GLMs and Logistic Regression

36-600

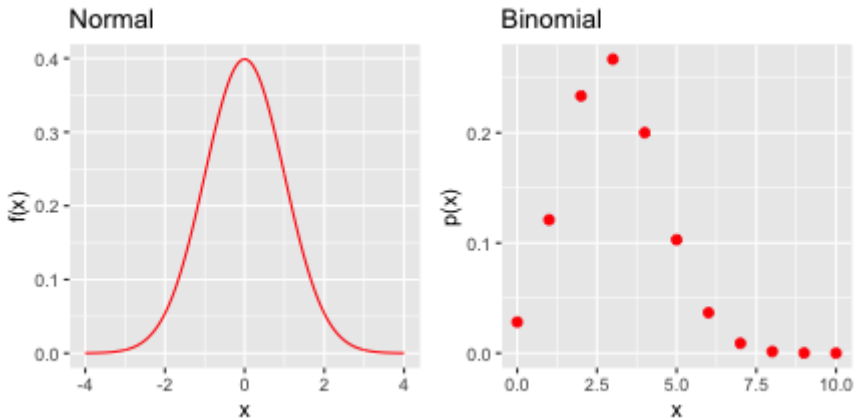
Fall 2021

Review: Probability Distributions

A probability distribution is a mathematical function $f(x|\theta)$ where

- x may take on continuous values or discrete values;
- θ is a set of parameters governing the shape of the distribution (e.g., $\theta = \{\mu, \sigma^2\}$ for a normal);
- $f(x|\theta) \geq 0$ for all y; and
- $\sum_x f(x|\theta) = 1$ or $\int_x f(x|\theta) = 1$.

When x is discrete, $f(x|\theta)$ is a *probability mass function*, or *pmf*, and when x is continuously valued, $f(x|\theta)$ is a *probability density function* or *pdf*.



Probability Distributions and Regression

We are discussing distributions because in parameterized regression we make assumptions about how the response variable is distributed around the true regression line.

For instance, for linear regression, we assume that for every $\mathbf{x}$...

- the distribution governing the possible values of Y is a *normal* distribution;
- the mean of the normal distribution is $E[Y|\mathbf{x}] = \beta_0 + \sum \beta_i x_i$; and
- the variance of the normal distribution is σ^2, which is a constant (i.e., does not vary with x).

What if we cannot (or more to the point, should not) assume that $Y|\mathbf{x} \sim \mathcal{N}(\beta_0 + \sum \beta_i x_i, \sigma^2)$, i.e., what if Y is not a normally distributed random variable?

This is where we enter the realm of *generalized linear models* or *GLMs*.

Generalization

In typical linear regression, the domain of $Y|\mathbf{x}$ is assumed to be $(-\infty, \infty)$, matching the domain of the mean μ of a normal distribution.

What, however, happens if we observe that the response variable is, e.g., discretely valued, with possible values 0, 1, 2, ...?

The normal distribution isn't the correct one to assume here. What would be the right distribution to assume? In practice, there will be many possibilities and we might not know which one is right, but any assumption we make *should* be consistent with how the response is empirically distributed.

Domain of Y	Consistent Distributions
$(-\infty, \infty)$	normal
$[0, \infty)$	Poisson, gamma, exponential, chi-square
$[0, n]$	multinomial
$[0, 1]$	beta, binomial

Some of the distributions above are continuous and some are discrete. At first, this does not matter, as the first step in generalization is to map the linear regression line in such a way that it "fits" within the domain boundaries. This mapping is done by way of a *link function*.

