Classification

Will Doyle

Classification is the process of predicting group membership. Understanding which individuals are likely to be members of which groups is a key task for data scientists. For instance, most recommendation engines that are at the hear of consumer web sites are based on classification algorithms, predicting which consumers are likely to purchase which products.

Pizza

Today we'll be working with the pizza dataset, which comes from the subreddit random acts of pizza. Each line represents a post to this subreddit. We have various characteristics of these posts, along with the request text from the post itself. We'll use these characteristics of the posts to predict whether or not the poster received pizza. This lesson is inspired by this article

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.3
```

```
## -- Attaching packages --------------------------------------------------------------------------- tidyve
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
## Warning: package 'tibble' was built under R version 3.5.3
```

```
## Warning: package 'tidyr' was built under R version 3.5.3
```

```
## Warning: package 'purrr' was built under R version 3.5.3
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
## Warning: package 'stringr' was built under R version 3.5.3
```

```
## Warning: package 'forcats' was built under R version 3.5.3
```

```
## -- Conflicts -------------------------------------------------------------------------- tidyverse_c
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.5.3
```

```r
library(modelr)
```

```
## Warning: package 'modelr' was built under R version 3.5.3
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(forcats)
```

```
load("za_train.Rdata")
```

Our goal is to create a classifier that will accurately classify people in a testing dataset as to whether they will receive a pizza or not, based on the content of their post. This is a VERY common task in data science–taking user supplied content and using it to accurately classify that user, typically as someone who will buy a product or service.

## Dependent Variable

Our dependent variable is a binary variable, `got_pizza` that denotes whether the user indicated that someone had sent them a pizza after posting in the subreddit "random acts of pizza". Let's take a look at this and see how many people posted that they got a pizza.

```
table(za_train$got_pizza)
```

```
##
##    0    1
## 2136  700
```

This tells us the raw numbers. Lots of times we want to know the proportions. The function `prop.table` can do this for us.

```
prop.table(table(za_train$got_pizza))
```

```
##
##         0         1
## 0.7531735 0.2468265
```

So, 0.2468265 of posts indicate that they were sent a pizza as a result of their post. We're interested in taking information in the posts themselves to see what makes it more or less likely that they would indicate that they received a pizza.

Conditional Means as a Classifier

We'll start by generating some cross tabs and some quick plots, showing the probability of receiving pizza according to several characteristics of the post. We start with a basic crosstab of the dependent variable. We use `prop.table` to change this from raw counts to proportions. I also provide a brief exampl of how to do a table using the `kable` function.

```
#Cross Tabs

za_train%>%
  count(got_pizza)%>% # Count numbers getting pizza
  mutate(p=prop.table(n))%>% #mutate for proportions using prop.table
  kable(format="markdown") # output to table
```

got_pizza	n	p
0	2136	0.7531735
1	700	0.2468265

So, about 75% of the sample didn't get pizza, about 25% did.

Next, we cross-tabulate receiving pizza with certain terms. First, if the request mentioned the word "student."

```
prop.table(table(za_train$student,za_train$got_pizza),margin=1)
```

```
##
##                     0         1
##   No student 0.7590683 0.2409317
##   Student    0.6820276 0.3179724
```

Next, if the request mentioned the word "grateful."

```
g_table<-table(za_train$grateful,za_train$got_pizza);g_table
```

```
##
##                          0    1
##   Grateful not in post 2071  673
##   Grateful in post       65   27
```

```
prop.table(g_table,margin=1)
```

```
##
##                               0         1
##   Grateful not in post 0.7547376 0.2452624
##   Grateful in post     0.7065217 0.2934783
```

Crosstabs using binary data are equivalent to generating conditional means, as shown below.

