

Multinomial and Ordinal Regression Notes

Daniel J. Eck

Contents

Introduction to multinomial response models	1
Nominal responses	2
Interpretations	2
Estimation	2
Sabermetrics example	3
Acknowledgments	11

We now develop modeling for multinomial responses and ordinal categorical responses. These models arise as a natural extension of the generalized linear modeling framework that we have discussed throughout.

Introduction to multinomial response models

In these notes we will suppose that the response variable Y is categorical, with J possible categories. The response variable can be:

- nominal (for example: your favorite soft drink among Coke, Pepsi, other)
- ordinal (for example: customer rating of service among good, neutral, or bad)

Let categories be coded as $1, \dots, J$ (in order, if ordinal). In these notes we will model the distribution of Y conditional on $x \in \mathbb{R}^p$. Let

$$\pi_j(x) = \mathbb{P}(Y \text{ in category } j \mid x), \quad j = 1, \dots, J,$$

and note that the multinomial model has the constraint

$$\sum_{j=1}^J \pi_j(x) = 1, \quad \text{for any } x.$$

As usual, there are n observations of (Y, x) . Assume that observations of Y are independent (conditional on x). Let x_i be the explanatory vector for observation i . The response variable Y may be represented in alternative formats. We could use **ungrouped codes** where $y_i = j$, $j \in \{1, \dots, J\}$, but this is inconvenient for mathematical expressions. We can use **ungrouped indicator vectors**:

$$y_i = (y_{i1}, \dots, y_{iJ})$$

where

$$y_{ij} = \begin{cases} 1 & \text{if observation } i \text{ in category } j \\ 0 & \text{otherwise,} \end{cases}$$

where $\sum_{j=1}^J y_{ij} = 1$ for every i . In this ungrouped indicator vector formulation we have that

$$Y_i \sim \text{Multinomial}(1, \pi(x_i)),$$

where $\pi(x) = (\pi_1(x), \dots, \pi_J(x))$. We can also consider **grouped codes** where we assume observation i represents n_i independent responses having the same explanatory vector x_i (i.e., replicates). Let

$$y_i = (y_{i1}, \dots, y_{iJ})$$

with y_{ij} defined as the number of responses in category j (for observation i) so that $\sum_{j=1}^J y_{ij} = n_i$. Then,

$$Y_i \sim \text{Multinomial}(n_i, \pi(x_i)).$$

Nominal responses

We will now consider the baseline-category logistic model for multinomial response data:

$$\log \left(\frac{\pi_j(x)}{\pi_J(x)} \right) = x^T \beta_j, \quad j = 1, \dots, J-1.$$

These are the baseline (category) logits, with J as the baseline category. The $J-1$ values of vectors β_j are unrestricted parameters to be estimated (MLE), where we define $\beta_J = 0$. Logits for other pairs of categories can also be determined. For categories a and b we can write

$$\begin{aligned} \log \left(\frac{\pi_a(x)}{\pi_b(x)} \right) &= \log \left(\frac{\pi_a(x)}{\pi_J(x)} \right) - \log \left(\frac{\pi_b(x)}{\pi_J(x)} \right) \\ &= x^T \beta_a - x^T \beta_b \\ &= x^T (\beta_a - \beta_b) \end{aligned}$$

Note: Any choice of baseline category leads to a model of the same form (with linearly transformed parameters).

Interpretations

Let x_k be the k th component of x , and let x_{-k} be the vector x with the k th component removed. Also let β_{jk} be the k th component of β_j and let β_{j-k} be the vector β with the k th component removed. Then exponentiation gives

$$\frac{\pi_j(x)}{\pi_J(x)} = e^{x^T \beta_j}$$

as the odds of $Y = j$ **conditional** on $\{Y = j \text{ or } Y = J\}$. We call it the odds of j relative to J . Then,

$$e^{\beta_{jk} x_k}$$

is the multiplicative change in odds when x_k increases by one with all other components x_{-k} held fixed. The response probabilities are given as

$$\pi_j(x) = \frac{e^{x^T \beta_j}}{\sum_{h=1}^J e^{x^T \beta_h}}$$

where we recall that $\beta_J = 0$. Remark: unlike in the binomial case, $\pi_j(x)$ is not necessarily a monotone function of each component x_k .

