Likelihood Notes - Categorical Data

Michael Winton

June 3, 2018

Likelihood Function

Logit Function

$$log(Odds) = log(\frac{\pi}{1-\pi}) = logit(\pi) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$$

And its inverse:

$$\pi = \frac{exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)}{1 + exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)}$$

Maximum Likelihood and $\hat{\pi}$

Plug in our measured $x_{11...np}$ and $y_{1...n}$ values and maximize the log likelihood function:

$$log[L(\beta_0, ...\beta_p|y_1, ...y_n)] = log(\prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i})$$

$$= \sum_{i=1}^{n} y_i log(\pi_i) + (1-y_i) log(1-\pi_i)$$

After this, it gets pretty ugly, but we arrive at simultaneous equations to solve for $\beta_0...\beta_p$.

$$= \sum_{i=1}^{n} y_i log\left[\frac{exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip})}{1 + exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip})}\right] + (1-y_i) log\left[\frac{1}{1 + exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip})}\right]$$

$$= \sum_{i=1}^{n} y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) - log(1 + exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}))$$

Taking derivatives of these n simultaneous equations with respect to $\beta_0...\beta_p$, setting equal to zero, and solving, we get $\hat{\beta}_0...\hat{\beta}_p$. This typically has to be done numerically (IRLS).

Once we have estimators $\hat{\beta}_0...\hat{\beta}_p$, we can estimate π for any $\vec{x}$:

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

Note that the i subscript is gone. There's only one $\hat{\pi}$ equation for our entire sample.

Calculating log likelihood

```r
# load Bilder's dataset
placekick <- read.csv("placekick.csv")
fit_dist <- glm(formula=good ~ distance, family=binomial(link=logit), data=placekick)

logL <- function(beta, x, y) {
  # only works for a single x feature
  beta0 <- beta[1]
  beta1 <- beta[2]
  pihat <- exp(beta0 + beta1 * x) / (1 + exp(beta0 + beta1 * x))
  sum(y * log(pihat) + (1 - y) * log(1 - pihat))
}

# manual calculation
logL(fit_dist$coefficients, placekick$distance, placekick$good)
```

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

```r
# built-in function
logLik(fit_dist)
```

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

Aside: Bernoulli (binary) or Binomial data give same results

```r
# working with data in binomial form
w <- aggregate(formula=good ~ distance, data=placekick, FUN=sum)
n <- aggregate(formula=good ~ distance, data=placekick, FUN=length)
placekick_binom <- data.frame(distance=w$distance, w=w$good, n=n$good, proportion=w$good/n$goo
fit_dist_binom <- glm(formula=proportion ~ distance, weights=n, family=binomial(link=logit), da
# note results are identical, whether bernoulli or binomial-formatted
fit_dist$coefficients
```

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

```r
fit_dist_binom$coefficients
```

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

Likelihood Ratio Test (LRT)

$$\Lambda = \frac{\text{Max Likelihood under } H_0}{\text{Max Likelihood under } H_0, H_{alt}}$$

It turns out that $-2log(\Lambda)$ has a convenient χ^2_q distribution, where q is the number of variables being tested in H_0, so that format is used more frequently. Reject H_0 when $-2log(\Lambda) > \chi^2_q$ critical value.

```
# for q=1, alpha=0.05, calculate critical value
qchisq(0.95, df=1)
```

```
## [1] 3.841459
```

```
# for q=1, that is the square of the critical value for standard normal dist!
qnorm(0.975)**2
```

```
## [1] 3.841459
```

For hypothesis tests comparing two models, use the `anova(...)` function:

```
fit_dist_change <- glm(formula=good ~ distance + change, family=binomial(link=logit), data=plac
# -2log(lambda) is reported as "Deviance" in the anova output
anova(fit_dist, fit_dist_change, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: good ~ distance
## Model 2: good ~ distance + change
##   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
```

```
# confirm the p-value
1-pchisq(5.2455, df=1)
```

```
## [1] 0.0220036
```

The ratio also simplifies nicely, so that once you have $\hat{\pi}$ for two models, it's easy to do a LRT:

$$-2log(\Lambda) = -2log\left[\frac{L(\hat{\beta}^{(0)}|y_1...y_n)}{L(\hat{\beta}^{(alt)}|y_1...y_n)}\right] = -2\Big(log[L(\hat{\beta}^{(0)}|y_1...y_n)] - log[L(\hat{\beta}^{(alt)}|y_1...y_n)]\Big)$$

$$= -2\sum_{i=1}^{n} y_i log(\hat{\pi}_i^{(0)}) + (1-y_i)log(1-\hat{\pi}_i^{(0)}) - y_i log(\hat{\pi}_i^{(alt)}) - (1-y_i)log(1-\hat{\pi}_i^{(alt)})$$

