```r
library(car)  #for Anova function
```

```
## Loading required package: carData
```

Logistic Regression

Review of Normal Linear Regression

$$Y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i$$

$$where\ \epsilon_i\ for\ i = 1, ...n\ are\ independent$$

$\beta_0, \ldots, \beta_p$ are regression parameters

since $E(\epsilon_i) = 0$:

$$E(Y_i) = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$$

MLE:

$$L(\pi_1, ..., \pi_n | y_1, ..., y_n) = P(Y_1 = y_1) \times ... \times P(Y_n = y_n)$$

$$= \prod_{i=1}^{n} P(Y_i = y_i)$$

$$= \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^(1 - y_i)$$

But this would only give us $\hat{\pi}_i = 0\ or\ 1$ so it's pretty much useless

Instead we use the logistic regression model:

Logistic Regression Formula

$$\pi_i = \frac{exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}}{1 + exp(\beta_0 + B_1 x_{i1} + ... + \beta_p x_{ip}}$$

Or, after some algebra… we call this the $logit\ transformation$:

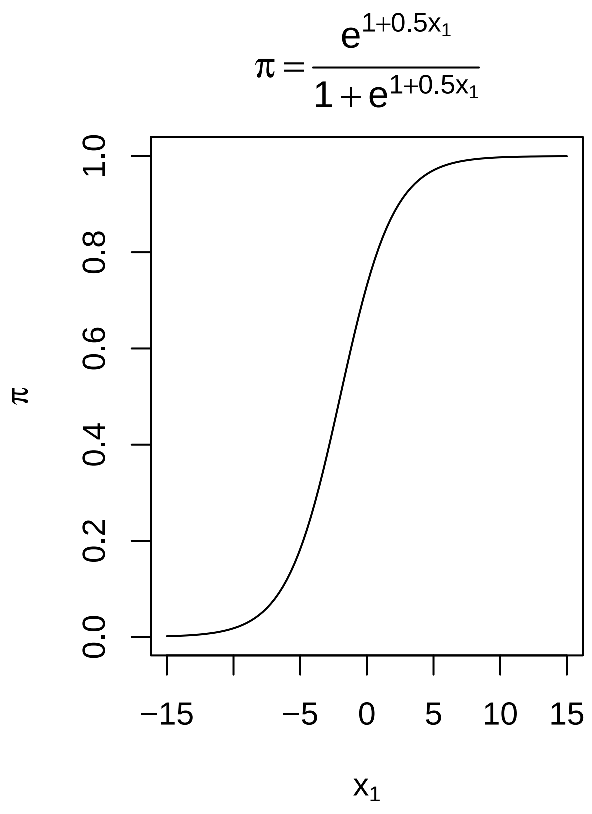
$$log(\frac{\pi_i}{1 - \pi_i}) = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$$

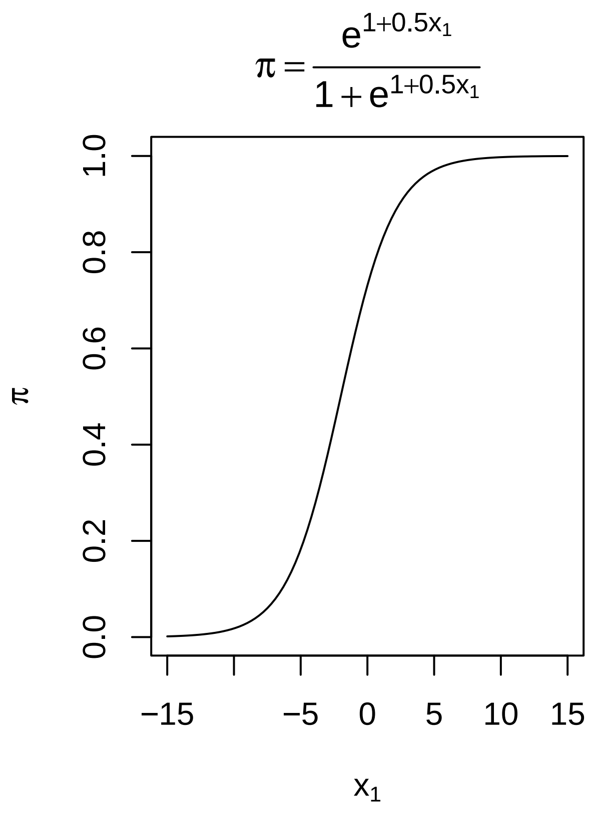
Notice that the LHS is the natural logarithm of the odds of success.

