Class 8 Demo: Using Neural Networks with Logistic Regression

```
### Install new packages:
# install.packages("nnet")
# install.packages("sjPlot")

### Load packages:
#library(knitr)
#library(janitor)
#library(haven)
#library(readxl)
#library(psych)
#library(statar)
library(tidyverse)
library(mktg482)
library(nnet)
library(sjPlot)
```

BBB Analysis

First, let's read in the bbb dataset. Note that we transform the buyer variable to be a factor, i.e., a categorical variable. This is required by the `nnet` function that we will use to fit a neural network so that it knows we are dealing with a classification problem (i.e., a discrete outcome).

```
# Clear environment of datasets
rm(list=ls())
load("../../Data/bbb.Rdata")
bbb <- bbb %>% mutate(buyer = factor(buyer))
```

Caution: Now that `buyer` is no longer a number, we cannot do mathematical operations on it. For instance, if we wanted to count the number of buyers, we previously could use code like the below. However, this will naturally fail now because `sum` makes no sense for non-numbers.

```
bbb %>%
  summarize(num_buyer = sum(buyer))
```

Instead, if we wanted to count the number of buyers, we could write:

```
bbb %>%
  summarize(num_buyer = sum(buyer==1))
```

```
# A tibble: 1 x 1
  num_buyer
      <int>
1      4522
```

This creates a logical (or Boolean or TRUE/FALSE) variable as can be seen below:

```
bbb %>%
  select(buyer) %>%
  mutate(buyer==1)
```

```
# A tibble: 50,000 x 2
```

```
   buyer `buyer == 1`
   <fct> <lgl>
 1 0     FALSE
 2 0     FALSE
 3 0     FALSE
 4 0     FALSE
 5 0     FALSE
 6 1     TRUE
 7 0     FALSE
 8 0     FALSE
 9 1     TRUE
10 0     FALSE
# ... with 49,990 more rows
```

For the purpose of mathematical operations on logical variables, `TRUE` and `FALSE` are treated as `1` and `0` respectively.

Neural Network

Following the guidance from the lecture, we first run a NN model (using `nnet`) and add the prediction to the dataset. The `nnet` function fits a neural network with a single hidden layer. However, we must specify how many hidden nodes we want in the layer. This is an example of what is known as a "tuning parameter".

In our case I picked `size = 5`. One always needs to play around with this number. Basically, it should be as small as possible while giving a good prediction. We will return to this later in the quarter.

There are also two other parameters we will specify to the function. First, we will specify another tuning parameter, `decay = 0.1`; `decay` is a "penalty" parameter that determines how the model estimates weights (again, more later in the quarter). Second, we will specify `maxit=1000`; this means we will stop fitting (or estimating) the NN after 1000 iterations, even if it has not converged on a solution.

Typically, we do not need to change `decay` and `maxit`, but `size` often requires adjustment.

Another final point is that we should always set a random number "seed" before fitting a NN. This is because the initial weights used in the first iteration of the estimation are random so setting a seed guarantees you will be able to exactly replicate your results. We can do this via the `set.seed()` and specify a number, here 1234 (note: it is good practice to do this and henceforth we will do so at the beginning of all of our code):

```
set.seed(1234)
nn_bbb <- nnet(buyer ~ gender + last + total + child + youth +
               cook + do_it + reference + art + geog,
               data=bbb, size=5, decay=0.1, maxit=1000)
```

Next, we create a prediction, using the usual `predict()` command and add it to our original dataset. Notice that predict here is of `type="raw"`, which is specific to the neural network (recall for predicting from a logistic regression we use `type="response"`):

```
bbb <- bbb %>%
  mutate(score_nn = predict(nn_bbb, type="raw"))
```

Logistic Regression

We now recreate out logistic prediction from the assignment.

```
lr_bbb <- glm(buyer ~ gender + last + total + child + youth +
              cook + do_it + reference + art + geog,
```

```
              family=binomial, data=bbb)
bbb <- bbb %>%
  mutate(score_lr = predict(lr_bbb, type="response"))
```

Model Comparison

We are now ready to compare the performance of the NN and the logistic regression. How do we do that? With a gains plot and/or AUC of course!