Estimation

The log likelihood for this model is given by

$$\begin{aligned} l(\theta) &\propto \log \prod_{i=1}^n \prod_{j=1}^J \pi_j(x_i; \theta)^{y_{ij}} \\ &= \sum_{i=1}^n \left\{ \sum_{j=1}^{J-1} y_{ij} (\alpha_j + x_i^T \beta_j) - \log \left[1 + \sum_{j=1}^{J-1} \exp(\alpha_j + x_i^T \beta_j) \right] \right\} \end{aligned}$$

$$= \sum_{j=1}^{J-1} \left[\alpha_j \left(\sum_{i=1}^n y_{ij} \right) + \sum_{k=1}^p \beta_{jk} \left(\sum_{i=1}^n x_{ik} y_{ij} \right) \right] - \sum_{i=1}^n \log \left[1 + \sum_{j=1}^{J-1} \exp(\alpha_j + x_i^T \beta_j) \right]$$

where y_{ij} is from the ungrouped indicator or grouped format, intercept terms are made explicit, and β_J is a reference category level taken to be 0. From the above derivation we see that the multinomial response model has $\sum_{i=1}^n y_{ij}$ as the sufficient statistic for α_j ; this is the total number of outcomes in category j .

We consider MLEs $\hat{\beta}_1, \dots, \hat{\beta}_{J-1}$ when they uniquely exist. Note that the total number of (scalar) free parameters is $(J-1)p$ where p is the number of explanatory variables ($\beta_j \in \mathbb{R}^p$). Substituting the MLEs gives estimated response functions

$$\hat{\pi}_j(x), \quad (\text{for all } j),$$

the “fitted values”

$$\hat{\pi}_{ij} = \hat{\pi}_j(x), \quad (\text{for all } i \text{ and } j),$$

and the estimated logits, such as

$$\log \left(\frac{\hat{\pi}_j(x)}{\hat{\pi}_J(x)} \right) = x^T \hat{\beta}_j.$$

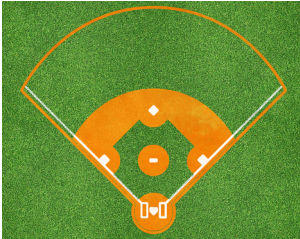
Many quantities are as they were for binary/binomial models: MLEs, saturated model, deviances, Wald and likelihood ratio tests and confidence intervals, AIC, G^2 , X^2 . Previously discussed optimization algorithms can be used for parameter estimation.

Sabermetrics example

In the game of baseball it is important to understand how the characteristics of a player’s swing translate to positive outcomes on the baseball field. We want to estimate the probabilities of different baseball outcomes given quality of swing and hitting tendency variables (obtained by STATCAST).

The outcomes of interest are: single (1 base), double (2 bases), triple (3 bases), home runs (4 bases), and outs. The more bases the better, 4 bases = 1 run. The team with the most runs wins the game. An out is a bad outcome for a batter, an out means the batter ended their time at the plate without reaching base.