Generalization: Link Functions

Let's keep things in the realm of one predictor, where the linear function is

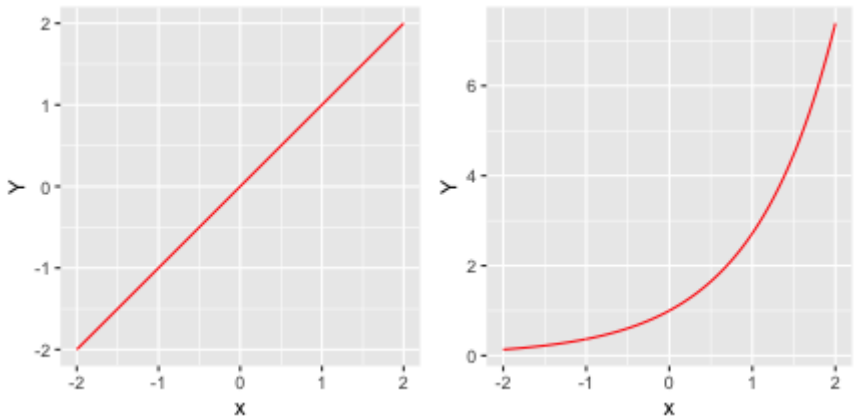
$$\beta_0 + \beta_1 x \, .$$

As noted above, the range of this function is $(-\infty, \infty)$. But let's assume that the data we observe is bounded from below at 0, i.e., $Y \in [0, \infty)$. The regression line $\mu|x$ around which the observed data are randomly scattered cannot be negative, so we need to find a mapping function.

There are no unique transformations, but rather ones that are commonly used. Here, that commonly used function would be:

$$g(\mu|x) = \log(\mu|x) = \beta_0 + \beta_1 x \;\Rightarrow\; \mu|x = e^{\beta_0 + \beta_1 x} \, .$$

$g(\cdot)$ is the *link* function. To the right, below, we demonstrate the application of this link function.



Generalization: Optimization

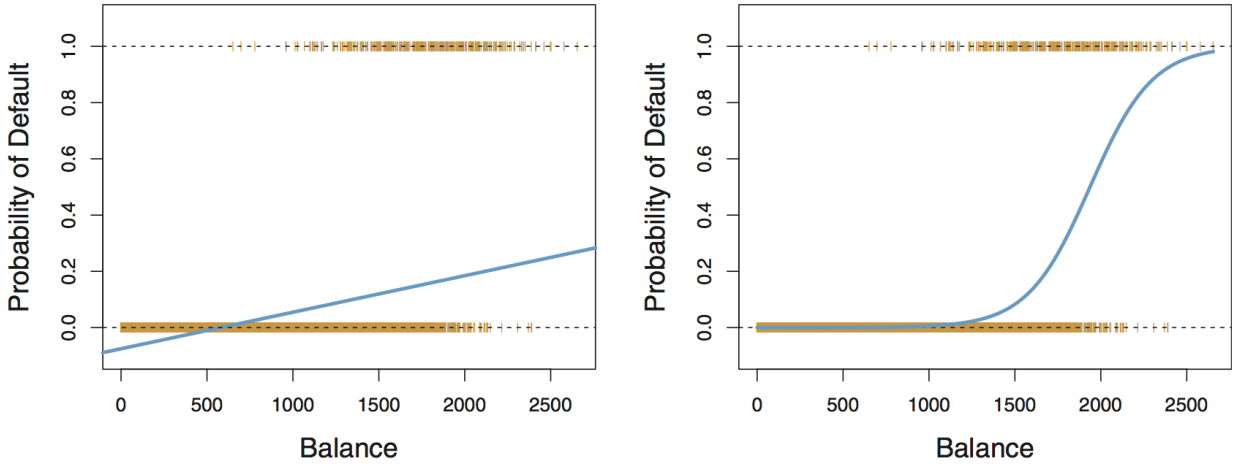
Once the link function is set, we use numerical optimization to estimate β_0 and β_1 via maximization of the likelihood function, which for our current example would look like:

$$\mathcal{L} = \prod_{i=1}^{n_{\text{train}}} f(Y_i|\mu_i = e^{\beta_0+\beta_1 x_i}) \,,$$

where n_{train} is the number of (training set) data. Before we continue, a few notes about this:

- Maximizing a likelihood function is *exactly* the same idea as optimizing an objective function like you've done for curve fitting. The goal is to find optimal values of β_0 and β_1.
- However, to do this, we need to *pick an appropriate distribution*. It is not just about the domain, but whether are not $Y|x$ is discretely valued or continuously valued. In our example, if $Y|x$ is discretely valued (specifically: 0, 1, 2, ...), then we could assume the Poisson distribution. If continuous, on the other hand, perhaps we pick the gamma distribution. We need to do this so that we h exactly $f(Y_i|\mu_i = e^{\beta_0+\beta_1 x_i})$ is defined!
- While multiple linear regression utilizes formulae and is fast, Poisson regression and other GLMs utilize numerical optimization and thus are relatively slow, something that might be noticed for large datasets.

Logistic Regression



(Figure 4.2, *Introduction to Statistical Learning* by James et al.)