Instead of having R calculate the summary data table and then plotting the gains curve in Excel, I have created an R function `gainsplot()` that does it automatically for you; it also calculates the AUC of each model. To use it, you need to use load the `mktg482` library as we did using the command `library(mktg482)`; if that command fails for you, you need to install this package by typing

```
devtools::install_github("fzettelmeyer/mktg482", upgrade = "never", force = TRUE)
```

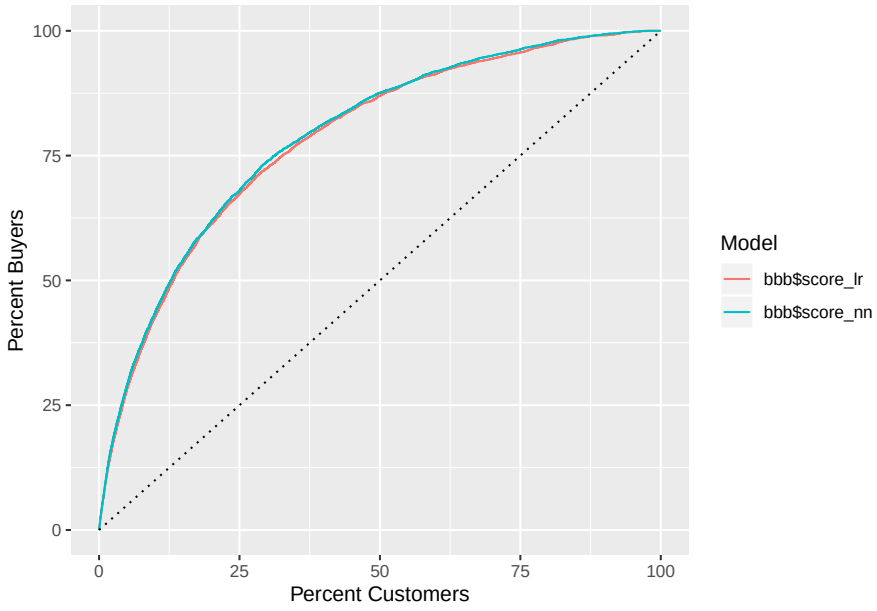
into the console (you only need to do this once).

The syntax for using the `gainsplot()` function is:

```
gainsplot(score1, score2, ..., scoreN, label.var = <dependent variable>)
```

Let's use it!

```
gainsplot(bbb$score_lr, bbb$score_nn, label.var = bbb$buyer)
```




```
# A tibble: 2 x 2
  model          auc
  <chr>        <dbl>
1 bbb$score_lr 0.812
2 bbb$score_nn 0.818
```

As you can see, the models perform extremely similarly. The NN performs perhaps a tiny bit better. However, the logistic regression model does admirably well and we should probably use it given how fast and easy to interpret it is.

Advertising Analysis

Now, a second example. In this dataset, the response to advertising `res` was recorded. 10,000 consumers were targeted with either adcopy A (`ada` equal to 1) or adcopy B (`ada` equal to 0). In addition, we have information on gender (`female`), number of prior purchases (`numpurch`), and prior total spend (`totdol`). Please ignore the `training` variable for now.

Note, in general it is good practice to explicitly turn transform *all* categorical variables into factors so we will do that here and going forward:

```
ads <- read_csv("../../Data/firewall_advertising.csv")
ads <- ads %>%
    mutate(res=factor(res), female=factor(female), ada=factor(ada))
str(ads)
```

```
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    10000 obs. of  7 variables:
 $ age     : num  31 44 41 34 40 35 44 33 41 35 ...
 $ numpurch: num  1 2 1 2 1 1 1 2 1 1 ...
 $ totdol  : num  125 768 87 57 56 69 84 54 248 70 ...
 $ ada     : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 2 2 1 2 ...
 $ female  : Factor w/ 2 levels "0","1": 1 1 1 2 1 2 2 2 1 2 ...
 $ res     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ training: num  0 0 0 0 0 0 0 0 0 0 ...
```

Neural Network

We fit the neural network and obtain the predictions just like we did above:

```
nn_ads <- nnet(res ~ age + numpurch + totdol + ada + female,
               data=ads,size=5, decay=0.1, maxit=1000)
ads <- ads %>%
  mutate(score_nn = predict(nn_ads, type="raw"))
```

