PS733 Homework 1 Q&A

Haohan Chen

February 19, 2018

How to make make prediction with a linear model?

I demonstrate this by a mini analysis on a dataset I randomly generate.

The table below shows the data.

	Y	constant	x1
1	-5.27	1.00	-5.76
2	13.99	1.00	7.22
3	39.62	1.00	24.05
4	-9.99	1.00	-8.56
5	8.67	1.00	4.04
6	12.59	1.00	6.59
7	22.38	1.00	13.50
8	5.73	1.00	2.12
9	45.94	1.00	28.81
10	7.88	1.00	3.33
11	17.83	1.00	10.01
12	27.19	1.00	16.78
13	3.01	1.00	0.29
14	-7.26	1.00	-7.48
15	42.89	1.00	26.39
16	-30.10	1.00	-22.73
17	26.16	1.00	15.54
18	11.10	1.00	5.43
19	28.64	1.00	17.15
20	17.68	1.00	10.19
21	47.72	1.00	30.09
22	-10.07	1.00	-9.40
23	38.83	1.00	24.08
24	46.32	1.00	28.46
25	10.07	1.00	5.06
26	-34.61	1.00	-24.42
27	18.93	1.00	10.73
28	0.23	1.00	-2.16
29	25.33	1.00	14.51
30	16.55	1.00	8.48

Now I fit two linear model $y_i = \beta_0 + \beta_1 x_{i1} + \epsilon_i \quad \epsilon_i \sim N(0, \sigma^2)$, one with an Ordinary Least Square Estimator, another with an Maximum Likelihood Estimator.

OLS Inference and Prediction

```
m_ols <- lm(Y ~ x1, data = d_linear)
summary(m_ols)
```

```
##
## Call:
## lm(formula = Y ~ x1, data = d_linear)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1591 -0.3667 -0.1291  0.4372  0.9474
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.029113   0.120769   25.08   <2e-16 ***
## x1          1.493978   0.007612  196.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5773 on 28 degrees of freedom
## Multiple R-squared:  0.9993, Adjusted R-squared:  0.9992
## F-statistic: 3.852e+04 on 1 and 28 DF,  p-value: < 2.2e-16
```

Now, how do we predict the level of Y at the 10th percentile of the x_1? As I demonstrated in the lab, we need to:

1. Get the estimated coefficients using the 'coef()' function
2. Get the Variance-Covariance Matrix using the 'vcov()' function. If you wonder what this is, it describes the variance of our estimated beta's (how certain we are about our estimated coefficients), and the covariance among beta's.
3. Simulate a sample of the estimated coefficients (if your goal is hypothesis testing, wich you probably have done a lot, you stop here).
4. Use (1) the simulated distribution of the coefficients AND; (2) the estimated Sigma retrieved with the 'Sigma()' function AND (2) the 10th quantile x_1 to simulate a sample of the predicted Y

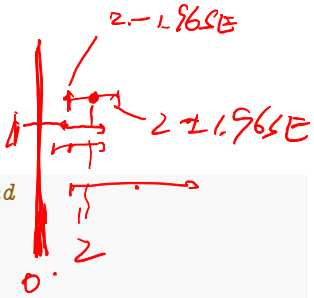
Here's how to code this:

```
# STEP 1: Get the estimated coefficients
beta_est <- coef(m_ols)
# STEP 2: Get the variance-covariance matrix
beta_vcov <- vcov(m_ols)
# STEP 3: Simulate a sample of the estimated coefficient
beta_sim <- MASS::mvrnorm(10000, beta_est, beta_vcov)
# Aside: We can get the confidence intervals of our coefficients from here
apply(beta_sim, 2, function(x) quantile(x, c(0.025, 0.5, 0.975)))
```

```
##       (Intercept)       x1
## 2.5%     2.789670 1.479003
## 50%      3.026888 1.494014
## 97.5%    3.264330 1.509109
```

```
# The result from simulation (above), should be very closed to results we get
# from the confint() function.
confint(m_ols)
```

```
##                 2.5 %   97.5 %
## (Intercept) 2.781729 3.276498
## x1          1.478386 1.509570
```



```r
# Now we have the simulated distribution of the coefficients, we need
# (1) The estimated sigma (variance of the regression)
sigma <- sigma(m_ols)
print(sigma)
```

```
## [1] 0.5773357
```

```r
# (2) the i'th quantile x_i. That is, we need our x_new
  # Note: Don't forget the intercept!
x_new_10q <- c(`(Intercept)` = 1, x1 = as.numeric(quantile(d_linear$x1, 0.10)))
print(x_new_10q)
```