- To the left is a linear regression fit. The regression line *is not* limited to lay within the domain [0,1].
- To the right is a logisitic regression fit. The regression line *is* limited to lay within the range [0,1].

Logistic Regression

Logistic regression is appropriate for datasets where the response variable can only take on two discrete values (assumed to map to 0 and 1). The underlying distribution is the *binomial distribution*, whose parameter is p, the probability of success (or of seeing the outcome 1).

Our goal is to estimate $p|\mathbf{x}$.

Why are we talking about "logistic regression," and not "binomial regression"? It is because the conventional choice for the link function $g(p|\mathbf{x})$ is the *logit* function:

$$\log\left[\frac{E[Y|\mathbf{x}]}{1-E[Y|\mathbf{x}]}\right] = \beta_0 + \sum_{i=1}^{p}\beta_i x_i\,,$$

so that

$$E[Y|\mathbf{x}] = p|\mathbf{x} = \frac{e^{\beta_0+\sum_{i=1}^{p}\beta_i x_i}}{1+e^{\beta_0+\sum_{i=1}^{p}\beta_i x_i}}\,.$$

In logistic regression, the likelihood function to be optimized is

$$\mathcal{L} = \left(\prod_{i:Y_i=1} p_i\right)\left(\prod_{i:Y_i=0}(1-p_i)\right)\,.$$

Logistic Regression: Inference

Because we utilize a link function, inference is a bit different for logistic regression. In logistic regression, we utilize the concept of "odds."

Let's assume we have one predictor variable, and let's suppose that the predicted response is $p|x = 0.8$. That means that we expect that if we were to repeatedly sample response values at x, we would expect class 1 to be sampled four times as often as class 0:

$$O = \frac{E[Y|x]}{1 - E[Y|x]} = \frac{p}{1-p} = \frac{0.8}{1-0.8} = 4 = e^{\beta_0 + \beta_1 x}\,.$$

Thus we say that for the particular value x, the odds O are 4 (or 4-1 in favor of class 1).

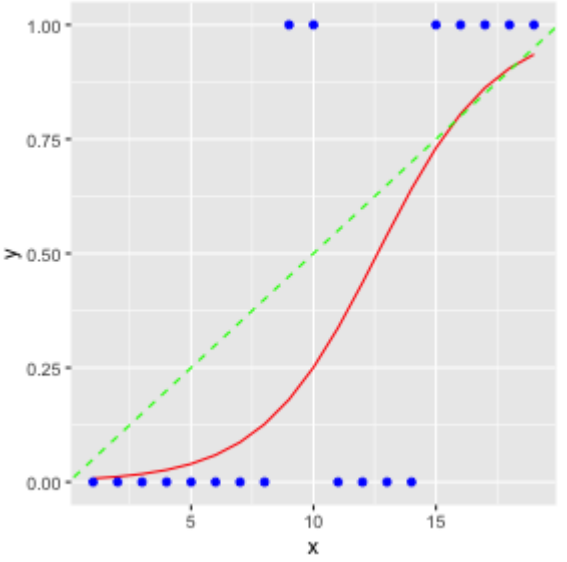
How does the odds change if I change the value of a predictor variable by one unit?

$$O_{\text{new}} = e^{\beta_0 + \beta_1 (x+1)} = e^{\beta_0 + \beta_1 x} e^{\beta_1} = e^{\beta_1} O_{\text{old}}\,.$$

For every unit change in x, the odds changes by a factor e^{β_1}. Values of β_1 far from 0 mean that the odds change more quickly with x than for values of β_0 close to zero.

Logistic Regression: Output

```
set.seed(101)
x = 1:19
y = rbinom(length(x),1,0.05*x)
out.log = glm(y~x,family=binomial)
suppressMessages(library(tidyverse))
ggplot(data=data.frame(x=x,y=out.log$fitted.values),mapping=aes(x=x,y=y)) + geom_line(color="red") +
  geom_point(data=data.frame(x=x,y=y),mapping=aes(x=x,y=y),color="blue") +
  geom_abline(slope=0.05,intercept=0,color="green",linetype="dashed")
```



Here, the true $p|x$ is given by the green dashed line, while the estimated $p|x$ is given by the red line. The takeaway point is that a logistic function is "only so flexible" and may not replicate the truth.