Logistic Regression

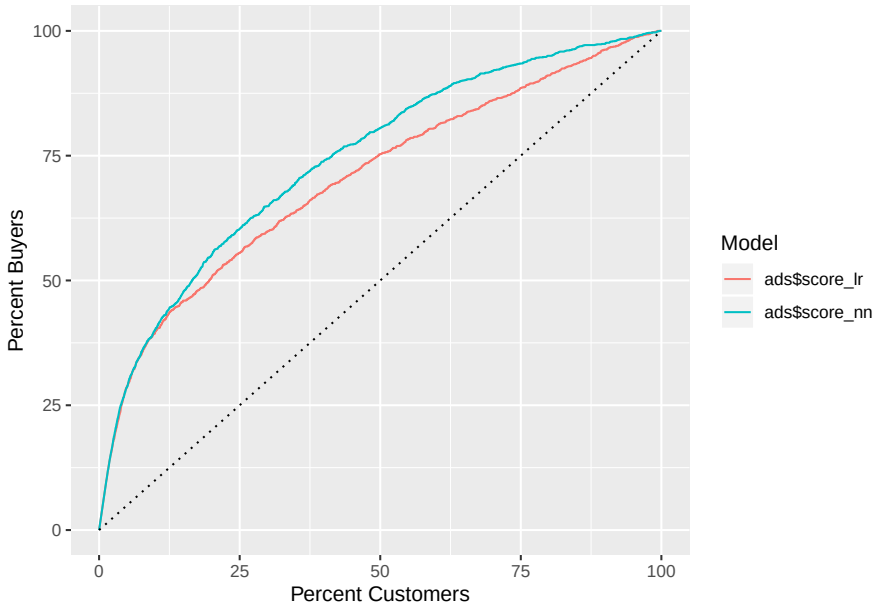
We also fit the logistic regression and obtain the predictions just like we did above:

```
lr_ads <- glm(res ~ age + numpurch + totdol + ada + female,
              family=binomial, data=ads)
ads <- ads %>%
  mutate(score_lr = predict(lr_ads, type="response"))
```

Model Comparison

We now compare the models just like we did above:

```
gainsplot(ads$score_lr, ads$score_nn, label.var = ads$res)
```




```
# A tibble: 2 x 2
  model          auc
  <chr>        <dbl>
1 ads$score_lr 0.735
2 ads$score_nn 0.78
```

As you can see, the models do *not* perform similarly. The NN performs MUCH better. So, this suggest we should reevaluate the logistic regression. Let's look at the model output:

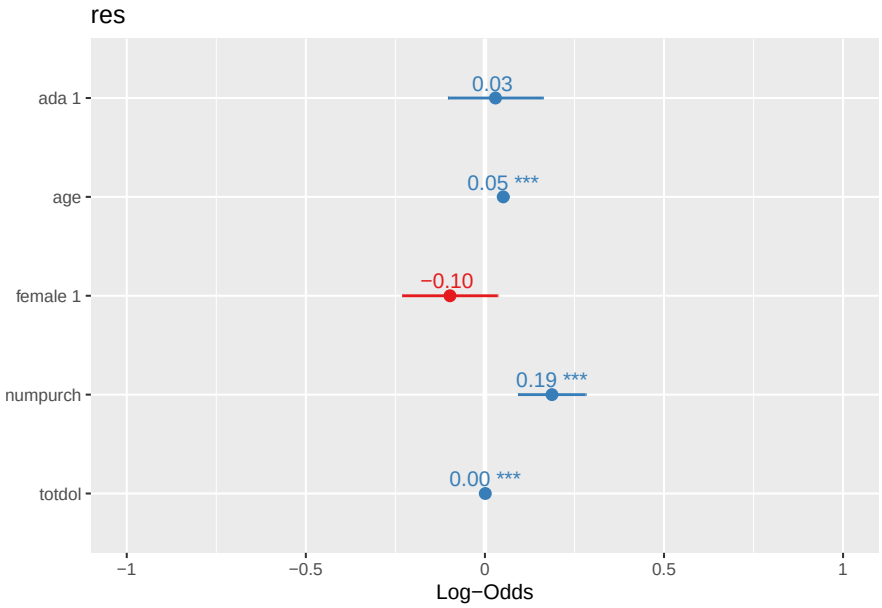
```
summary(lr_ads)
```

```

Call:
glm(formula = res ~ age + numpurch + totdol + ada + female, family = binomial,
    data = ads)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5032  -0.4572  -0.4068  -0.3625   2.4797

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -4.6847322  0.3135787 -14.940  < 2e-16 ***
age          0.0511779  0.0078473   6.522 6.95e-11 ***
numpurch     0.1872025  0.0481534   3.888 0.000101 ***
totdol       0.0010460  0.0001011  10.350  < 2e-16 ***
ada1         0.0294470  0.0675837   0.436 0.663046
female1     -0.0975399  0.0676866  -1.441 0.149570
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7731.5  on 9999  degrees of freedom
Residual deviance: 6273.8  on 9994  degrees of freedom
AIC: 6285.8
```

```
Number of Fisher Scoring iterations: 5
```