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

Let's manually calculate the LRT based on this equation:

```
linpred0 <- fit_dist$coefficients[1] + fit_dist$coefficients[2] * placekick$distance
linpreda <- fit_dist_change$coefficients[1] + fit_dist_change$coefficients[2] * placekick$dista
  fit_dist_change$coefficients[3] * placekick$change
pihat0 <- exp(linpred0) / (1+exp(linpred0))
pihata <-  exp(linpreda) / (1+exp(linpreda))
dev <- -2 * sum(placekick$good * log(pihat0/pihata) + (1-placekick$good) * log((1-pihat0)/(1-pi
# deviance = -2log(lambda)
dev
```

```
## [1] 5.245543
```

```
# p-value
1-pchisq(dev, df=1)
```

```
## [1] 0.02200306
```

Deviance

Deviance is simply how much two models differ, as measured by difference between their $-2log(\Lambda)$ statistic. A single model by itself cannot have a deviance, since by definition it is a ratio.

Residual Deviance measures how much the estimated probabilities of success ($\hat{\pi}_i$) deviate from the actual observed proportions of success (either Bernoulli or Binomial format). This is more formally defined as how much our model with $\beta_0 ... \beta_p$ parameters differs from a **fully saturated** model. A fully saturated model is defined as having one β parameter per observation, which is the max possible to estimate.

Residual deviance statistics are often used as an easy way to compare two models, because they're easy to generate with R. You have Residual Deviance$_{modelA}$: Observed Proportion of Successes and Residual Deviance$_{modelB}$: Observed Proportion of Successes, which allows you to easily get to Residual Deviance$_{modelA}$: Residual Deviance$_{modelB}$. This is easy to calculate using the equation above, or using the `anova(...)` function in R.

If data is in Bernoulli form, calculate this for each model:

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

If data is in Binomial form, use this variation for each model:

$$-2log(\Lambda) = -2\sum_{n=1}^{J} w_i log\left(\frac{\hat{\pi}_i^{(0)}}{w_j/n_j}\right) + (1-w_i)log\left(\frac{1-\hat{\pi}_i^{(0)}}{1-w_j/n_j}\right)$$

Null deviance is defined as the difference between a model with *only* an intercept term (ie. the grand mean): $logit(\pi_i) = \beta_0 \forall i = 1..n$ and the observed proportion of successes (ie. fully saturated model). This model says $\hat{\pi}$ is the same for all observations.

The difference between Residual Deviance and Null Deviance is a measure of whether your model has any more predictive value than just taking the grand mean of π.

```r
# fit_dist
# fit_dist_change
(resid_dev_fit_dist <- fit_dist$deviance)
```

```
## [1] 775.745
```

```r
(resid_dev_fit_dist_change <- fit_dist_change$deviance)
```

```
## [1] 770.4995
```

```r
# we know that the model with more variables always has lower Resid Deviance, and fewer df
(dev_between_models <- resid_dev_fit_dist - resid_dev_fit_dist_change)
```

```
## [1] 5.245543
```

```r
# we need degrees of freedom to calculate p-value (from ChiSq distr)
(df_fit_dist <- fit_dist$df.residual)
```

```
## [1] 1423
```

```r
(df_fit_dist_change <- fit_dist_change$df.residual)
```

```
## [1] 1422
```

```r
(df_delta <- df_fit_dist - df_fit_dist_change)
```

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

```r
# calculate p-value and display results
(p_value <- 1 - pchisq(dev_between_models, df=df_delta))
```

```
## [1] 0.02200306
```

```r
data.frame(H0_resid_dev=resid_dev_fit_dist, Ha_resid_dev=resid_dev_fit_dist_change, deviance=de
           df=df_delta, p_value=p_value)
```

```
##   H0_resid_dev Ha_resid_dev deviance df    p_value
## 1      775.745     770.4995 5.245543  1 0.02200306
```

```r
# confirm this matches anova results
anova(fit_dist, fit_dist_change, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: good ~ distance
## Model 2: good ~ distance + change
##   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
```