Need to use iteratively reweighted least squares (IRLS) to find the parameter estimates. Basically, the weights are shifted over and over until the likelihood converges to its maximum value. To do so, we use R's glm() function and optim() function. Implementation shown later.

Plot of logistic regression model

```
par(mfrow = c(1, 2), mar = c(5, 4, 4, 1))  #prepare plot with 1 row 2 columns 'makeframebyrow', mar par
beta0 <- 1
beta1 <- 0.5

curve(expr = exp(beta0 + beta1 * x)/(1 + exp(beta0 + beta1 *
    x)), xlim = c(-15, 15), col = "black", main = expression(pi ==
    frac(e^{
        1 + 0.5 * x[1]
    }, 1 + e^{
        1 + 0.5 * x[1]
    })), xlab = expression(x[1]), ylab = expression(pi))

beta <- -0.5
curve(expr = exp(beta0 + beta1 * x)/(1 + exp(beta0 + beta1 *
    x)), xlim = c(-15, 15), col = "black", main = expression(pi ==
    frac(e^{
        1 + 0.5 * x[1]
    }, 1 + e^{
        1 + 0.5 * x[1]
    })), xlab = expression(x[1]), ylab = expression(pi))
```





Example of Logistic Regression

Data consisting of a bunch of independent field kicks in football. The "good" variable is our target variable that we want to estimate. 1 = successful field kick, 0 = failed field kick.

```r
placekick <- read.table(file = "./Placekick.csv", header = TRUE,
    sep = ",")
head(placekick)
```

```
##   week distance change  elap30 PAT type field wind good
## 1    1       21      1 24.7167   0    1     1    0    1
## 2    1       21      0 15.8500   0    1     1    0    1
## 3    1       20      0  0.4500   1    1     1    0    1
## 4    1       28      0 13.5500   0    1     1    0    1
## 5    1       20      0 21.8667   1    0     0    0    1
## 6    1       25      0 17.6833   0    0     0    0    1
```

We're only going to focus on using one variable for now, "distance", which is how far the kicker was from the field goal.

Thus:

$$logit(\pi) = \beta_0 + \beta_1 x_1$$

where x1 = distance

```r
mod.fit <- glm(formula = good ~ distance, family = binomial(link = logit),
    data = placekick)  #glm = generalized linear model
mod.fit$coefficients
```

```
## (Intercept)    distance
##   5.8120798  -0.1150267
```

```r
# more detailed summary of mod.fit
summary(mod.fit)
```

```
##
## Call:
## glm(formula = good ~ distance, family = binomial(link = logit),
##     data = placekick)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7441   0.2425   0.2425   0.3801   1.6092
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.812080   0.326277   17.81   <2e-16 ***
## distance    -0.115027   0.008339  -13.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1013.43  on 1424  degrees of freedom
## Residual deviance:  775.75  on 1423  degrees of freedom
## AIC: 779.75
##
## Number of Fisher Scoring iterations: 6
```

```r
# The last line of this outputing # of fisher scoring
# iterations tells us how many iterations it took for the
# logit function to reach convergence. In this case, it
# took 6 iterations
```

Example of Logistic Regression W/ 2 Explanatory Variables

```
mod.fit2 <- glm(formula = good ~ change + distance, family = binomial(link = logit),
    data = placekick)
mod.fit2$coefficients
```

Using glm()

```
## (Intercept)      change    distance
##   5.8931814  -0.4477832  -0.1128888
```

In equation form, mod.fit2 is:

$$logit(\hat{\pi}) = 5.8932 - 0.4478 change - 0.1129 distance$$

```
vcov(mod.fit2)  #covariance matrix. different from the hessian matrix
```

```
##              (Intercept)        change      distance
## (Intercept)  0.111011379 -0.0094878323 -2.625598e-03
## change      -0.009487832  0.0375091687 -1.311512e-04
## distance    -0.002625598 -0.0001311512  7.129494e-05
```

```
# to get this covariance matrix, you first get the hessian
# matrix, log it and take the inverse and multiply by -1
```

Example of obtaining the estimated covariance matrix for the regression parameter estimates:

```
round(summary(mod.fit)$coefficients, 4)
```

```
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   5.8121     0.3263  17.8133        0
## distance     -0.1150     0.0083 -13.7937        0
```

```
vcov(mod.fit)  #covariance matrix
```

```
##             (Intercept)      distance
## (Intercept)  0.10645675 -2.606250e-03
## distance    -0.00260625  6.953996e-05
```

```
vcov(mod.fit)[2, 2]  #var-hat(beta-hat_1)
```

```
## [1] 6.953996e-05
```

```
summary(mod.fit)$coefficients[2, 2]^2  #same as the line above because we're squaring the std.error, wh
```

```
## [1] 6.953996e-05
```