As usual, the coefficients are a bit hard to read so we plot the with the `plot_model` command from the `sjPlot` package:

```
plot_model(lr_ads, show.values=TRUE, transform=NULL)
```



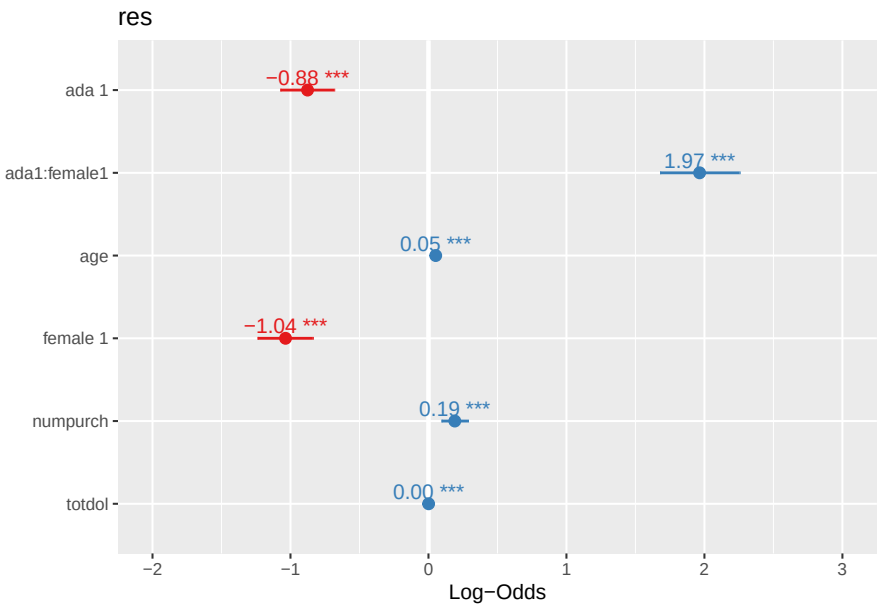
Is there anything surprising about these estimates? Can you think of a way to improve the model? What could be missing?

Discuss!

Improving the Logistic Regression

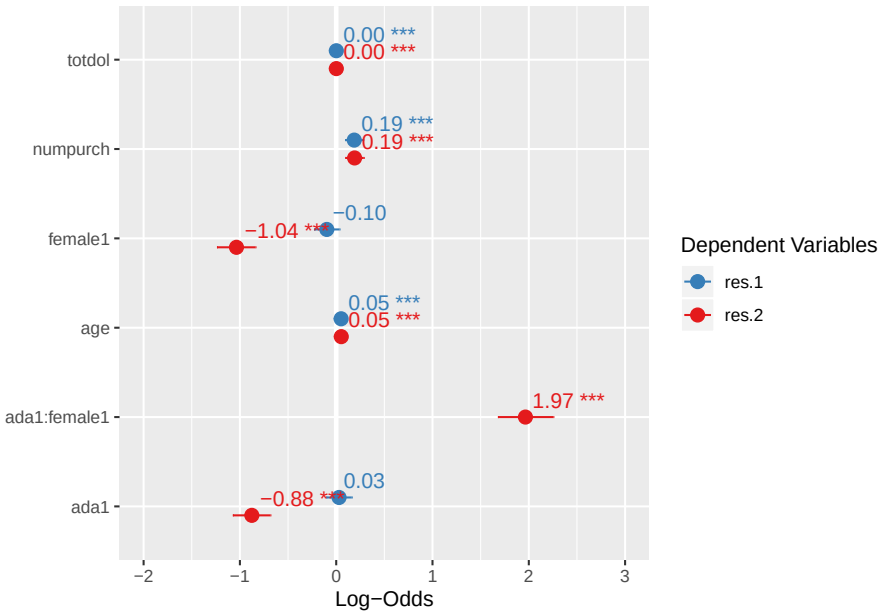
Let's fit a new logistic regression with an interaction between `ads` and `gender`, save the predictions, and plot the model coefficients:

```
lr_ads2 <- glm(res ~ age + numpurch + totdol + ada*female,
               family=binomial, data=ads)
ads <- ads %>%
  mutate(score_lr2 = predict(lr_ads2, type="response"))
plot_model(lr_ads2, show.values=TRUE, transform=NULL)
```