Odds Ratio and Confidence Interval

Pick an appropriate value for x (not always “1 unit”), and if there are no interaction or higher order terms, the Odds Ratio (OR) is simple:

$$OR = \frac{Odds_{x+c}}{Odds_x} = \frac{exp(\beta_0 + \beta_1(x_1 + c) + ... + \beta_p x_p)}{exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)} = exp(c\beta_1)$$

If we add an interaction term, we have to specify the OR for changes to x_1 *at a fixed value* for the term it interacts with (x_2):

$$OR = \frac{Odds_{x+c}}{Odds_x} = \frac{exp(\beta_0 + \beta_1(x_1 + c) + ... + \beta_p(x_1 + c)x_2)}{exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_1 x_2)} = exp(c\beta_1 + c\beta_p x_2)$$

Similarly, if we have higher order terms for x_1, we have to specify the OR *at a specific value* of x_1:

$$OR = \frac{Odds_{x+c}}{Odds_x} = \frac{exp(\beta_0 + \beta_1(x_1 + c) + ... + \beta_p(x_1 + c)^2)}{exp(\beta_0 + \beta_1 x_1 + ... + \beta_p x_1^2)} = exp(c\beta_1 + 2cx_1 + c^2)$$

Confidence Interval for OR

Wald CI only gives good estimates for large n, but it’s easy to calculate. Basically, just calculate the CI for the value inside the exponential term above (e.g. for no interactions, $c\beta_1$), and then exponentiate.

In general calculating CI with LRT is preferred over Wald. This is not simple algebra, though. It involves a numerical method iterating to optimize for $\tilde{\beta}_0$ based on a range of tested β_1 values. Using `confint` is much more straightforward.

```r
# calculate manual example using Wald CI for c=-10 (10 yard shorter field goal)
c <- -10
beta1 <- fit_dist$coefficients[2]
var_beta1 <- vcov(fit_dist)[2,2]
(OR_ci <- rev(exp(c*beta1 + c(-1,1) * c * qnorm(0.975) * sqrt(var_beta1))))
```

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

```r
# calculate with confint.default() which uses Wald CI; result is identical
beta1_ci <- confint.default(fit_dist, parm="distance", level=0.95)
(OR_ci2 <- rev(exp(c*beta1_ci)))
```

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

```r
# calculate with confint() which uses LRT
# in this particular case, results are similar to Wald CI (due to large n)
beta1_ci_lrt <- confint(fit_dist, parm="distance", level=0.95)
```

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

```
(OR_ci_lrt <- rev(exp(c*beta1_ci_lrt)))
```

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

Probability of Success and its Confidence Interval

Wald Confidence Interval

Similar to how we calculate CI for OR, we first find CI for the linear predictor, and then plug it back in to the equation for $\hat{\pi}$.

For example, Wald CI for $\beta_0 + \beta_1 x$ is:

$$(1-\alpha)100\% \text{ Wald CI} = \hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-\alpha/2}\sqrt{V\hat{a}r(\hat{\beta}_0 + \hat{\beta}_1 x)}$$

Get the variance terms from `vcov`:

$$V\hat{a}r(\hat{\beta}_0 + \hat{\beta}_1 x) = V\hat{a}r(\hat{\beta}_0) + x^2 V\hat{a}r(\hat{\beta}_1) + 2xC\hat{o}v(\hat{\beta}_0, \hat{\beta}_1)$$

Now just plug this in to the equation for $\hat{\pi}$:

$$\hat{\pi} = \frac{exp(\hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-\alpha/2}\sqrt{V\hat{a}r(\hat{\beta}_0 + \hat{\beta}_1 x)})}{1 + exp(\hat{\beta}_0 + \hat{\beta}_1 x \pm Z_{1-\alpha/2}\sqrt{V\hat{a}r(\hat{\beta}_0 + \hat{\beta}_1 x)})}$$

In reality, we shouldn't have to calculate this manually We use `predict(...)` to calculate Wald CIs - we can obtain the linear predictor and the standard error ($= \sqrt{Var(\text{linear predictor})}$).

```
# predict probability of success from 20 yards
x <- 20
# Approach 1: manual calculation (Wald)
(lp <- fit_dist$coefficients[1] + fit_dist$coefficients[2]*x)
```

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

```
(pihat <- exp(lp)/(1+exp(lp)))
```

```
## (Intercept)
##   0.9710145
```

```
# Approach 2: use predict(...) function to get the linear predictor
(lp <- predict(fit_dist, newdata=data.frame(distance=x), type="link"))
```

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

```
(pihat <- exp(lp)/(1+exp(lp)))
```

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

```
# Approach 3: use predict(...) to directly get pihat
(pihat <- predict(fit_dist, newdata=data.frame(distance=x), type="response"))
```