```
#Predictions using conditional means

za_train%>%group_by(grateful)%>%summarize(mean(got_pizza))
```

```
## # A tibble: 2 x 2
##   grateful             `mean(got_pizza)`
##   <fct>                            <dbl>
## 1 Grateful not in post             0.245
## 2 Grateful in post                 0.293
```

Note how the mean of got pizza is equivalent to the proportion answering "1" in the previous table.

But, we can also use conditional means to get proportions for very particular sets of characteristics. In this case, what about individuals who included some combination of the terms "grateful","student" and "poor" in their posts?

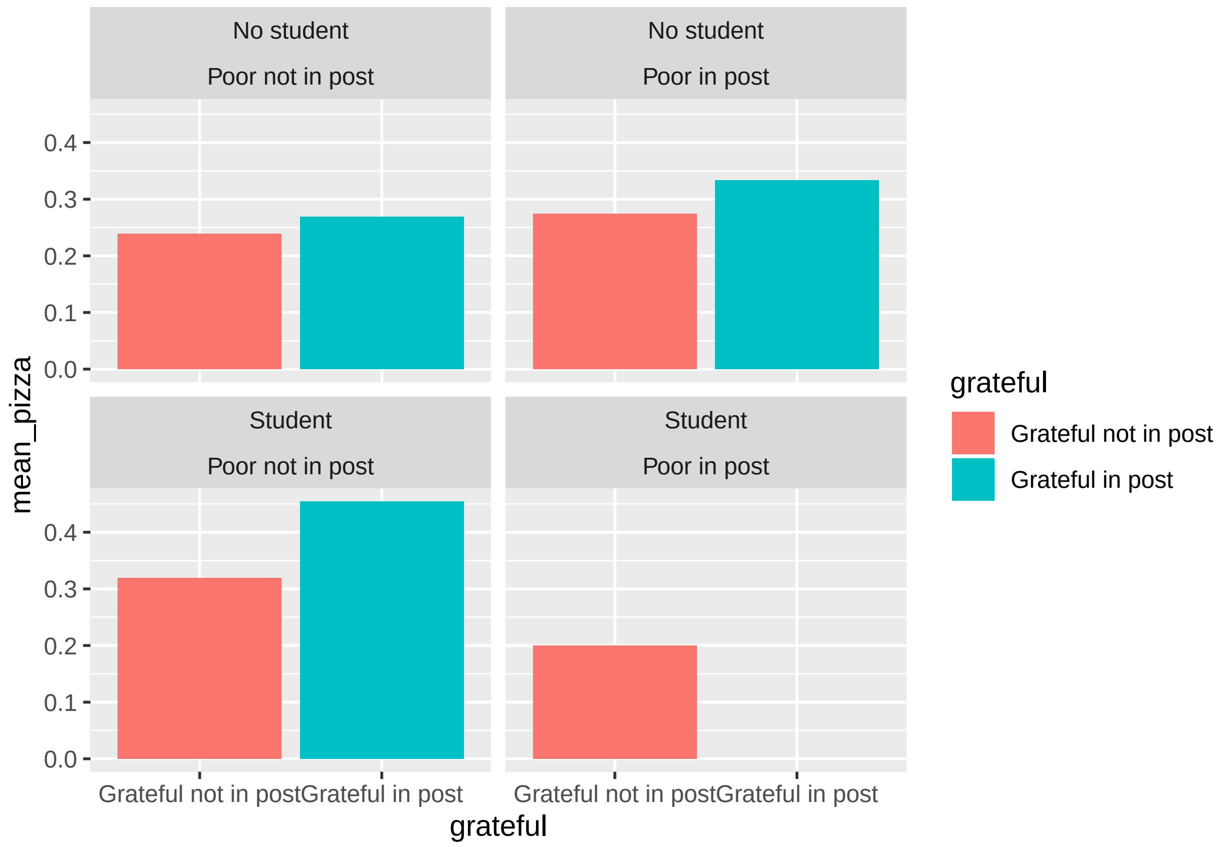
```
za_sum<-za_train%>%
  group_by(grateful,student,poor)%>%
  summarize(mean_pizza=mean(got_pizza))%>%
  arrange(-mean_pizza)

za_sum%>%kable()
```

grateful	student	poor	mean_pizza
Grateful in post	Student	Poor not in post	0.4545455
Grateful in post	No student	Poor in post	0.3333333
Grateful not in post	Student	Poor not in post	0.3193717
Grateful not in post	No student	Poor in post	0.2745098
Grateful in post	No student	Poor not in post	0.2692308
Grateful not in post	No student	Poor not in post	0.2392441
Grateful not in post	Student	Poor in post	0.2000000

The initial evidence here makes it look like the posts that included the terms “Grateful” and “student” had the highest probability of receiving a pizza (or at least posting that they received a pizza!).

Probability of Receiving Pizza, Using Various Terms in Post

```
gg<-ggplot(za_sum,aes(x=grateful,y=mean_pizza,fill=grateful))
gg<-gg+geom_bar(stat="identity")
gg<-gg+facet_wrap(~student+poor)
gg
```



Classification Using Linear Probability Model

We can use standard OLS regression for classification. It’s not ideal, but most of the time it’s actually not too bad, either. Below we model the binary outcome of receiving pizza as a function of karma, total posts, posts on the pizza subreddit, whether or not the poster mentioned the words “student” or “grateful.”