Actual matrix calculation of covariance matrix using $(X'VX)^-1$

```
pi.hat <- mod.fit$fitted.values
V <- diag(pi.hat * (1 - pi.hat))  #getting only diagonal of hte matrix
X <- cbind(1, placekick$distance)
solve(t(X) %*% V %*% X)  # t() is to find transpose %*% for matrix multiplication, solve for inverse
```

```
##             [,1]          [,2]
## [1,]  0.10645678 -2.606250e-03
## [2,] -0.00260625  6.953997e-05
```

```
# this returns the same thing as the vcov function
```

Using likelihood function We can use both glm() and log-likelihood function to fit our model. Log-likelihood is a more general approach that will be useful later. Here's an example of how to use the log-likelihood function instead of the glm() to estimate a logistic regression model:

```
# setting up the function
logL <- function(beta, x, Y) {
    pi <- exp(beta[1] + beta[2] * x)/(1 + exp(beta[1] + beta[2] *
        x))  #pi for logistic regression
    sum(Y * log(pi) + (1 - Y) * log(1 - pi))
}

logL(beta = mod.fit$coefficients, x = placekick$distance, Y = placekick$good)
```

```
## [1] -387.8725
```

```
logLik(mod.fit)  #Automatic extraction of MLE, good to see that it matches up with our prior glm() func
```

```
## 'log Lik.' -387.8725 (df=2)
```

Now to optimize our estimates with optim():

```
# Find the starting values for parameter estimates
reg.mod <- lm(formula = good ~ distance, data = placekick)
reg.mod$coefficients
```

```
## (Intercept)    distance
##  1.25202444 -0.01330212
```

```
mod.fit.optim <- optim(par = reg.mod$coefficients, fn = logL,
    hessian = TRUE, x = placekick$distance, Y = placekick$good,
    control = list(fnscale = -1), method = "BFGS")
# since we want to maximize logL, we have control =
# list(fnscale = -1) which tells the function to minimize
# the negative of logL. optim() default is to minimize a
# function so we need to do this

names(mod.fit.optim)
```

```
## [1] "par"         "value"       "counts"      "convergence" "message"
## [6] "hessian"
```

```
mod.fit.optim$par
```

```
## (Intercept)    distance
##   5.8112544  -0.1150046
```

```
mod.fit.optim$value  #maximum value of the function
```

```
## [1] -387.8725
```

```
mod.fit.optim$convergence  #0 means convergence was achieved
```

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

```
-solve(mod.fit.optim$hessian)  #covariance matrix
```

```
##              (Intercept)      distance
## (Intercept)  0.106482867 -2.607258e-03
## distance    -0.002607258  6.957463e-05
```

Alternative: Logistic Regression w/ Multiple Trials

Let's say we have J trials each with their own different weights for their explanatory variables. Then the log-likelihood function is:

$$log[L(\beta_0, ..., \beta_p|w_1, ..., w_j)] = \Sigma_{j=1}^{J} log[(\binom{n_j}{x_j})] + w_j log(\pi_j) + (n_j - w_j) log(1 - \pi_j)$$

Because $\binom{n_j}{x_j}$ is a constant, the estimated parameter values do not change. So the MLEs are the same for every binary response

Example:

```
w <- aggregate(x = good ~ distance, data = placekick, FUN = sum)

n <- aggregate(x = good ~ distance, data = placekick, FUN = length)

w.n <- data.frame(distance = w$distance, success = w$good, trials = n$good,
    proportion = round(w$good/n$good, 4))
head(w.n)  #now a binomial variable
```

```
##   distance success trials proportion
## 1       18       2      3     0.6667
## 2       19       7      7     1.0000
## 3       20     776    789     0.9835
## 4       21      19     20     0.9500
## 5       22      12     14     0.8571
## 6       23      26     27     0.9630
```

```
mod.fit.bin <- glm(formula = success/trials ~ distance, weights = trials,
    family = binomial(link = logit), data = w.n)
summary(mod.fit.bin)
```

```
##
## Call:
## glm(formula = success/trials ~ distance, family = binomial(link = logit),
##     data = w.n, weights = trials)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0373  -0.6449  -0.1424   0.5004   2.2758
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.812080   0.326277   17.81   <2e-16 ***
## distance    -0.115027   0.008339  -13.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 282.181  on 42  degrees of freedom
## Residual deviance:  44.499  on 41  degrees of freedom
## AIC: 148.46
##
## Number of Fisher Scoring iterations: 5
```

Hypothesis Tests for Regression Parameters

We test if $H_0 : \beta_r = 0$ vs. $H_1 : \beta_r \neq 0$. If the hypothesis is not rejected, the r^{th} explanatory variable is not included in the logit model.