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

```
# Aside: You can get the fitted values for the initial dataset by omitting the newdata param
(head(predict(fit_dist, type="response")))
```

```
##         1         2         3         4         5         6
## 0.9675956 0.9675956 0.9710145 0.9303017 0.9710145 0.9496174
```

```
# Equivalent to:
(head(fit_dist$fitted.values))
```

```
##         1         2         3         4         5         6
## 0.9675956 0.9675956 0.9710145 0.9303017 0.9710145 0.9496174
```

```
# calculate PiHat and Wald CI using linear predictor and std err returned by predict(...)
(lp <- predict(fit_dist, newdata=data.frame(distance=x), type="link", se=TRUE))
```

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

```
(lp_se <- lp$se)
```

```
## [1] 0.1732707
```

```
(pihat <- predict(fit_dist, newdata=data.frame(distance=x), type="response"))
```

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

```
# note that when using se=TRUE, we need to refer to lp$fit instead of just lp for the linear p
(ci_lp <- lp$fit + c(-1,1) * qnorm(0.975) * lp_se)
```

```
## [1] 3.171942 3.851151
```

```
# have to transform CI on the linear predictor to CI on Pi
(ci_pi <- exp(ci_lp) / (1+exp(ci_lp)))
```

```
## [1] 0.9597647 0.9791871
```

Profile Likelihood Interval

We use `mcprofile(...)` to calculate profile likelihood intervals. We need to pass in a matrix of all of our x values to be evaluated, rather than a dataframe. **This matrix needs to include a column of 1's** to be multiplied by β_0 (so that matrix dimensions match).

```
# Now calculate Profile Likelihood Interval for PiHat at x=20 yards
library(mcprofile)
```

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

```
K <- matrix(data=c(1,x), nrow=1, ncol=2)
# Calculate -2log(Lambda)
(linear_combo <- mcprofile(fit_dist, CM=K))
```

```
##
##    Multiple Contrast Profiles
##
##    Estimate Std.err
## C1     3.51   0.173
```

```
# Calculate CI for beta_0 + beta_1 * x
(ci_logit_pli <- confint(linear_combo, level=0.95))
```

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

```
# Calculate CI for Pi
(ci_pi_pli <- exp(ci_logit_pli$confint) / (1+exp(ci_logit_pli$confint)))
```

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

Probability Plots

We need to make sure our data is in binomial form. See above for example of converting Bernoulli trials to Binomial format. Also we need to wrap our confidence interval prediction logic in a formula that can accept a single x from the `curve(...)` function.