```
# Linear model
lm_mod<-lm(got_pizza~
             karma+
             total_posts+
             raop_posts+
             student+
             grateful,
           data=za_train,y=TRUE,na.exclude=TRUE);summary(lm_mod)
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'na.exclude' will be disregarded
```

```
##
## Call:
## lm(formula = got_pizza ~ karma + total_posts + raop_posts + student +
##     grateful, data = za_train, y = TRUE, na.exclude = TRUE)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6932 -0.2288 -0.2260 -0.2172  0.7950
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              2.260e-01  9.298e-03  24.310  < 2e-16 ***
## karma                    5.039e-06  2.403e-06   2.097  0.03609 *
## total_posts             -1.180e-04  2.058e-04  -0.573  0.56640
## raop_posts               1.558e-01  2.566e-02   6.072 1.43e-09 ***
## studentStudent           8.221e-02  3.022e-02   2.720  0.00657 **
## gratefulGrateful in post  4.752e-02  4.536e-02   1.048  0.29489
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4275 on 2830 degrees of freedom
## Multiple R-squared:  0.01913,    Adjusted R-squared:  0.01739
## F-statistic: 11.04 on 5 and 2830 DF,  p-value: 1.521e-10
```

We're going to do something a bit different with the predictions from this model. After creating predictions, we're going to classify everyone with a predicted probablity above .5 as being predicted to get a pizza, while everyone with a predicted probability below .5 is predicted to not get one. We'll compare our classifications with the actual data.

```
#Predictions
za_train<-za_train%>%
  add_predictions(lm_mod)%>% ## Add in predictions from the model
  rename(pred_lm=pred)%>% ## rename to be predictions from ols (lm)
  mutate(pred_lm_out=ifelse(pred_lm>=.5,1,0))
```

Let's create a table that shows the predictions of our model against what actually happened

```
pred_table<-table(za_train$got_pizza,za_train$pred_lm_out)

pred_table
```

```
##
##         0    1
##   0 2125   11
##   1  680   20
```

```
prop.table(pred_table)
```

```
##
##               0           1
##   0 0.749294781 0.003878702
##   1 0.239774330 0.007052186
```

```
rownames(pred_table)<-c("Predicted 0","Predicted 1")
colnames(pred_table)<-c("Actually 0","Actually 1")
```

```
ModelMetrics::confusionMatrix(za_train$got_pizza,za_train$pred_lm_out)
```

```
##      [,1] [,2]
## [1,] 2125  680
## [2,]   11   20
```

```
caret::confusionMatrix(as.factor(za_train$got_pizza),as.factor(za_train$pred_lm_out))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2125   11
##          1  680   20
##
##                Accuracy : 0.7563
##                  95% CI : (0.7401, 0.7721)
##     No Information Rate : 0.9891
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0345
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.75758
##             Specificity : 0.64516
##          Pos Pred Value : 0.99485
##          Neg Pred Value : 0.02857
##              Prevalence : 0.98907
##          Detection Rate : 0.74929
##    Detection Prevalence : 0.75317
##       Balanced Accuracy : 0.70137
##
##        'Positive' Class : 0
##
```

The confusion matrix generated here is explained here.

We're usually interested in three things: the overall accuracy of a classification is the proportion of cases accurately classified. The sensitivity is the proportion of "ones" that are accurately classified as ones– it's the probability that a case classified as positive will indeed be positive. Specificity is the probability that a case classified as 0 will indeed by classified as 0.

Question: how do you get perfect specificity? How do you get perfect sensitivity?

There are several well-known problems with linear regression as a classification algortihm. Two should give us pause: it can generate probabilites outside of 0,1 and it implies a linear change in probabilities as a function of the predictors which may not be justified given the underlying relationship between the predictors and the probability that the outcome is 1. Logistic regresssion should give a better predicted probability, one that's more sensitive to the actual relationship between the predictors and the outcome.

Logistic regression as a classifier

Logistic regression is set up to handle binary outcomes as the dependent variable. In particular, the predictions will always be a probability, which makes it better than the ironically named linear probability model. The downside to logistic regression is that it is modeling the log odds of the outcome, which means all of the coefficients are expressed as log odds, which no one understands intuitively. In this class, we're going to concentrate on logistic regression's ability to produce probabilities as predictions. Below I run the same model using logistic regression. Note the use of `glm` and the `family` option, which specifies a functional form and a particular link function.