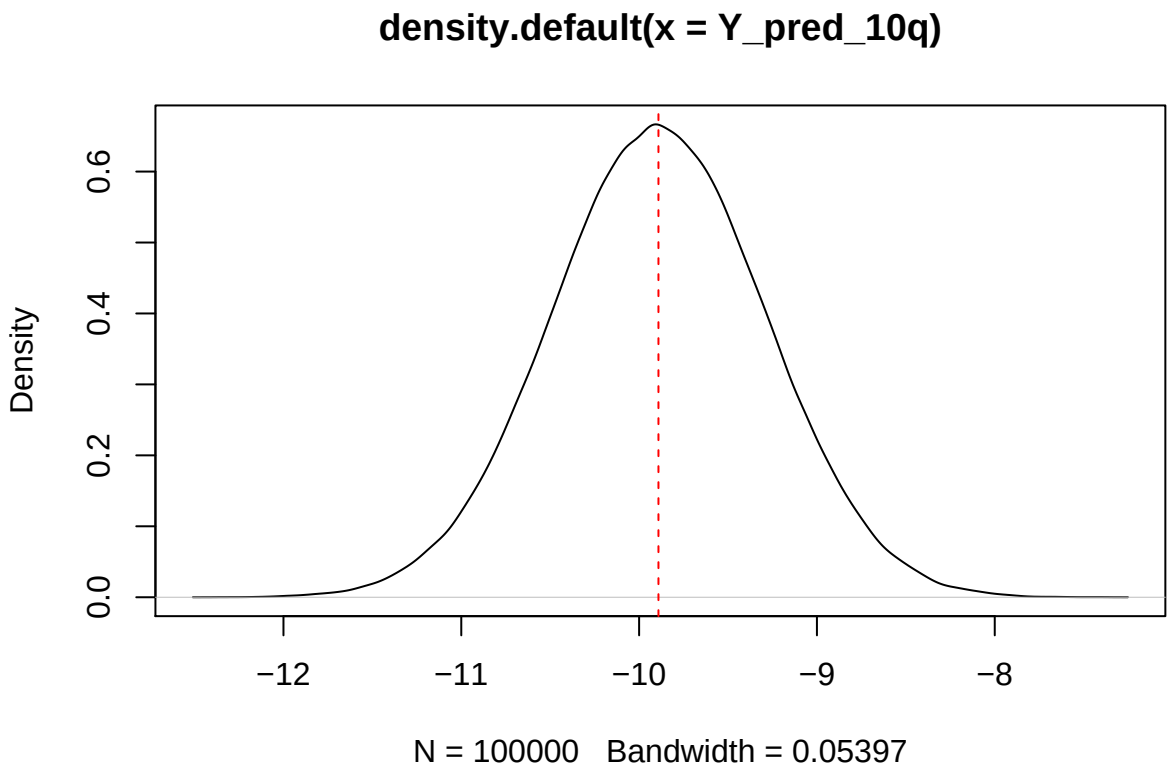
```
## (Intercept)          x1
##    1.000000   -8.647968
```

```r
# STEP 4: Simulate the predicted Y.
Y_pred_10q <- rnorm(100000, x_new_10q %*% t(beta_sim), sigma)
  # An alternative way to code this simulation (slower, but more intuitive)
  # Y_pred_10q <- apply(beta_sim, 1, function(x) rnorm(1000, sum(x_new_10q * x), sigma))

# Analysis of the outcome: The 95% Confidence Interval
quantile(Y_pred_10q, c(0.025, 0.5, 0.975))
```

```
##       2.5%        50%      97.5%
## -11.071530  -9.891519  -8.719848
```

```r
# Plot the distribution and the mean
plot(density(Y_pred_10q))
abline(v = mean(Y_pred_10q), lty = 2, col = "red")
```




```
# Q: "Can I use the `predict` function?" Yes. But make sure you use get the predictive interval
predict(m_ols, newdata = data.frame(rbind(x_new_10q)), interval = "prediction")
```

```
##                 fit       lwr       upr
## x_new_10q -9.890757 -11.11979 -8.661721
```

```
# But I strongly suggest you understand the above simulation method, because --
# when you work on more complex dataset, the predict function is less flexible than
# the type of task you need. This will be a powerful tool.
```

