PH 295: Lab 1 Assignment Answer

2016-09-13

I. Model misspecification

The first exercise illustrates the effect of misspecification of a parametric model on inference about the effect of a treatment A on an outcome Y. Consider the setting that the observed data consist of n i.i.d. copies of $O = (W, A, Y)$, where

$$W \sim Uniform\,(0, 1)$$

$$A \sim Bernoulli(expit[1/12 + (W - 1/2)^2])$$

$$Y \sim Bernoulli(expit[(W - 1/2)^2])) \ ,$$

where $expit(x) = exp(x)/[1 + exp(x)]$.

Suppose we are interested in the parameter

$$\psi_0(W) = \frac{P(Y = 1|A = 1, W)/[1 - P(Y = 1|A = 1, W)]}{P(Y = 1|A = 0, W)/[1 - P(Y = 1|A = 0, W)]} \ ,$$

which is the ratio of the odds of the outcome Y comparing $A = 1$ to $A = 0$ conditional on W. Suppose we know that this parameter is actually constant in W so we can write $\psi_0 = \psi_0(W)$.

Your "Statistics for Dummies" book recommends you use a main terms logistic regression model, which assumes that

$$P(Y = 1|A, W) = expit[\beta_0 + \beta_1 A + \beta_2 W]$$

for some unknown parameter $\beta = (\beta_0, \beta_1, \beta_2)$.

1. Show that if the assumed logistic regression model is true then $\psi_0 = exp(\beta_1)$.

Under the assume logistic regression model, we can define

$$p_{1,W} = P(Y = 1|A = 1, W) = expit(\beta_0 + \beta_1 + \beta_2 W)$$
$$p_{0,W} = P(Y = 1|A = 0, W) = expit(\beta_0 + \beta_2 W)$$

Also note that

$$1 - p_{1,W} = \frac{1}{1 + exp(\beta_0 + \beta_1 + \beta_2 W)}$$
$$1 - p_{0,W} = \frac{1}{1 + exp(\beta_0 + \beta_2 W)} \ ,$$

and

$$p_{1,W}/(1-p_{1,W}) = exp(\beta_0 + \beta_1 + \beta_2 W)$$
$$p_{0,W}/(1-p_{0,W}) = exp(\beta_0 + \beta_2 W)$$

Putting it all together, we have

$$\begin{aligned}\psi_0 &= p_{1,W}/(1-p_{1,W})/\{p_{0,W}/(1-p_{0,W})\} \\ &= exp(\beta_0 + \beta_1 + \beta_2 W)/exp(\beta_0 + \beta_2 W) \\ &= exp(\beta_1)\end{aligned}$$

2. The logistic regression model is in fact incorrect, so what is the true value of ψ_0?

Using the above notation, under the true model $p_{1,W} = p_{0,W}$ so we can see that the true value of ψ_0 is 1.

3. What is the value of $\beta_{1,0}$, the limit of the maximum likelihood estimator $\beta_{1,n}$ of β_1? (numerical approximation is ok)

We can use a large simulated data set to find the value of $\beta_{1,0}$.

```
library(rje)
```

```
##
## Attaching package: 'rje'
```

```
## The following object is masked from 'package:base':
##
##     arrayInd
```

```
# simulate data
N <- 1e6
W <- runif(N, 0, 1)
A <- rbinom(N, 1, plogis(1/12 + (W-1/2)^2))
Y <- rbinom(N, 1, plogis((W-1/2)^2))

# fit glm
fm <- glm(Y ~ A + W, family = binomial)

# approximate value of \beta_{1,0}
fm$coefficients[2]
```

```
##           A
## 0.009655599
```

```
# the nominal conditional odds ratio
exp(fm$coefficients[2])
```

```
##        A
## 1.009702
```

```
# can repeat the exercise with W \sim U(0,10) to magnify effect
N <- 1e6
W <- runif(N, 0, 10)
A <- rbinom(N, 1, plogis(1/12 + (W-1/2)^2))
Y <- rbinom(N, 1, plogis((W-1/2)^2))

# fit glm
```

```r
fm <- glm(Y ~ A + W, family = binomial)

# approximate value of \beta_{1,0}
fm$coefficients[2]
```

```
##          A
## 0.03036293
```

```r
# the nominal conditional odds ratio
exp(fm$coefficients[2])
```

```
##        A
## 1.030829
```