```r
#Logisitic model

logit_mod<-glm(got_pizza~
             karma+
             total_posts+
             raop_posts+
             student+
             grateful,
             data=za_train,
            na.action=na.exclude,
            family=binomial(link="logit"),
               y=TRUE)

summary(logit_mod)
```

```
##
## Call:
## glm(formula = got_pizza ~ karma + total_posts + raop_posts +
##     student + grateful, family = binomial(link = "logit"), data = za_train,
##     na.action = na.exclude, y = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0224  -0.7192  -0.7146  -0.6478   1.9943
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -1.235e+00  5.211e-02 -23.699  < 2e-16 ***
## karma                    5.358e-05  2.246e-05   2.385  0.01707 *
## total_posts             -1.939e-03  1.439e-03  -1.347  0.17786
## raop_posts               7.467e-01  1.362e-01   5.482 4.21e-08 ***
## studentStudent           4.171e-01  1.536e-01   2.715  0.00662 **
## gratefulGrateful in post  2.384e-01  2.359e-01   1.010  0.31226
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3169.6  on 2835  degrees of freedom
## Residual deviance: 3118.7  on 2830  degrees of freedom
## AIC: 3130.7
##
## Number of Fisher Scoring iterations: 4
```

With these results in hand we can generate predicted probabilities and see if this model did any better. To get predicted probabilities, we need to specify `type=response` in our prediction call.

```r
za_train<-za_train%>%
  mutate(pred_logit=predict(logit_mod,type="response"))
```

We can convert the predictions to a binary variable by setting a "threshold" of .5. Any prediction above .5 is considered to be a 1, anything below, a 0.

```
za_train<-za_train%>%
    mutate(pred_logit_out=ifelse(pred_logit>=.45,1,0))

za_train<-za_train%>%
    mutate(pred_logit_out=as.factor(pred_logit_out))

za_train<-za_train%>%
    mutate(got_pizza=as.factor(got_pizza))
```

Now we create a confusion matrix to see how we did.

```
confusionMatrix(data=as.factor(za_train$pred_logit_out),reference=as.factor(za_train$got_pizza))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2119  676
##          1   17   24
##
##                Accuracy : 0.7556
##                  95% CI : (0.7394, 0.7714)
##     No Information Rate : 0.7532
##     P-Value [Acc > NIR] : 0.3899
##
##                   Kappa : 0.0385
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99204
##             Specificity : 0.03429
##          Pos Pred Value : 0.75814
##          Neg Pred Value : 0.58537
##              Prevalence : 0.75317
##          Detection Rate : 0.74718
##    Detection Prevalence : 0.98554
##       Balanced Accuracy : 0.51316
##
##        'Positive' Class : 0
##
```

Applying predictions to the testing dataset.

With our new (not very good) classifier, we can now add predictions to the testing dataset, and see how good this classifier is at predicting out of sample information.

```
load("za_test.RData")

za_test<-za_test%>%
  mutate(pred_logit=predict(logit_mod,newdata=.,type="response"))%>%
      mutate(pred_logit_out=ifelse(pred_logit>=.45,1,0))
```

```r
za_test<-za_test%>%
    mutate(pred_logit_out=as.factor(pred_logit_out))

za_test<-za_test%>%
    mutate(got_pizza=as.factor(got_pizza))


confusionMatrix(data=za_test$pred_logit_out,reference=za_test$got_pizza)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2095  674
##          1   11   23
##
##                Accuracy : 0.7556
##                  95% CI : (0.7393, 0.7714)
##     No Information Rate : 0.7513
##     P-Value [Acc > NIR] : 0.3086
##
##                   Kappa : 0.0407
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9948
##             Specificity : 0.0330
##          Pos Pred Value : 0.7566
##          Neg Pred Value : 0.6765
##              Prevalence : 0.7513
##          Detection Rate : 0.7474
##    Detection Prevalence : 0.9879
##       Balanced Accuracy : 0.5139
##
##        'Positive' Class : 0
##
```

Thinking about classifiers

First, make sure that your dependent variable really is binary. If you're working with a continuous variable (say, income) don't turn it into a binary variable (e.g. low income).

Second, remember that classifiers must always balance sensitivity and specificity. Don't be overly impressed by a high overall percent correctly predicted, nor a high level of either specificity or sensitivity. Instead, look for classifiers that have both.