Linear MLE inference and prediction

Below I fit a linear model with Maximum Likelihood Estimator. Note the choice of family for the `GLM` function is by default `gaussian`, that is, a linear model. Remember to specify the family when you are not fitting a linear model.

```
m_mle <- glm(Y ~ x1, data = d_linear, family = "gaussian")
summary(m_mle)
```

```
##
## Call:
## glm(formula = Y ~ x1, family = "gaussian", data = d_linear)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1591  -0.3667  -0.1291   0.4372   0.9474
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.029113   0.120769   25.08   <2e-16 ***
## x1          1.493978   0.007612  196.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.3333166)
##
##     Null deviance: 12849.5868  on 29  degrees of freedom
## Residual deviance:     9.3329  on 28  degrees of freedom
## AIC: 56.107
##
## Number of Fisher Scoring iterations: 2
```

The way I do prediction is almost identical to that of OLS, because the two packages provide the same set of functions. **Note:** The `predict` function for GLM models does not offer a convenient option for you to get the predictive interval. So just use the simulation method.

```
# STEP 1: Get the estimated coefficients
beta_est <- coef(m_mle)
# STEP 2: Get the variance-covariance matrix
beta_vcov <- vcov(m_mle)
# STEP 3: Simulate a sample of the estimated coefficient
beta_sim <- MASS::mvrnorm(10000, beta_est, beta_vcov)
# Aside: We can get the confidence intervals of our coefficients from here
apply(beta_sim, 2, function(x) quantile(x, c(0.025, 0.5, 0.975)))
```

```
##       (Intercept)       x1
## 2.5%     2.794329 1.478626
## 50%      3.028997 1.494025
## 97.5%    3.258912 1.508690
```

```
# The result from simulation (above), should be very closed to results we get
# from the confint() function.
confint(m_mle)
```

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

```
##                2.5 %   97.5 %
```

```
## (Intercept) 2.792410 3.265816
## x1          1.479059 1.508896
```

```r
# Now we have the simulated distribution of the coefficients, we need
# (1) The estimated sigma (variance of the regression)
sigma <- sigma(m_mle)
print(sigma)
```

```
## [1] 0.5773357
```

```r
# (2) the i'th quantile x_i. That is, we need our x_new
  # Note: Don't forget the intercept!
x_new_10q <- c(`(Intercept)` = 1, x1 = as.numeric(quantile(d_linear$x1, 0.10)))
print(x_new_10q)
```