The swing and batting tendency variables are: exit velocity (launch speed off the bat), launch angle (angle off the bat into the air), and spray angle (where on the field the ball is going).



```
library(VGAM) # has model-fitting functions
bball = read.csv("bball.csv")
bball$events = as.factor(bball$events)
```

We now fit two baseline category logistic models (nested), using a function from the VGAM package:

```
mod1 = vglm(events ~ launch_speed + launch_angle + spray_angle,
             family=multinomial, data=bball)
mod2 = vglm(events ~ launch_speed + launch_angle + spray_angle +
             I(launch_angle^2),
             family=multinomial, data=bball)
```

Why are we considering a quadratic term for launch angle?

```
llrts = deviance(mod1) - deviance(mod2)
llrts.df = df.residual(mod1) - df.residual(mod2)
```

```
llrts; llrts.df
```

```
## [1] 12213.38
```

```
## [1] 4
```

```
1 - pchisq(llrts, llrts.df)
```

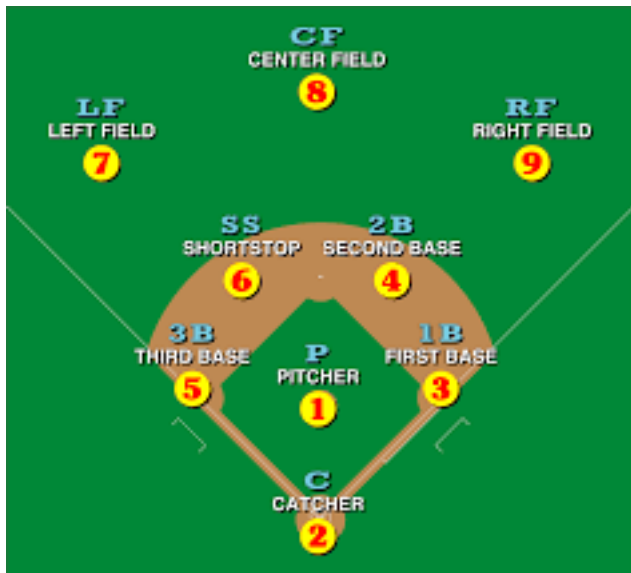
```
## [1] 0
```

```
AIC(mod1); AIC(mod2)
```

```
## [1] 83119.98
```

```
## [1] 70914.6
```

What type of polynomial should we consider for spray angle?



```
##      user  system elapsed
## 8.628    0.692    9.325
```

The polynomial model called mod3 is suppressed in the .Rmd file (echo = FALSE). We see that AIC and the LRT both favor the polynomial model for spray angle.

```
llrts = deviance(mod2) - deviance(mod3)
llrts.df = df.residual(mod2) - df.residual(mod3)
1 - pchisq(llrts, llrts.df)
```

```
## [1] 0
```

```
AIC(mod2); AIC(mod3)
```

```
## [1] 70914.6
```

```
## [1] 66678.96
```

We now consider some interaction terms.

```
system.time({mod4 = vglm(events ~ launch_speed +
  launch_angle + spray_angle + I(launch_angle^2) +
  I(spray_angle^2) + I(spray_angle^3) + I(spray_angle^4) +
  I(spray_angle^5) + I(spray_angle^6) +
  I(spray_angle*launch_angle) + I(spray_angle*launch_speed) +
  I(launch_angle*launch_speed),
  family=multinomial, data=bball)})
```

```
##      user  system elapsed
##    9.148   0.530   9.679
```

```
llrts = deviance(mod3) - deviance(mod4)
llrts.df = df.residual(mod3) - df.residual(mod4)
1 - pchisq(llrts, llrts.df)
```

```
## [1] 0
```

```
AIC(mod3); AIC(mod4)
```

```
## [1] 66678.96
```

```
## [1] 66325.19
```