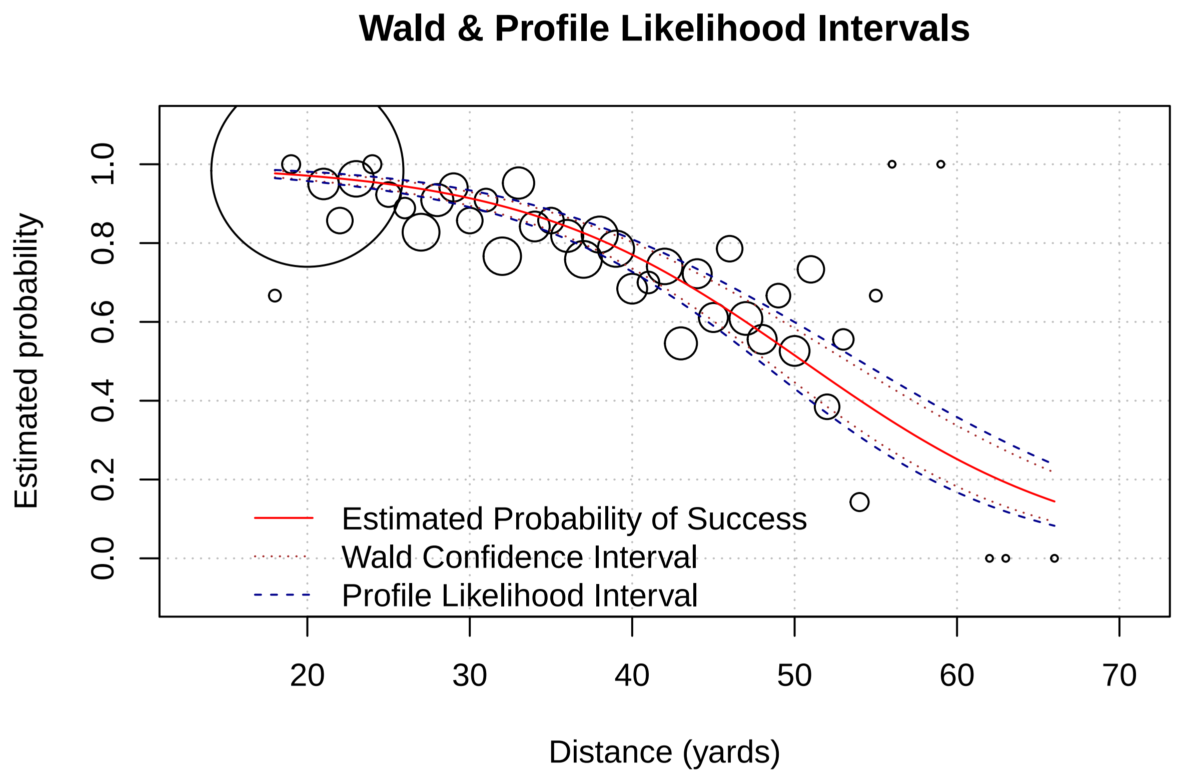
```
# Start with data in binomial format
head(placekick_binom)
```

```
##   distance   w   n proportion
## 1       18   2   3  0.6666667
## 2       19   7   7  1.0000000
## 3       20 776 789  0.9835234
## 4       21  19  20  0.9500000
## 5       22  12  14  0.8571429
## 6       23  26  27  0.9629630
```

```
# Need to create functions that takes x as input and returns CI.
wald_ci <- function(mod_fit, distance, alpha, which="both") {
  lp <- predict(mod_fit, newdata=data.frame(distance=distance), type="link", se=TRUE)
  ci_lp_lower <- lp$fit - qnorm(0.975) * lp$se
  ci_lp_upper <- lp$fit + qnorm(0.975) * lp$se
  ci_pi_lower <- exp(ci_lp_lower) / (1+exp(ci_lp_lower))
  ci_pi_upper <- exp(ci_lp_upper) / (1+exp(ci_lp_upper))
    list(lower=ci_pi_lower, upper=ci_pi_upper)
}

profile_ci <- function(mod_fit, distance, alpha) {
  matrix_vals <- c(rep(1, length(distance)),distance)
  K <- matrix(data=matrix_vals, nrow=length(distance), ncol=2)
  linear_combo <- mcprofile(mod_fit, CM=K)
  ci_logit <- confint(linear_combo, level=1-alpha)
  exp(ci_logit$confint) / (1+exp(ci_logit$confint))
}
```