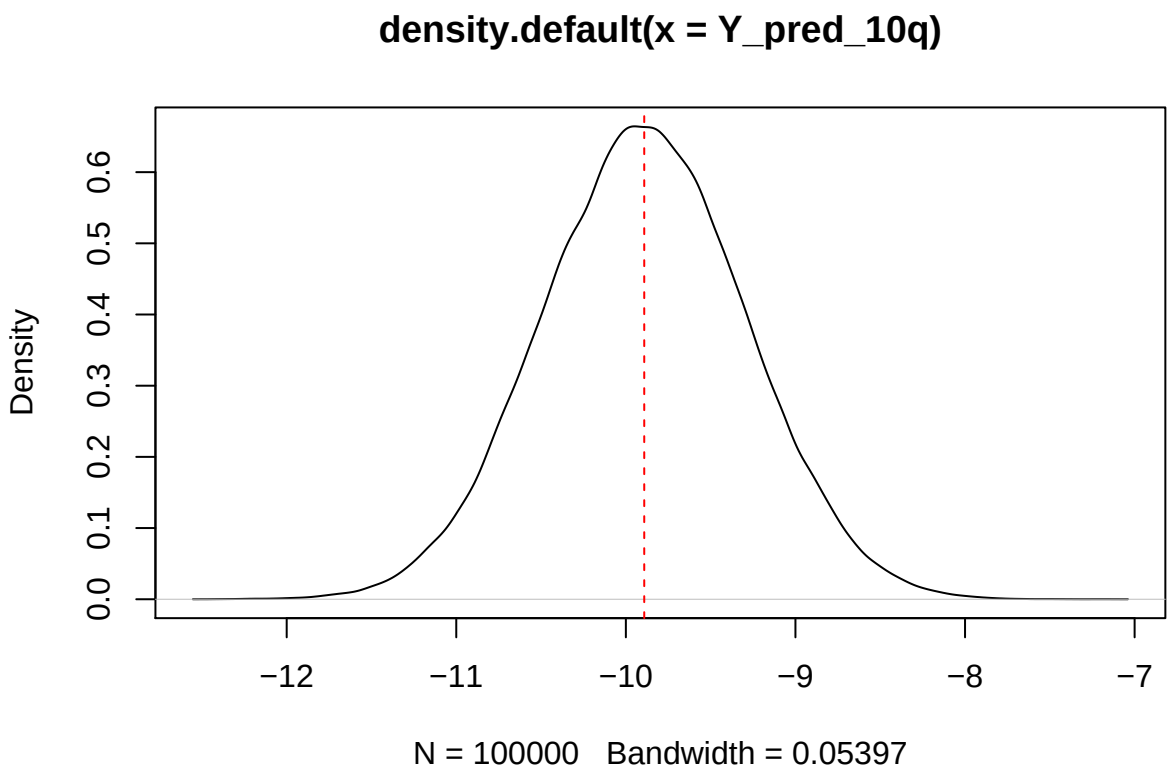
```
## (Intercept)          x1
##    1.000000   -8.647968
```

```r
# STEP 4: Simulate the predicted Y.
Y_pred_10q <- rnorm(100000, x_new_10q %*% t(beta_sim), sigma)
  # An alternative way to code this simulation (slower, but more intuitive)
  # Y_pred_10q <- apply(beta_sim, 1, function(x) rnorm(1000, sum(x_new_10q * x), sigma))

# Analysis of the outcome: The 95% Confidence Interval
quantile(Y_pred_10q, c(0.025, 0.5, 0.975))
```

```
##      2.5%       50%     97.5%
## -11.06732  -9.89174  -8.71739
```

```r
# Plot the distribution and the mean
plot(density(Y_pred_10q))
abline(v = mean(Y_pred_10q), lty = 2, col = "red")
```




```
# Q: "Can I use the `predict` function?" It's not going to be much helful for GLM...
predict(m_mle, newdata = data.frame(rbind(x_new_10q)))
```

```
## x_new_10q
## -9.890757
```

```
# This is because the GLM family models does not offer a convenient way to get the predictive interval
# So go with simulation!
```

Predictive Probability at certain level of x

Again, I create a dataset with binary outcome Y to demonstrate this.

Below I show the first 30 observations of the generated dataset. Y is a binary outcome.

```
xtable(d_logit)
```

	Y	constant	x1	x2
1	0	1.00	-5.76	2.00
2	1	1.00	7.22	2.00
3	1	1.00	24.05	3.00
4	1	1.00	-8.56	3.00
5	1	1.00	4.04	2.00
6	0	1.00	6.59	3.00
7	0	1.00	13.50	3.00
8	1	1.00	2.12	3.00
9	1	1.00	28.81	4.00
10	1	1.00	3.33	4.00
11	1	1.00	10.01	3.00
12	1	1.00	16.78	3.00
13	1	1.00	0.29	4.00
14	1	1.00	-7.48	4.00
15	1	1.00	26.39	4.00
16	0	1.00	-22.73	2.00
17	1	1.00	15.54	3.00
18	0	1.00	5.43	2.00
19	1	1.00	17.15	3.00
20	1	1.00	10.19	3.00
21	1	1.00	30.09	3.00
22	0	1.00	-9.40	2.00
23	1	1.00	24.08	4.00
24	1	1.00	28.46	3.00
25	1	1.00	5.06	4.00
26	0	1.00	-24.42	2.00
27	1	1.00	10.73	1.00
28	1	1.00	-2.16	3.00
29	1	1.00	14.51	2.00
30	1	1.00	8.48	3.00

Predicted probability

First we fit a logit model:

```
m_logit <- glm(Y ~ x1 + x2, data = d_logit, family = binomial(link = "logit"))
summary(m_logit)
```

```
##
## Call:
## glm(formula = Y ~ x1 + x2, family = binomial(link = "logit"),
##     data = d_logit)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
```

```
## -2.31487   0.09854   0.32789   0.57867   1.15328
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.23137    1.87060  -1.193   0.2329
## x1           0.10841    0.05589   1.940   0.0524 .
## x2           1.12551    0.69975   1.608   0.1077
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 32.596  on 29  degrees of freedom
## Residual deviance: 20.941  on 27  degrees of freedom
## AIC: 26.941
##
## Number of Fisher Scoring iterations: 6
```