We test by either Wald's test or the likelihood ratio test. LRT usually performs better because of what we learned about true vs stated tests.

Wald test:

$$Z_0 = \frac{\hat{\beta}_r}{\sqrt{\hat{Var}(\hat{\beta}_r)}}$$

LRT:

$$\Lambda = \frac{MLE\ under\ H_0}{MLE\ under\ H_0\ or\ H_A}$$

We prefer to convert it to -2log(Delta)… so:

$$-2log(\Lambda) = -2\Sigma_{i=1}^{n} y_i log\left(\frac{\hat{\pi}_i^{(0)}}{\hat{\pi}_i^{(a)}}\right) + (1 - y_i) log\left(\frac{1 - \hat{\pi}_i^{(0)}}{1 - \hat{\pi}_i^{(a)}}\right)$$

Source: Trust me bro. It's easy to do in R with anova() and Anova(). Here's how:

Imagine we're using a model $logit(\pi) = \beta_0 + \beta_1 change + \beta_2 distance$

It's results are saved above in mod.fit2

Wald R Code

```
summary(mod.fit2)
```

```
##
## Call:
## glm(formula = good ~ change + distance, family = binomial(link = logit),
##     data = placekick)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7061   0.2282   0.2282   0.3750   1.5649
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.893181   0.333184  17.687   <2e-16 ***
## change      -0.447783   0.193673  -2.312   0.0208 *
## distance    -0.112889   0.008444 -13.370   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 1013.4  on 1424  degrees of freedom
## Residual deviance:  770.5  on 1422  degrees of freedom
## AIC: 776.5
##
## Number of Fisher Scoring iterations: 6
```

```
# Includes Wald's test. Z_0 = -0.447/0/1936 = -2.32124 (the
# Z value column) Since prob is less than 0.05, we don't
# reject it

# Thus, we say there is marginal evidence that change is
# important to include in the model GIVEN that distance is
# in the model. You have to be sure to state all the other
# GIVEN variables in the model if you say this statement.

# Similary, we say that there is strong evidence the
# importance of distance given that change is in the model
```

LRT Code

```
Anova(mod.fit2, test = "LR")
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: good
##          LR Chisq Df Pr(>Chisq)
## change      5.246  1      0.022 *
## distance  218.650  1     <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Kinda the same thing as Wald except you're now using a
# Chisq distribution
```

similar way for LRT. Only difference is that it's sequential:

```
anova(mod.fit2, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: good
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                      1424    1013.43
## change    1   24.277      1423     989.15 8.343e-07 ***
## distance  1  218.650      1422     770.50 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

little anova() will be useful later when you compare specific models. Example:

```
mod.fit.Ho <- glm(formula = good ~ distance, family = binomial(link = logit),
    data = placekick)
```

```
anova(mod.fit.Ho, mod.fit2, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: good ~ distance
## Model 2: good ~ change + distance
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      1423     775.75
## 2      1422     770.50  1   5.2455    0.022 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again, LRT is great when functions like anova and Anova arent available. Generally, this won't be the case for logistic regression applications, but it's helpful. Here's an example for LRT test of $H_0 : logit(\pi) = \beta_0$ vs. $H_a : logit(\pi) = \beta_0 + \beta_1 change$

```
mod.fit.Ho <- glm(formula = good ~ 1, family = binomial(link = logit),
    data = placekick)
mod.fit.Ha <- glm(formula = good ~ change, family = binomial(link = logit),
    data = placekick)
anova(mod.fit.Ho, mod.fit.Ha, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: good ~ 1
## Model 2: good ~ change
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1      1424    1013.43
## 2      1423     989.15  1   24.277 8.343e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
pi.hat.Ho <- mod.fit.Ho$fitted.values
pi.hat.Ha <- mod.fit.Ha$fitted.values
y <- placekick$good

stat <- -2 * sum(y * log(pi.hat.Ho/pi.hat.Ha) + (1 - y) * log((1 -
    pi.hat.Ho)/(1 - pi.hat.Ha)))  #-logLRT formula to find chisq values
pvalue <- 1 - pchisq(q = stat, df = 1)  #getting pvalue from Chisq distribution
data.frame(stat, pvalue)  #same as above, just using LRT
```

```
##       stat       pvalue
## 1 24.27703 8.342813e-07
```

Deviance Vocabulary **Deviance** is the amount one model deviates from another measured by the transformed LRT statistic $-2log(\Lambda)$. For example, when we tested for change, we got 5.246. This basically means the estimated probability of success for the model excluding **change** deviates from those that include **change**.