We now obtain summary information for our final model.

```
system.time(print(summary(mod4)))
```

```
##
## Call:
## vglm(formula = events ~ launch_speed + launch_angle + spray_angle +
##      I(launch_angle^2) + I(spray_angle^2) + I(spray_angle^3) +
##      I(spray_angle^4) + I(spray_angle^5) + I(spray_angle^6) +
##      I(spray_angle * launch_angle) + I(spray_angle * launch_speed) +
##      I(launch_angle * launch_speed), family = multinomial, data = bball)
##
## Coefficients:
##
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept):1      -1.021e+00  9.570e-02 -10.664 < 2e-16 ***
## (Intercept):2      -8.758e+00  3.668e-01 -23.876 < 2e-16 ***
## (Intercept):3      -9.472e+00  1.349e+00  -7.022 2.19e-12 ***
## (Intercept):4      -8.201e+01  4.605e+00 -17.807 < 2e-16 ***
## launch_speed:1       7.482e-03  1.010e-03   7.407 1.29e-13 ***
## launch_speed:2       5.941e-02  3.752e-03  15.834 < 2e-16 ***
## launch_speed:3       4.155e-02  1.370e-02   3.032 0.00243 **
## launch_speed:4       4.737e-01  3.957e-02  11.970 < 2e-16 ***
## launch_angle:1       4.413e-02  4.475e-03   9.862 < 2e-16 ***
## launch_angle:2      -3.720e-02  1.970e-02  -1.888 0.05898 .
## launch_angle:3      -1.407e-01  6.302e-02    NA      NA
## launch_angle:4       2.102e+00  1.734e-01  12.124 < 2e-16 ***
## spray_angle:1      -1.719e-02  3.477e-03  -4.946 7.59e-07 ***
## spray_angle:2      -1.983e-02  8.191e-03  -2.421 0.01549 *
## spray_angle:3       2.235e-02  2.845e-02   0.786 0.43200
## spray_angle:4       1.610e-02  3.667e-02   0.439 0.66069
## I(launch_angle^2):1  -1.432e-03  2.963e-05 -48.335 < 2e-16 ***
## I(launch_angle^2):2  -4.206e-03  1.143e-04 -36.779 < 2e-16 ***
## I(launch_angle^2):3  -4.513e-03  3.618e-04 -12.474 < 2e-16 ***
## I(launch_angle^2):4  -3.649e-02  1.096e-03 -33.296 < 2e-16 ***
```

```

## I(spray_angle^2):1          -1.615e-04  6.331e-05  -2.551  0.01075 *
## I(spray_angle^2):2          -2.376e-03  1.571e-04 -15.121  < 2e-16 ***
## I(spray_angle^2):3          -3.395e-03  4.625e-04  -7.341  2.11e-13 ***
## I(spray_angle^2):4          2.587e-03  2.677e-04   9.666  < 2e-16 ***
## I(spray_angle^3):1          -5.570e-06  1.138e-06  -4.895  9.83e-07 ***
## I(spray_angle^3):2          -4.860e-07  3.545e-06  -0.137  0.89096
## I(spray_angle^3):3          1.243e-06  1.438e-05   0.086  0.93112
## I(spray_angle^3):4          5.779e-06  7.238e-06   0.798  0.42460
## I(spray_angle^4):1          1.736e-08  3.992e-08   0.435  0.66367
## I(spray_angle^4):2          2.810e-06  1.302e-07  21.586  < 2e-16 ***
## I(spray_angle^4):3          3.142e-06  4.417e-07   7.113  1.13e-12 ***
## I(spray_angle^4):4          -1.555e-07  2.189e-07  -0.710  0.47759
## I(spray_angle^5):1          5.978e-10  1.939e-10   3.083  0.00205 **