Want Predictive Probability for $x_2 = 2$ and average x_1.

```
# STEP 1: Get the estimated coefficients
beta_est <- coef(m_logit)
# STEP 2: Get the variance-covariance matrix
beta_vcov <- vcov(m_logit)
# STEP 3: Simulate a sample of the estimated coefficient
beta_sim <- MASS::mvrnorm(10000, beta_est, beta_vcov)
# Aside: We can get the confidence intervals of our coefficients from here
apply(beta_sim, 2, function(x) quantile(x, c(0.025, 0.5, 0.975)))
```

```
##       (Intercept)           x1         x2
## 2.5%    -6.000300 -0.002770503 -0.2805416
## 50%     -2.261127  0.107445646  1.1321446
## 97.5%    1.488850  0.217770113  2.5482581
```

```
# The result from simulation (above), should be very closed to results we get
# from the confint() function.
confint(m_logit)
```

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

```
##                   2.5 %    97.5 %
## (Intercept) -6.30357317 1.3938608
## x1           0.01739943 0.2474546
## x2          -0.16270406 2.7137109
```

```
# Mind your interpretation of the coefficients. Talk about it later.

# Now we have the simulated distribution of the coefficients, we need
# (1) The estimated sigma (variance of the regression)
sigma <- sigma(m_logit)
print(sigma)
```

```
## [1] 0.8806801
```

```
# (2) define new x: x2 = 2; x1 = mean(x1)
  # Note: Don't forget the intercept!
```

```r
x_new <- c(`(Intercept)` = 1, x1 = mean(d_logit$x1), x2 = 2)
print(x_new)
```

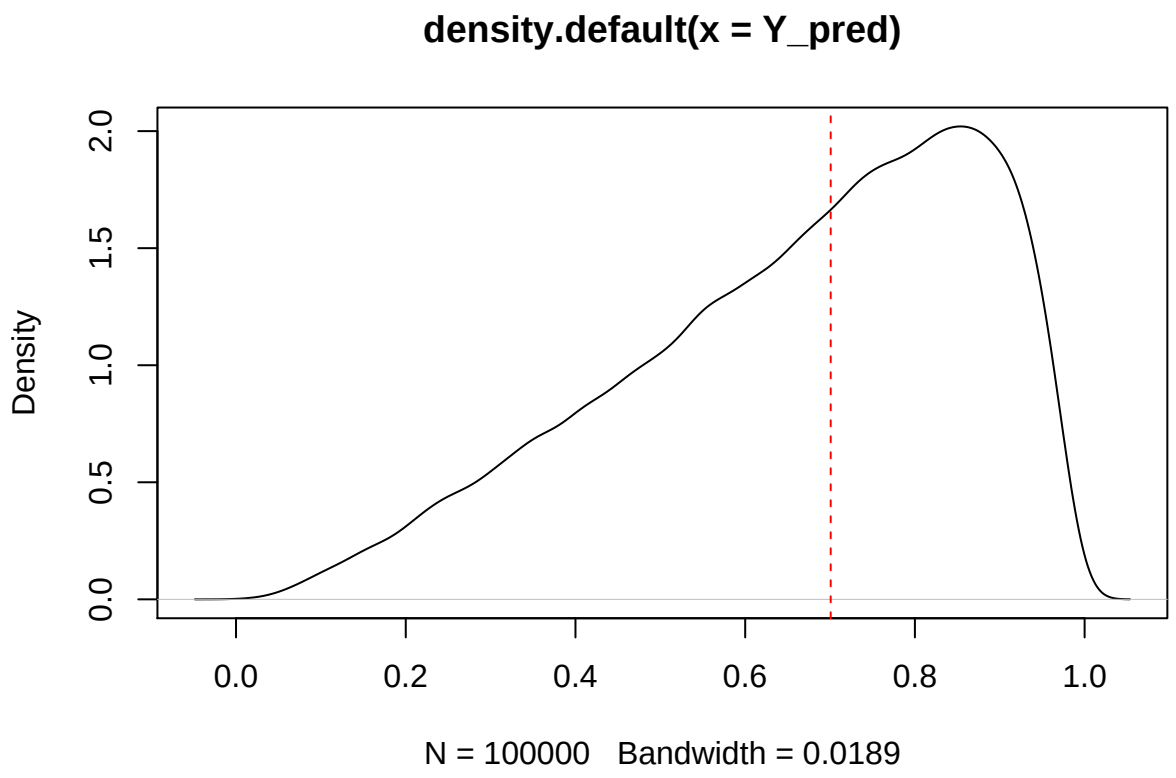
```
## (Intercept)          x1          x2
##    1.000000    7.744062    2.000000
```

```r
# STEP 4: Simulate the predicted Y.
Y_pred <- 1 / (1 + exp(- rnorm(100000, x_new %*% t(beta_sim), sigma)))
  # An alternative way to code this simulation (slower, but more intuitive)
  # Y_pred_10q <- apply(beta_sim, 1, function(x) rnorm(1000, sum(x_new_10q * x), sigma))

# Analysis of the outcome: The 95% Confidence Interval
quantile(Y_pred, c(0.025, 0.5, 0.975))
```

```
##      2.5%       50%     97.5%
## 0.2023469 0.7007668 0.9568255
```

```r
# Plot the distribution and the mean
plot(density(Y_pred))
abline(v = median(Y_pred), lty = 2, col = "red")
```




```r
# Q: "Can I use the `predict` function?" Won't be helpful either. See linear MLE explanation
predict(m_logit, newdata = data.frame(rbind(x_new)))
```

```
##     x_new
## 0.8592009
```

```r
# Note: the GLM family models does not offer a convenient way to get the predictive interval
# So go with simulation!
```