Residual deviance is how the observed proportion of success differs from the model of interest

Null deviance is how the probabilities estimated from the null model deviates from the observed proportion of success. Null model : $\large logit(\pi_i) = \beta_0$. Since this only contains the intercept term, π_i will always be the same value, the MLE.

Usually the **residual deviance** is an intermediate step for performing LRT to compare two models. Example:

$$H_0 : logit(\pi^{(0)}) = \beta_0 + \beta_1 x_1$$

$$H_a : logit(\pi^{(a)}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

Residual deviance for Ho:

$$-2log(\Lambda) = -2\Sigma_{i=1}^{n} y_i log\left(\frac{\hat{\pi}_i^{(0)}}{y_i}\right) + (1-y_i)log\left(\frac{1-\hat{\pi}_i^{(0)}}{1-y_i}\right)$$

Residual deviance for Ha:

$$-2log(\Lambda) = -2\Sigma_{i=1}^{n} y_i log\left(\frac{\hat{\pi}_i^{(a)}}{y_i}\right) + (1-y_i)log\left(\frac{1-\hat{\pi}_i^{(a)}}{1-y_i}\right)$$

Funny enough, if we subtract the residual deviance of Ha by the residual deviance for Ho, we get the LRT function.

Residual deviance for binomial responses W_j for $j = 1, \ldots, J$.

$$-2log(\Lambda) = -2\Sigma_{j=1}^{J}\left[w_j log\left(\frac{\hat{\pi}_j}{w_j/n_j}\right) + (n_j - w_j)log\left(frac1 - \hat{\pi}_j 1 - w_j/n_j\right)\right]$$

R Example of testing for WITHOUT using anova()

$$H_0 : logit(\pi^{(0)}) = \beta_0 + \beta_1 distance$$

$$H_a : logit(\pi^{(a)}) = \beta_0 + \beta_1 change + \beta_2 distance$$

```r
mod.fit.Ho <- glm(formula = good ~ distance, family = binomial(link = logit),
    data = placekick)
df <- mod.fit.Ho$df.residual - mod.fit2$df.residual

stat <- mod.fit.Ho$deviance - mod.fit2$deviance
pvalue <- 1 - pchisq(q = stat, df = df)

data.frame(Ho.resid.dev = mod.fit.Ho$deviance, Ha.resid.dev = mod.fit2$deviance,
    df = df, stat = round(stat, 4), pvalue = round(pvalue, 4))
```

```
##   Ho.resid.dev Ha.resid.dev df   stat pvalue
## 1      775.745     770.4995  1 5.2455  0.022
```

RECAP: Lots of different ways to perform LRT. Generally, use Anova() and anova(), which are easiest to use.

Odds ratio

$$OR = \frac{Odds_{x+c}}{Odds_x} = \frac{exp(\beta_0 + \beta_1(x+c))}{exp(B_0 + B_1 x)} = exp(c\beta_1)$$

where c > 0 and is the increase of a variable holding all other variables constant.

$$\widehat{OR} = exp(c\hat{\beta}_1)$$

Since this is an estimated odds ratio, we have to make CI to make inferences with a level of confidence.

Wald CI when n >= 40:

First find $\widehat{Var}(\hat{\beta}_1)$ first from estimated covariance matrix. Then:

$$exp\left(c\hat{\beta}_1 \pm cZ_{1-a}\sqrt{\widehat{Var}(\hat{\beta}_1)}\right)$$

Inside the square root should be Var(cB1), but property of variance let us square it and bring it out.

So interpretation should be:
"With (1-a)100% confidence, the odds of a success change by an amount between"lower" to "upper" times for every c-unit increase in x."

Likelihood Ratio (LR) when n < 40:

$$-2log\left(\frac{L(\tilde{\beta}_0, \beta_1|y_1, ..., y_n)}{L(\hat{\beta}_0, \hat{\beta}_1|y_1, ..., y_n)}\right) < \chi^2_{1,1-a}$$

where $\tilde{\beta}_0$ is the MLE of β_0. You also gotta use some iterative numerical procedure to find lower and upper limits. After limits are found, we have to do this:

$$exp(c \times lower) < OR < exp(c \times upper)$$

```
# Odds ratio
exp(mod.fit$coefficients[2])
```

```
##  distance
## 0.8913424
```

```
exp(-10 * mod.fit$coefficients[2])  #when distance coefficient decreases by -10, the odds of a success
```

```
## distance
## 3.159035
```

```

beta.ci <- confint(object = mod.fit, parm = "distance", level = 0.95)  #automatically finds CI with LR
```