## I(spray_angle^5):2          -1.497e-09  1.076e-09  -1.391  0.16429
## I(spray_angle^5):3          -3.256e-10  5.290e-09  -0.062  0.95092
## I(spray_angle^5):4          -1.949e-09  2.659e-09  -0.733  0.46360
## I(spray_angle^6):1          2.375e-12  4.872e-12   0.488  0.62590
## I(spray_angle^6):2          -5.001e-10  2.937e-11 -17.028  < 2e-16 ***
## I(spray_angle^6):3          -5.583e-10  1.215e-10  -4.594  4.34e-06 ***
## I(spray_angle^6):4          -5.527e-11  5.019e-11  -1.101  0.27081
## I(spray_angle * launch_angle):1  2.590e-04  2.559e-05  10.122  < 2e-16 ***
## I(spray_angle * launch_angle):2  3.545e-04  5.329e-05   6.652  2.88e-11 ***
## I(spray_angle * launch_angle):3 -1.617e-04  1.974e-04  -0.819  0.41287
## I(spray_angle * launch_angle):4 -1.786e-04  2.428e-04  -0.736  0.46198
## I(spray_angle * launch_speed):1  2.205e-04  3.502e-05   6.296  3.06e-10 ***
## I(spray_angle * launch_speed):2  1.172e-04  7.382e-05   1.587  0.11241
## I(spray_angle * launch_speed):3 -6.553e-05  2.623e-04  -0.250  0.80272
## I(spray_angle * launch_speed):4 -2.098e-04  3.153e-04  -0.665  0.50584
## I(launch_angle * launch_speed):1 -3.362e-04  5.187e-05  -6.482  9.05e-11 ***
## I(launch_angle * launch_speed):2  2.225e-03  2.048e-04  10.866  < 2e-16 ***
## I(launch_angle * launch_speed):3  3.709e-03  6.498e-04   5.708  1.15e-08 ***
## I(launch_angle * launch_speed):4  1.310e-03  1.360e-03   0.963  0.33551
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Names of linear predictors: log(mu[,1]/mu[,5]), log(mu[,2]/mu[,5]),
## log(mu[,3]/mu[,5]), log(mu[,4]/mu[,5])
##
## Residual deviance: 66221.19 on 199948 degrees of freedom
##
## Log-likelihood: -33110.59 on 199948 degrees of freedom
##
## Number of Fisher scoring iterations: 14
##
## Warning: Hauck-Donner effect detected in the following estimate(s):
## '(Intercept):2', '(Intercept):3', '(Intercept):4', 'launch_angle:3', 'I(launch_angle^2):2', 'I(launch_angle^2):3', 'I(launch_angle^2):4', 'I(launch_angle^3):2', 'I(launch_angle^3):3', 'I(launch_angle^3):4', 'I(launch_angle^4):2', 'I(launch_angle^4):3', 'I(launch_angle^4):4', 'I(launch_angle^5):2', 'I(launch_angle^5):3', 'I(launch_angle^5):4', 'I(launch_angle^6):2', 'I(launch_angle^6):3', 'I(launch_angle^6):4', 'I(spray_angle * launch_angle):1', 'I(spray_angle * launch_angle):2', 'I(spray_angle * launch_angle):3', 'I(spray_angle * launch_angle):4', 'I(spray_angle * launch_speed):1', 'I(spray_angle * launch_speed):2', 'I(spray_angle * launch_speed):3', 'I(spray_angle * launch_speed):4', 'I(launch_angle * launch_speed):1', 'I(launch_angle * launch_speed):2', 'I(launch_angle * launch_speed):3', 'I(launch_angle * launch_speed):4'
##
## Reference group is level 5 of the response
##
## user system elapsed
## 80.681 4.033 84.758