If you want the predictive probability for all $x_2 \in \{1, 2, 3, 4, 5\}$, simply repeat the above simulation five times, changing the value of x_2.

Want: How Predictive Probability Change along x_1, when $x_2 = 2$

This is a little bit more challenging than above, because x_1 is continuous. So you need to get a sequence of x_1 to show the change of Y along x_1.

```r
# STEP 1: Get the estimated coefficients
beta_est <- coef(m_logit)
# STEP 2: Get the variance-covariance matrix
beta_vcov <- vcov(m_logit)
# STEP 3: Simulate a sample of the estimated coefficient
beta_sim <- MASS::mvrnorm(10000, beta_est, beta_vcov)
# Aside: We can get the confidence intervals of our coefficients from here
apply(beta_sim, 2, function(x) quantile(x, c(0.025, 0.5, 0.975)))
```

```
##       (Intercept)           x1         x2
## 2.5%    -5.970067 -0.002923229 -0.2620267
## 50%     -2.208560  0.109318685  1.1180238
## 97.5%    1.442963  0.218237200  2.5201717
```

```r
# The result from simulation (above), should be very closed to results we get
# from the confint() function.
confint(m_logit)
```

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

```
##                    2.5 %    97.5 %
## (Intercept) -6.30357317 1.3938608
## x1           0.01739943 0.2474546
## x2          -0.16270406 2.7137109
```

```r
# Mind your interpretation of the coefficients. Talk about it later.

# Now we have the simulated distribution of the coefficients, we need
# (1) The estimated sigma (variance of the regression)
sigma <- sigma(m_logit)
print(sigma)
```

```
## [1] 0.8806801
```