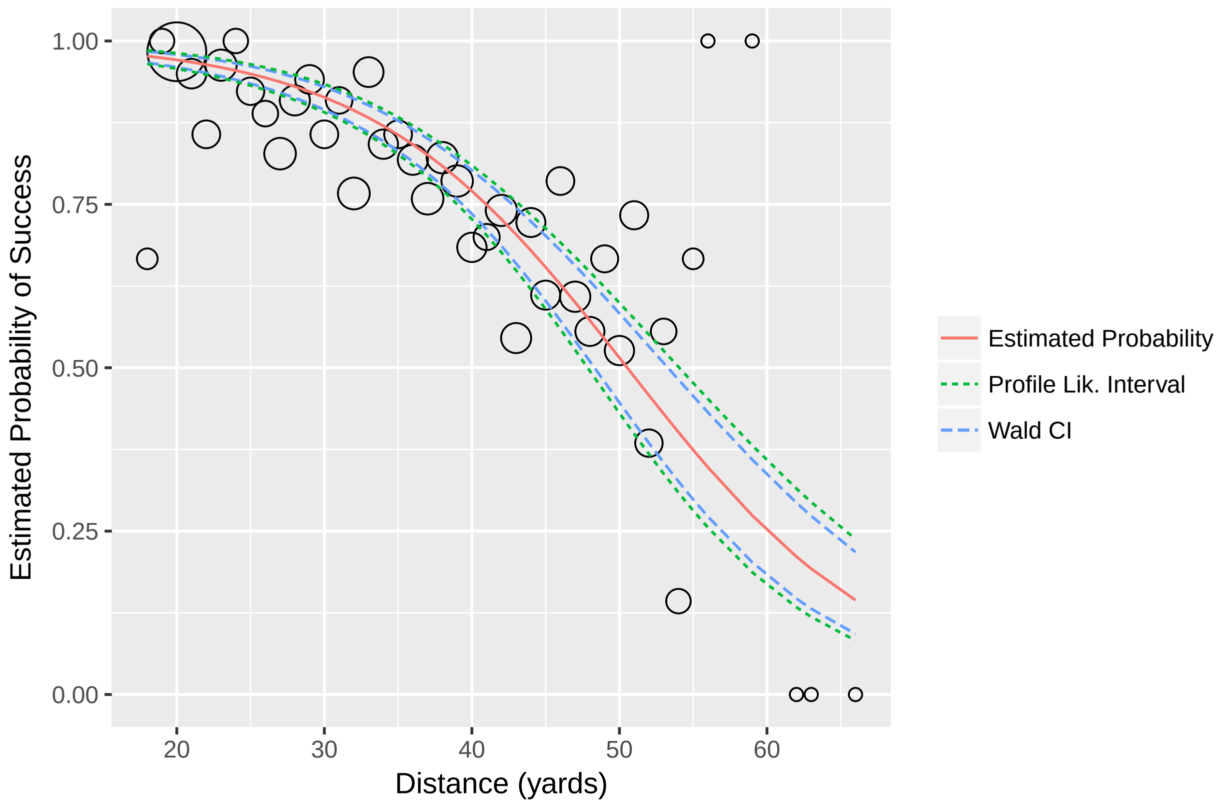
Now we can plot the data, the main probability curve, and the CI curves. The `symbols(...)` function has a similar signature to `plot(...)` but lets us scale size of the points. Note the choice of scaling by the square root.

```
# Plot points, PiHat curve

# plot(placekick_binom$distance, placekick_binom$proportion, xlab="Distance (yards)", ylab="Pi
#       main="Wald & Profile Likelihood Intervals", panel.first=grid(col="gray", lty="dotted"))
symbols(x=placekick_binom$distance, y=placekick_binom$proportion, circles = sqrt(placekick_bino
        inches = 0.5, xlab = "Distance (yards)", ylab = "Estimated probability",
        main="Wald & Profile Likelihood Intervals", panel.first=grid(col="gray", lty="dotted")
curve(expr=predict(fit_dist, newdata=data.frame(distance=x), type="response"), add=TRUE, col="

# plot 95% Wald CI limits.
curve(expr=wald_ci(fit_dist, x, 0.05)$lower, add=TRUE, col="brown", lty="dotted", xlim=c(18,66
curve(expr=wald_ci(fit_dist, x, 0.05)$upper, add=TRUE, col="brown", lty="dotted", xlim=c(18,66

# plot 95% confidence profile likelihood limits.  (This runs slowly)
curve(expr=profile_ci(fit_dist, x, 0.05)$lower, add=TRUE, col="darkblue", lty="dashed", xlim=c
curve(expr=profile_ci(fit_dist, x, 0.05)$upper, add=TRUE, col="darkblue", lty="dashed", xlim=c
legend(x=15, y=0.2, legend = c("Estimated Probability of Success",
                               "Wald Confidence Interval", "Profile Likelihood Interval"),
       lty = c("solid", "dotted", "dashed"), col = c("red", "brown", "darkblue"), bty = "n")
```



```
# Alternately, use ggplot2 to generate the plot
ggplot(placekick_binom, aes(distance, proportion)) +
  geom_point(shape=1, aes(size = n^0.5)) +
  geom_line(aes(distance, predict(fit_dist, newdata=data.frame(distance=distance), type="respon
              linetype="Estimated Probability", color="Estimated Probability")) +
  geom_line(aes(distance, wald_ci(fit_dist, distance, 0.05)$lower, linetype="Wald CI", color="W
  geom_line(aes(distance, wald_ci(fit_dist, distance, 0.05)$upper, linetype="Wald CI", color="W
  geom_line(aes(distance, profile_ci(fit_dist, distance, 0.05)$lower, linetype="Profile Lik. In
              color="Profile Lik. Interval")) +
  geom_line(aes(distance, profile_ci(fit_dist, distance, 0.05)$upper, linetype="Profile Lik. In
              color="Profile Lik. Interval")) +
  labs(title="95% Wald & Profile Likelihood Intervals", x="Distance (yards)", y="Estimated Prob
       color="", linetype="", shape="") +
  scale_size(range = c(2, 10)) +
  guides(size=FALSE) +
  xlim(18,66) +
  theme(plot.title = element_text(lineheight=1, face="bold"))
```

95% Wald & Profile Likelihood Intervals


Estimated Probability of Success
Distance (yards)
Estimated Probability
Profile Lik. Interval
Wald CI