```

Notice that several terms are dropped from consideration, and the summary table refers to this as a [Hauck-Donner effect](#) which is another name for “near-separation” in the multinomial model. Current `glmldr` software

does not yet address separation in the multinomial response model.

Let's now examine what we can do with this flexible final model. We plot the $\log\left(\frac{\hat{\pi}_j(x)}{\hat{\pi}_J(x)}\right)$ as a function of spray angle for each hit outcome where J corresponds to an out. We will set launch angle and exit velocity to their median values.

```
summary(bball$spray_angle)

##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
## -89.9156 -23.5685  -0.6636    0.2956   24.7655   90.0000

summary(bball$launch_angle)

##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
##   -89.00    -6.00    12.00    12.29    30.00    89.00

summary(bball$launch_speed)

##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
##      9.40    80.00    90.30    88.07    98.80   121.30

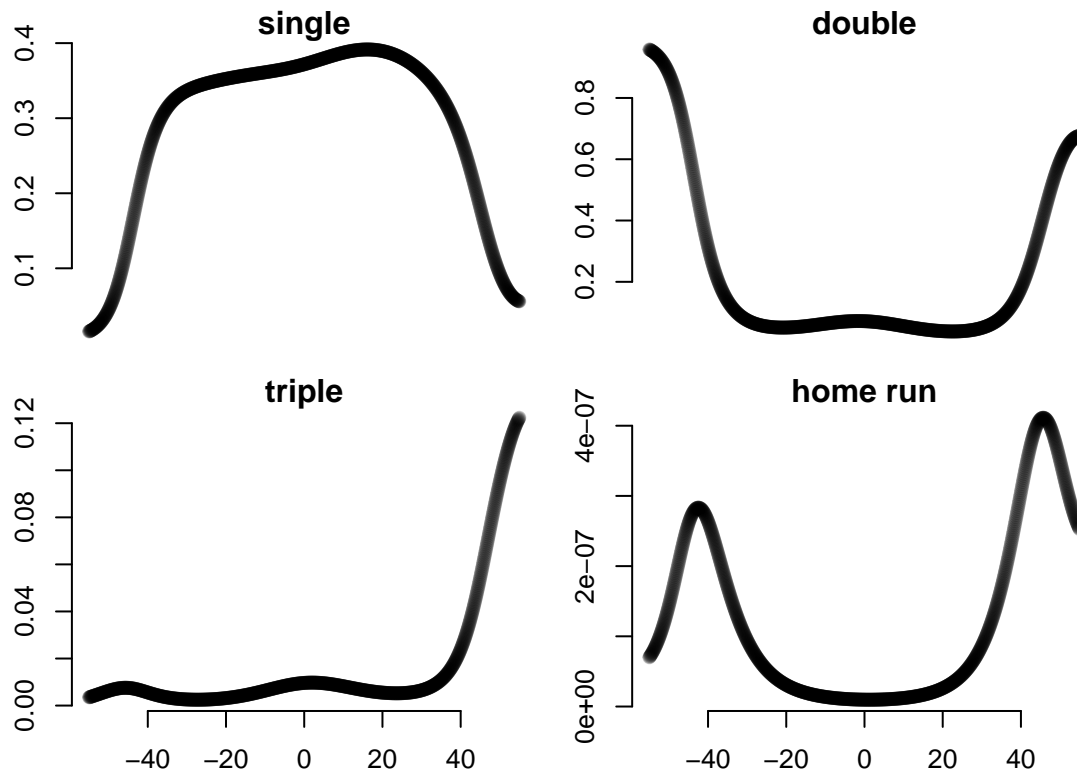