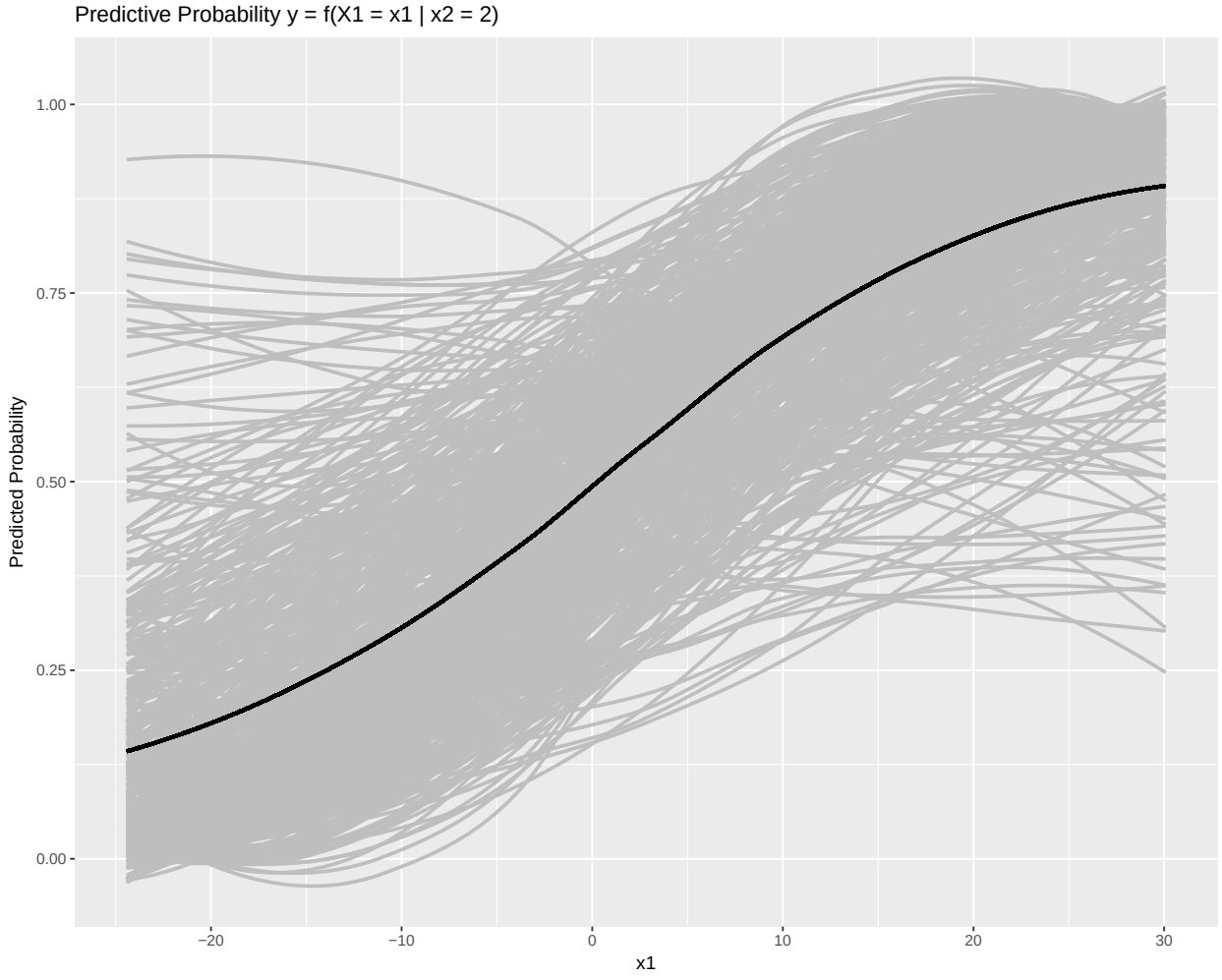
```r
# (2) define new x: x2 = 2; x1 = a sequence of x1 from min to max
  # Note: Don't forget the intercept!
  # Note: I take a sequence of x1 with interval 1. adjust it to fit your data
x1_seq <- seq(min(d_logit$x1), max(d_logit$x1), 0.5)
x_new <- cbind(`(Intercept)` = 1, x1 = x1_seq, x2 = 2)

# STEP 4: Simulate the predicted Y. for the whole sequence of x1

# !! Below are some data cleaning to make create a data format working with ggplot
# !! It's probably not the most efficient way though. Let me know if you have better idea.
Y_pred <- as.data.frame(
  t(apply(x_new, 1, function(x) 1 / (1 + exp(- rnorm(500, x %*% t(beta_sim), sigma)))))
```

```
  )
Y_pred$x1 <- x1_seq
# Reshape the dataset for plotting
Y_pred2 <- reshape2::melt(Y_pred, id.vars = "x1")
Y_pred_mean <- data.frame(Y_pred_m = apply(Y_pred[, 1:(ncol(Y_pred)-1)], 1, mean),
                          x1 = x1_seq)
Y_pred3 <- merge(Y_pred2, Y_pred_mean, by = "x1")

library(ggplot2)
ggplot(Y_pred3) +
  geom_smooth(aes(x = x1, y = value, group = variable),
              method = "loess", se = F, color = "gray") +
  geom_smooth(aes(x = x1, y = Y_pred_m, group = variable),
              method = "loess", se = F, color = "black") +
  guides(color = F) +
  xlab("x1") + ylab("Predicted Probability") +
  ggtitle("Predictive Probability y = f(X1 = x1 | x2 = 2)")
```




```
# Of course you can plot the predictive interval as ribbon etc.
```