LR Code

```
## Waiting for profiling to be done...
```

```
beta.ci
```

```
##       2.5 %      97.5 %
## -0.13181435 -0.09907103
```

```
rev(exp(-10 * beta.ci))  #OR C.I. for c = -10. #remember we have to multiply the limits by c and take t
```

```
##   97.5 %    2.5 %
## 2.693147 3.736478
```

```
# remove lables with as.numeric()
as.numeric(rev(exp(-10 * beta.ci)))
```

```
## [1] 2.693147 3.736478
```

With the code above, we say that with 95% confidence, the odds increase by between 2.69 and 3.73 when distance is decreased by -10 yards.

```
beta.ci <- confint.default(object = mod.fit, parm = "distance",
    level = 0.95)
beta.ci
```

Wald CI Code

```
##               2.5 %     97.5 %
## distance -0.1313709 -0.0986824
```

```
rev(1/exp(beta.ci * 10))  #invert OR C.I. for c = 10. We invert cuz it's OR is less than one
```

```
## [1] 2.682701 3.719946
```

```
beta.ci <- mod.fit$coefficients[2] + qnorm(p = c(0.025, 0.975)) *
    sqrt(vcov(mod.fit)[2, 2])
beta.ci
```

Wald CI Code By Applying Formula (Hard Way):

```
## [1] -0.1313709 -0.0986824
```

```
rev(1/exp(beta.ci * 10))
```

```
## [1] 2.682701 3.719946
```

```
# Examples of how to find profile likelihood ratio
# intervals without confint()

# Example of how to estimate the model logit(pi) = beta~_0
# + beta_1*distance where beta_1*x is held constant and
# beta_1 = -0.12.  The offset() function instructs R to not
# estimate a coefficient for beta1*x1 in the model (treat
# it as a constant).  Because there is only beta_0
# remaining in the model, we need to use the '1' to tell R
# to estimate beta_0.
mod.fit.ex <- glm(formula = good ~ 1 + offset(-0.12 * distance),
    family = binomial(link = logit), data = placekick)
mod.fit.ex$coefficients
```

LR Code Without confint(). Super difficult and unecessary

```
## (Intercept)
##    5.999299
```

```
logLik(mod.fit.ex)

## 'log Lik.' -388.0476 (df=1)

as.numeric(-2 * (logLik(mod.fit.ex) - logLik(mod.fit)) - qchisq(p = 0.95,
    df = 1))

## [1] -3.491277

######################## EXAMPLE using uniroot() to find
######################## the profile LR interval

# Calculate -2log(Lambda) - 3.84
find.root <- function(beta1, data.set, logLik.denom) {
    mod.fit.temp <- glm(formula = good ~ 1 + offset(beta1 * distance),
        family = binomial(link = logit), data = data.set)
    as.numeric(-2 * (logLik(mod.fit.temp) - logLik.denom) - qchisq(p = 0.95,
        df = 1))
}

# Test
find.root(beta1 = -0.12, data.set = placekick, logLik.denom = logLik(mod.fit))

## [1] -3.491277

find.root(beta1 = -0.1318144, data.set = placekick, logLik.denom = logLik(mod.fit))  # Bound from confi

## [1] 0.0002722009

find.root(beta1 = -0.09907103, data.set = placekick, logLik.denom = logLik(mod.fit))  # Bound from conf

## [1] -0.0002323373

# Use uniroot
save.lower <- uniroot(f = find.root, interval = c(-0.15, mod.fit$coefficients[2]),
    data.set = placekick, logLik.denom = logLik(mod.fit))
save.upper <- uniroot(f = find.root, interval = c(mod.fit$coefficients[2],
    -0.05), data.set = placekick, logLik.denom = logLik(mod.fit))
save.lower

## $root
## [1] -0.1318151
##
## $f.root
## [1] 0.000577683
##
## $iter
## [1] 6
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05

save.upper

## $root
## [1] -0.09907579
```

```
##
## $f.root
## [1] -0.002576836
##
## $iter
## [1] 8
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

```
# OR interval
round(1/c(exp(10 * save.upper$root), exp(10 * save.lower$root)),
    4)
```