## obtain predictions
new_data = data.frame(spray_angle = seq(-55,55, by = 0.1),
                      launch_speed = 90.30,
                      launch_angle = 12)
pred = predict(mod4, newdata = new_data, type = "response")

par(mfrow = c(2,2), oma = c(4,4,0,0), mar = c(1,2,1,1))
plot.new()
title("single")
plot.window(xlim = c(-55,55), ylim = c(min(pred[, 1]), max(pred[, 1])))
points(new_data$spray_angle, pred[, 1], pch = 19, col = rgb(0,0,0,alpha=0.2))
axis(2)

plot.new()
title("double")
plot.window(xlim = c(-55,55), ylim = c(min(pred[, 2]), max(pred[, 2])))
points(new_data$spray_angle, pred[, 2], pch = 19, col = rgb(0,0,0,alpha=0.2))
axis(2)

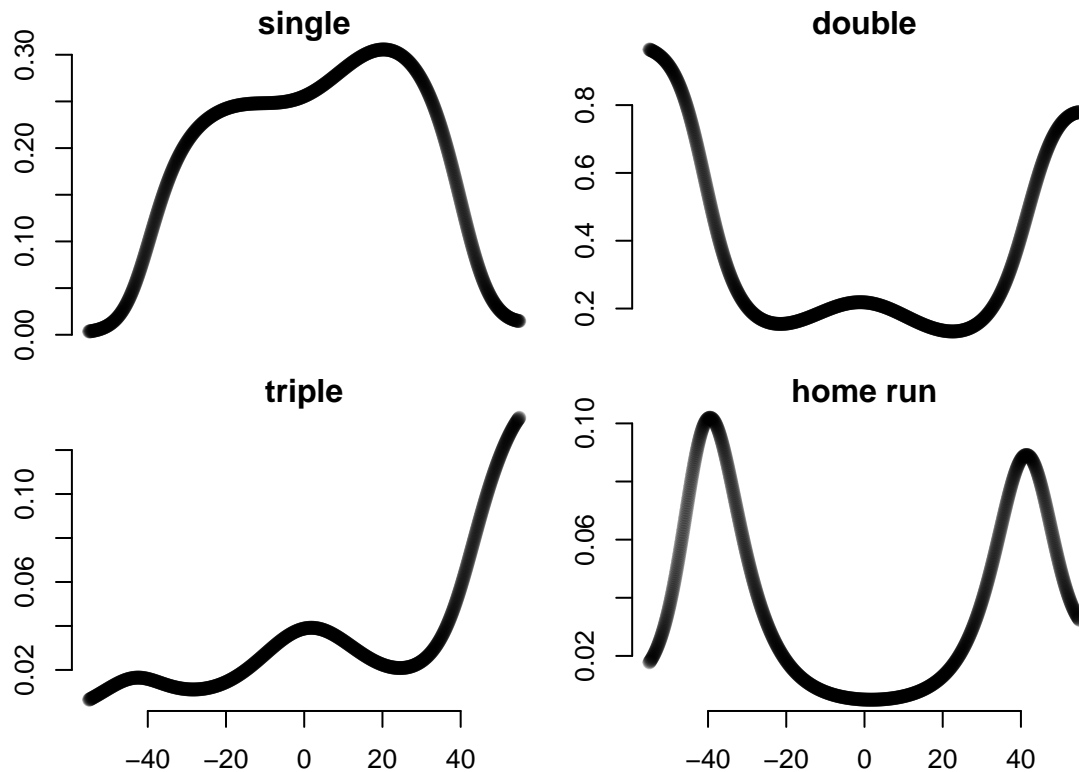
plot.new()
title("triple")
plot.window(xlim = c(-55,55), ylim = c(min(pred[, 3]), max(pred[, 3])))
points(new_data$spray_angle, pred[, 3], pch = 19, col = rgb(0,0,0,alpha=0.2))
axis(1)
axis(2)

plot.new()
title("home run")
plot.window(xlim = c(-55,55), ylim = c(min(pred[, 4]), max(pred[, 4])))
points(new_data$spray_angle, pred[, 4], pch = 19, col = rgb(0,0,0,alpha=0.2))
axis(1)
axis(2)
```