Logistic Regression: Output

```
summary(out.log)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4315  -0.4736  -0.1882   0.4954   1.8504
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.2800     2.3424  -2.254   0.0242 *
## x             0.4186     0.1843   2.271   0.0231 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 25.008  on 18  degrees of freedom
## Residual deviance: 14.236  on 17  degrees of freedom
## AIC: 18.236
##
## Number of Fisher Scoring iterations: 5
```

```
logLik(out.log)   # the maximum log-likelihood value
```

```
## 'log Lik.' -7.117803 (df=2)
```

Logistic Regression: Output

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.4315  -0.4736  -0.1882   0.4954   1.8504
```

The deviance residuals are, for each datum,

$$d_i = \text{sign}(y_i - \hat{p}_i)\sqrt{-2[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)]}\,,$$

where y_i is the i^{th} observed response and $\hat{p}_i$ is the estimated probability of success (i.e., the amplitude of the prediction curve for the i^{th} datum). The sum of squares of the deviance residuals is $-2 \log \mathcal{L}$.

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -5.2800     2.3424  -2.254   0.0242 *
x             0.4186     0.1843   2.271   0.0231 *
```

The intercept of the prediction curve is $e^{-5.28}/(1 + e^{-5.28}) = 0.005$ and $O_{\text{new}}/O_{\text{old}} = e^{0.4186}$.

```
    Null deviance: 25.008  on 18  degrees of freedom
Residual deviance: 14.236  on 17  degrees of freedom
AIC: 18.236
...
'log Lik.' -7.117803 (df=2)
```

The maximum value of the log of the likelihood function is -7.118. The residual deviance is -2 times -7.118, or 14.236. The AIC is $2k - 2 \log \mathcal{L}$ = $2 \cdot 2 - 2 \cdot (-7.118)$ = 18.236, where k is the number of degrees of freedom (here, `df` = 2). These are all metrics of quality of fit of the model.

Logistic Regression: Predictions

In this example, there was no training/testing split! In "real" analyses, there would be...you'd run on the model and generate test-set predictions via, e.g.,

```
resp.prob = predict(out.log,newdata=pred.test,type="response")
resp.pred = rep(NA,length(resp.prob))
for ( ii in 1:length(resp.prob) ) {
  if (resp.prob[ii] > 0.5) {
    resp.pred[ii] = "<class 1>"    # fill in name of class 1
  } else {
    resp.pred[ii] = "<class 0>"    # fill in name of class 0
  }
}
```

(Note the quotation marks. *I'm assuming you are dealing with a factor variable*, whose values you refer to by [quoted] name.) `resp.prob` is a number between 0 and 1. If that number is less than 0.5, we predict that the test datum is associated with class 0, otherwise we predict it is associated with class 1. In a future lecture, we will re-examine our use of 0.5 as a threshold for class splitting.

Model Diagnostics: Classification

The most straightforward diagnostic to use to assess logistic regression or any other classification model is the *confusion matrix*, whose rows are predicted classes and whose columns are observed classes. To create a confusion matrix and compute a misclassification rate, you can do the following:

```
resp.prob = predict(out.log,newdata=pred.test,type="response")   # same as on the last slide
resp.pred = ifelse(resp.prob>0.5,"<class 1>","<class 0>")        # compressed version of if statement on last slide
mean(resp.pred!=resp.test)                                       # compressed MCR calculator
table(resp.pred,resp.test)                                       # table with predicted rows and observed columns
```

Here's an example of a confusion matrix:

```
           class.test
class.pred QSO STAR
      QSO  129   39
      STAR  28  104
```

There are *many* metrics associated with confusion matrices. The misclassification rate, or MCR, is the ratio of the sum of the off-diagonal values in the confusion matrix (top right and bottom left) to the overall table sum. (For the confusion matrix above, the MCR is 0.223.) Other metrics include the *sensitivity* and *specificity*, etc.; for definitions of these and other metrics, see, e.g., this web page.

We will expand upon classification diagnostics in a future lecture.

Logistic Regression: Best-Subset Selection

Best-subset selection proceeds in a similar manner to how it does for linear regression; the primary difference is that we pass the argument `family=binomial` to `bestglm()`. Recall that we have to put the predictors and response into a single data frame, whose last column contains the response and is called `y`. Here, let's assume the data frame is called `df.train`.

```
library(bestglm)
bg.out = bestglm(df.train,family=binomial,IC="BIC")
bg.out$BestModel
```

Recall that `BestModel` can be treated like the output from `lm()` or `glm()`: you can create predictions and create misclassification matrices in the exact same manner as we did above with the output from `glm()`.