear =  - (X%*%beta2)
             origin =  matrix(rep(psi,n) ,nrow = n, byrow = 1)
             origin +  linear
         }

In [ ]:
```