Let's consider these predicted values at a new combination of launch angle and exit velocity.

```
## obtain predictions
new_data = data.frame(spray_angle = seq(-55,55, by = 0.1),
  launch_speed = 100,
  launch_angle = 20)
pred = predict(mod4, newdata = new_data, type = "response")
```

An alternative model fitting function is available in the `nnet` package:

```
library(nnet) # has multinom function
system.time({mod4alt = multinom(events ~ launch_speed +
  launch_angle + spray_angle + I(launch_angle^2) +
  I(spray_angle^2) + I(spray_angle^3) + I(spray_angle^4) +
  I(spray_angle^5) + I(spray_angle^6) +
  I(spray_angle*launch_angle) + I(spray_angle*launch_speed) +
  I(launch_angle*launch_speed), data=bball, maxit = 1e3)})
```

```
## # weights: 70 (52 variable)
## initial value 80471.895622
## iter 10 value 71115.175486
## iter 20 value 65694.311578
## iter 30 value 58754.075454
## iter 40 value 55973.366379
## iter 40 value 55973.366379
## iter 50 value 52903.873054
## iter 60 value 48282.900567
## iter 70 value 47041.990949
## iter 70 value 47041.990949
## iter 80 value 46612.356353
## iter 90 value 42350.893366
## iter 100 value 41161.169449
## iter 110 value 39638.418181
## iter 120 value 39035.829021
## iter 130 value 38485.011435
## iter 140 value 37740.808989
## iter 150 value 37463.917203
## iter 160 value 36840.650567
```

```

## iter 170 value 36493.551763
## iter 180 value 35782.149057
## iter 190 value 35530.791473
## iter 200 value 35370.728874
## iter 210 value 35226.571426
## iter 220 value 35121.421843
## iter 230 value 35081.384809
## iter 240 value 35020.726417
## iter 250 value 34972.501435
## iter 260 value 34945.988675
## iter 270 value 34926.375444
## iter 280 value 34919.280452
## iter 290 value 34917.335314
## iter 300 value 34908.771298
## iter 310 value 34891.101750
## iter 320 value 34890.065619
## iter 330 value 34885.742128
## iter 340 value 34883.971480
## iter 350 value 34883.333647
## iter 360 value 34867.268410
## iter 370 value 34857.764695
## iter 380 value 34856.510809
## iter 390 value 34853.943645
## iter 390 value 34853.943645
## iter 400 value 34851.416597
## iter 410 value 34850.155614
## iter 420 value 34835.960756
## iter 430 value 34647.082237
## iter 440 value 34631.274175
## iter 450 value 34583.920032
## iter 460 value 34537.209507
## iter 470 value 34525.313146
## iter 480 value 34510.846518
## iter 490 value 34502.484661
## iter 500 value 34470.171829
## iter 510 value 34465.324366
## iter 520 value 34455.917650
## iter 530 value 34454.726360
## iter 540 value 34451.287707
## iter 550 value 34450.967108
## iter 560 value 34449.403679
## iter 570 value 34448.851783
## iter 570 value 34448.851612
## final value 34448.798620
## converged

## user system elapsed
## 50.404 0.354 50.838

```

Remarks:

1. This multivariate response model can be represented as a multivariate response GLM, see Section 8.1.5 in [Agresti \[2013\]](#). A probit-type model is also possible, see Section 8.1.6 in [Agresti \[2013\]](#). The (residual) degrees of freedom depends on the data format - grouped data with n multinomial observations generally have $(J - 1)(n - p)$ degrees of freedom where p is the number of elements in x .

2. Note that [Gerber and Craig \(2020\)](#) develop a mixed effects multinomial logistic-normal model for baseball applications. This model accounts for positive and negative correlations between outcomes that the standard multinomial model is not flexible enough to handle. The idea is that a player who hits a lot of home runs tends to hit a lot of doubles. Thus these outcomes are expected to exhibit positive correlation, and the standard multinomial model specifies these correlations as negative.
3. The baseball example could also be fit using ordinal responses, see below.
4. One may be interested in taking this analysis further and investigating where the ball is likely to go when a specific batter faces a specific pitcher. People at UIUC have done [work on this project](#) (note that one student author works at the Houston Astros and the other student author is an intern at the Baltimore Orioles). Fitting a multinomial regression model to estimate outcomes of individual matchups is likely impossible (too many parameters). The SEAM approach instead uses distance based weighting to pool outcomes from similar players in order to perform its estimation. SEAM has its roots in [PECOTA](#) and it borrows aesthetics from nonparametric density estimation, model averaging, and synthetic control methodology. One can see that PECOTA performs well when [compared to other more refined statistical approaches](#). A discussion of the reasons for this is beyond the scope of this course.

Acknowledgments

Aspects of these notes closely follow Trevor Park's slides. We also borrow materials from [Agresti \[2013\]](#).

References

Alan Agresti. *Categorical Data Analysis*. John Wiley & Sons, 2013.