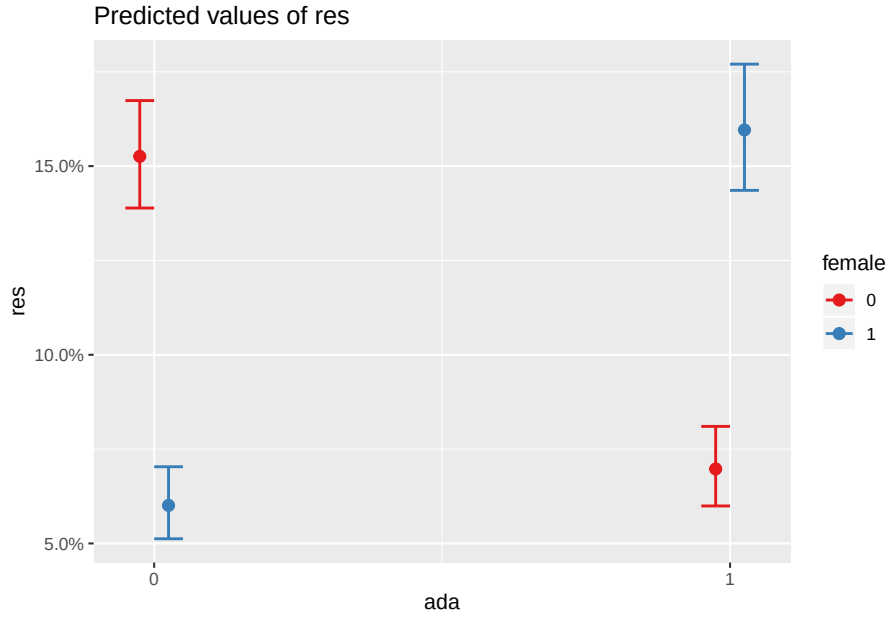
Notice how the coefficients for `ada`, `female`, and the interaction effect are now quite sizable? Using `plot_models()` (notice the plural), we can make this clearer:

```
plot_models(lr_ads, lr_ads2, show.values=TRUE, transform=NULL)
```



Finally, we can visualize the interaction effect to understand what happened. By using `plot_model()` and specifying `type = "int"`, `plot_model()` automatically starts looking for the interaction effects specified in the model and show us the effect for the interacted variables on the outcome:

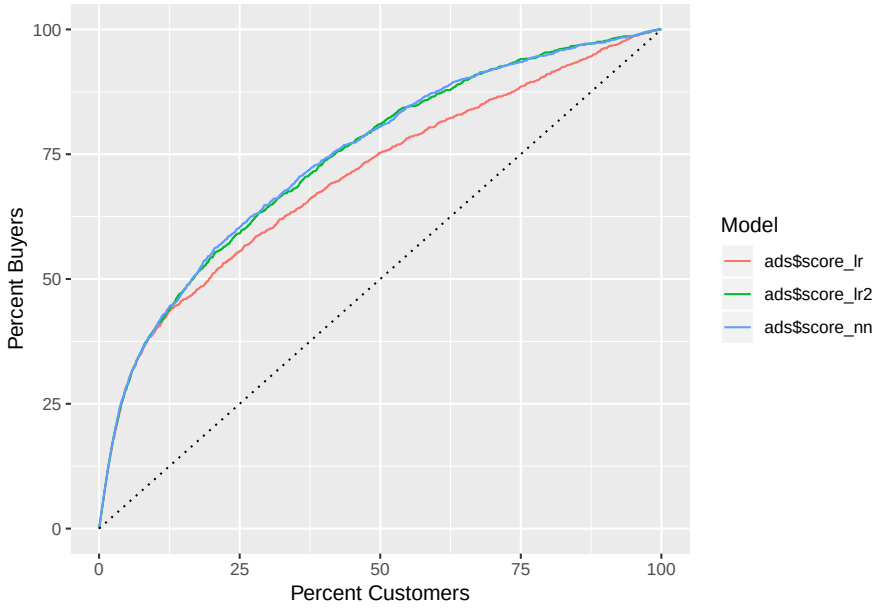
```
plot_model(lr_ads2, type = "int")
```



Model Comparison II

We are now ready to compare the performance of the NN and the new logistic regression using `gainsplot`.

```
gainsplot(ads$score_lr, ads$score_lr2, ads$score_nn, label.var = ads$res)
```




```
# A tibble: 3 x 2
  model           auc
  <chr>         <dbl>
```

```
1 ads$score_lr  0.735
2 ads$score_lr2 0.778
3 ads$score_nn  0.78
```

We seem to have matched the NN with our interaction!