```
## [1] 2.6933 3.7365
```

Probability of success

We now estimate π and make inferences about our estimate with CI's:

$$\hat{\pi} = \frac{exp(\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_p x_p)}{1 + exp(\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_p x_p)}$$

Wald

$$\hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-a/2}\sqrt{\widehat{Var}(\hat{\beta}_0 + \hat{\beta}_1 x)}$$

Estimated variance:

$$\widehat{Var}(\hat{\beta}_0 + \hat{\beta}_1 x) = \widehat{Var}(\hat{\beta}_0) + x^2\widehat{Var}(\hat{\beta}_1) + 2x\widehat{Cov}(\hat{\beta}_0, \hat{\beta}_1)$$

and then you transform it to fit logistic regression by the familiar exp(.)/[1+exp(.)]. **This is the Wald's CI for success.**

$$\frac{exp\left(\hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-a/2}\sqrt{\widehat{Var}(\hat{\beta}_0 + \hat{\beta}_1 x)}\right)}{1 + exp\left(\hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-a/2}\sqrt{\widehat{Var}(\hat{\beta}_0 + \hat{\beta}_1 x)}\right)}$$

```
install.packages("mcprofile")
```

LR. Use mcprofile package.

```
## Installing package into '/opt/r'
## (as 'lib' is unspecified)
```

```
library("mcprofile")
```

```
## Loading required package: ggplot2
```

Basically LR but its harder now czu you're maximizing with bunch of different variables. Iterative numerical procedures take a long time. BUT if you do it that way, you have to transform the Ci with exp(.)/[1+exp(.)] like the Wald interval.

R code to estimate π

```
linear.pred <- mod.fit$coefficients[1] + mod.fit$coefficients[2] *
    20
linear.pred
```

```
## (Intercept)
##    3.511547
```

```
as.numeric(exp(linear.pred)/(1 + exp(linear.pred)))
```

```
## [1] 0.9710145
```

```
# Or, the recommended way
predict.data <- data.frame(distance = 20)
predict(object = mod.fit, newdata = predict.data, type = "link")  #basically using glm() method from mo
```

```
##        1
## 3.511547
```

```
predict(object = mod.fit, newdata = predict.data, type = "response")
```

```
##         1
## 0.9710145
```

R code for Wald Interval for $\hat{\pi}$

```
alpha <- 0.05
linear.pred <- predict(object = mod.fit, newdata = predict.data,
    type = "link", se = TRUE)
linear.pred
```

```
## $fit
##        1
## 3.511547
##
## $se.fit
## [1] 0.1732707
##
## $residual.scale
## [1] 1
```

```
pi.hat <- exp(linear.pred$fit)/(1 + exp(linear.pred$fit))
CI.lin.pred <- linear.pred$fit + qnorm(p = c(alpha/2, 1 - alpha/2)) *
    linear.pred$se
CI.pi <- exp(CI.lin.pred)/(1 + exp(CI.lin.pred))
round(data.frame(predict.data, pi.hat, lower = CI.pi[1], upper = CI.pi[2]))
```

```
##   distance pi.hat lower upper
## 1       20      1     1     1
```

R code for LR Interval for $\hat{\pi}$ for more than one distance.

```
library(package = mcprofile)
K <- matrix(data = c(1, 20), nrow = 1, ncol = 2)  #B_0 = 1, B_1 = 20
K
```

```
##      [,1] [,2]
## [1,]    1   20
```

```r
# Calculate -2log(Lambda)
linear.combo <- mcprofile(object = mod.fit, CM = K)  #CM = contrast matrix
# CI for beta_0 + beta_1 *x
ci.logit.profile <- confint(object = linear.combo, level = 0.95)
ci.logit.profile
```

```
## 
##    mcprofile - Confidence Intervals
## 
## level:        0.95
## adjustment:   single-step
## 
##    Estimate lower upper
## C1     3.51  3.19  3.87
```

```r
names(ci.logit.profile)
```

```
## [1] "estimate"    "confint"     "CM"          "quant"       "alternative"
## [6] "level"       "adjust"
```

```r
exp(ci.logit.profile$confint)/(1 + exp(ci.logit.profile$confint))
```

```
##       lower    upper
## 1 0.9603165 0.979504
```

```r
exp(ci.logit.profile$estimate)/(1 + exp(ci.logit.profile$estimate))
```

```
##     Estimate
## C1 0.9710145
```

```r


head(w.n)
```

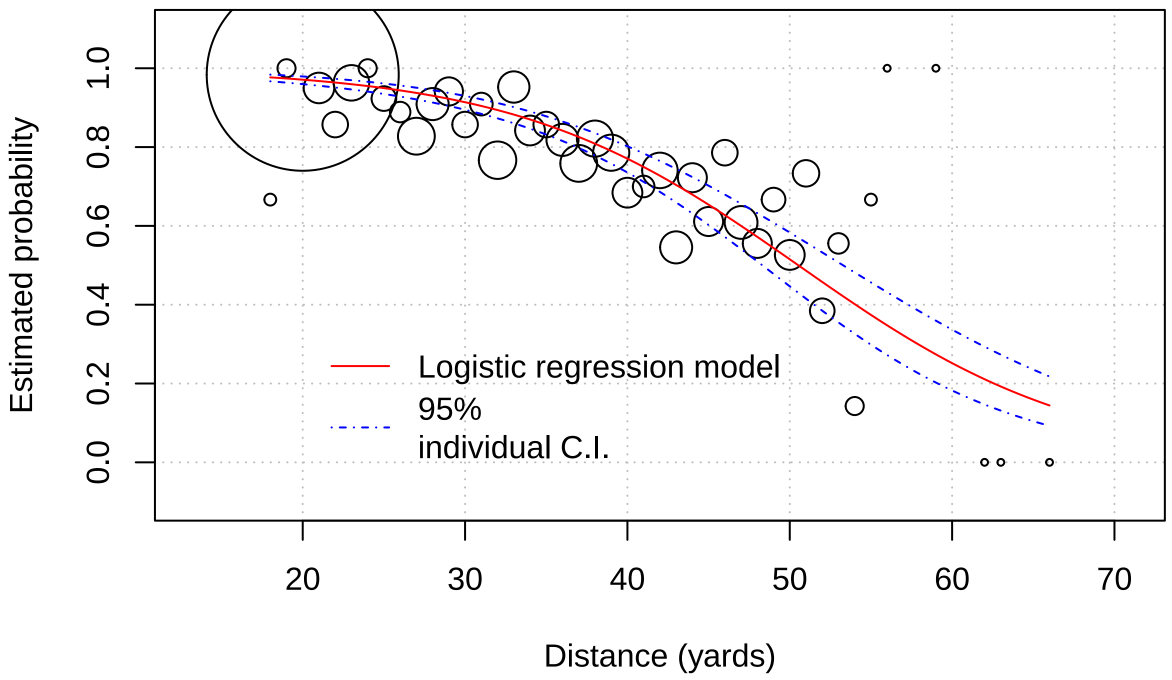
Plot of Wald Estimate and CI of success parameter

```
##   distance success trials proportion
## 1       18       2      3     0.6667
## 2       19       7      7     1.0000
## 3       20     776    789     0.9835
## 4       21      19     20     0.9500
## 5       22      12     14     0.8571
## 6       23      26     27     0.9630
```

```r
# Bubbles to show how many were sampled at a particular
# distance
symbols(x = w$distance, y = w$good/n$good, circle = sqrt(n$good),
    inches = 0.5, xlab = "Distance (yards)", ylab = "Estimated probability",
    panel.first = grid(col = "gray", lty = "dotted"))

curve(expr = predict(object = mod.fit, newdata = data.frame(distance = x),
    type = "response"), col = "red", add = TRUE, xlim = c(18,
    66))

# CI bands
ci.pi <- function(newdata, mod.fit.obj, alpha) {
    linear.pred <- predict(object = mod.fit.obj, newdata = newdata,
        type = "link", se = TRUE)
    CI.lin.pred.lower <- linear.pred$fit - qnorm(p = 1 - alpha/2) *
        linear.pred$se
    CI.lin.pred.upper <- linear.pred$fit + qnorm(p = 1 - alpha/2) *
        linear.pred$se
    CI.pi.lower <- exp(CI.lin.pred.lower)/(1 + exp(CI.lin.pred.lower))
    CI.pi.upper <- exp(CI.lin.pred.upper)/(1 + exp(CI.lin.pred.upper))
    list(lower = CI.pi.lower, upper = CI.pi.upper)
}

# test case
ci.pi(newdata = data.frame(distance = 20), mod.fit.obj = mod.fit,
    alpha = 0.05)
```

```
## $lower
##         1
## 0.9597647
##
## $upper
##         1
## 0.9791871
```

```
# Plot CI bands
curve(expr = ci.pi(newdata = data.frame(distance = x), mod.fit.obj = mod.fit,
    alpha = 0.05)$lower, col = "blue", lty = "dotdash", add = TRUE,
    xlim = c(18, 66))

curve(expr = ci.pi(newdata = data.frame(distance = x), mod.fit.obj = mod.fit,
    alpha = 0.05)$upper, col = "blue", lty = "dotdash", add = TRUE,
    xlim = c(18, 66))


# Legend
legend(x = 20, y = 0.4, legend = c("Logistic regression model",
    "95%
individual C.I."), lty = c("solid", "dotdash"), col = c("red",
    "blue"), bty = "n")
```



Yes, some are off, but not that many. There are only so many observations at a particular distance value so its hard for the model to fit well with those values. Our model isn't as bad as we think.

There is also a LR version for the plot thats similar. It will not be coded.