4. Design a simulation study to determine the type-1 error of the nominal level 0.05 Wald-style test of the null hypothesis that $\psi_0 = 1$ that rejects the null hypothesis whenever $|\beta_{1,n}/\hat{se}_{\beta,1,n}| > 1.96$ (the default test performed by `glm`). Assess the type-1 error at sample sizes $n = 100, 1000, 10000$.

```r
# function to simulate data
simu <- function(n.sim) {
    # modify W to follow Unif(0,10) instead of Unif(0,1)
    W <- runif(n.sim, 0, 10)
    # expit() from the rje package
    A <- rbinom(n.sim, 1, expit(1/12 + (W-.5)^2))
    Y <- rbinom(n.sim, 1, expit((W-.5)^2))
    # actual observed data
    df_observe <- data.frame(Y, A, W)

    # intervene when all A are 1
    df_A1 <- data.frame(Y, rep(1, n.sim), W)

    # intervene when all A are 0
    df_A0 <- data.frame(Y, rep(0, n.sim), W)

    return(list(df_observe, df_A1, df_A0))
}

# run the simulation in a loop (I'm also including larger sample sizes
# to illustrate the point)
n.grid <- c(100,1000,1e4,1e5)
n.repeat <- 1e2
reject_matrix <- matrix(NA, nrow = n.repeat, ncol = length(n.grid))

for (it in 1:length(n.grid)) {
    print(it)
    for (it2 in 1:n.repeat) {
        data_sim <- simu(n.sim = n.grid[it])[[1]]
        logistic_fit <- glm(Y ~ A + W, family = 'binomial', data = data_sim)
        pval <- coef(summary(logistic_fit))[,4]
        reject_matrix[it2,it] <- (pval['A'] <= 0.05)
    }
}
```

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

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## [1] 2
## [1] 3
## [1] 4
```

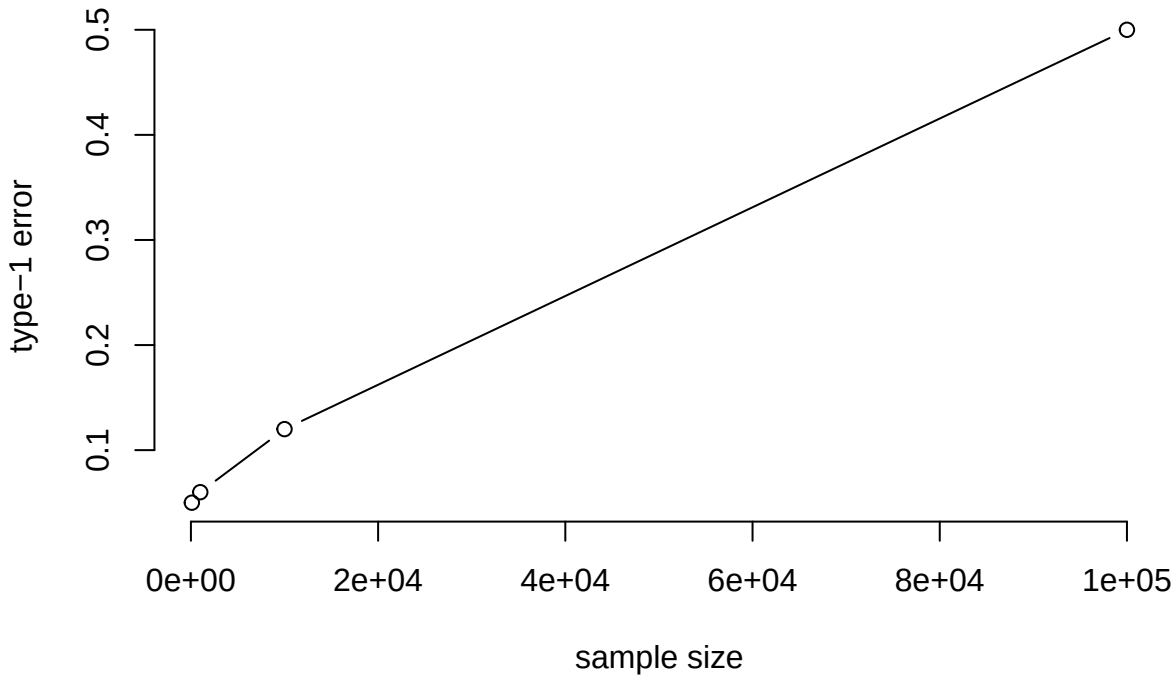
```
# type-1 error
type1Err <- colMeans(reject_matrix)
type1Err
```

```
## [1] 0.05 0.06 0.12 0.50
```

5. Comment on your findings.

The type-I error of a level 0.05 test should at least be converging to 0.05 asymptotically. However, here we find that the type-1 error increases with sample size n, and goes to 1 in the limit.

```
# plot the results of the simulation (always a good idea)
plot(type1Err ~ n.grid, bty="n",
     xlab="sample size", ylab="type-1 error",
     type="b")
```



II. Post-selection inference

In a 2001 paper that appeared in the Journal of Health Economics (that has been cited 1,398 times according to Google), Manning and Mullahy argue for the following algorithm for choosing a GLM when analyzing outcomes with a skewed distribution:

- Estimate the parameters of a main terms regression on the log-transformed values $E[log(Y)|A, W] = \beta_0 + \beta_1 A + \beta_2 W$ using maximum likelihood.
- Compute the residual from this regression

$$\varepsilon_{1,i} = log(Y_i) - (\beta_{0,n} + \beta_{1,n} A_i + \beta_{2,n} W_i)$$

- (slightly simplified from original paper) If the sample kurtosis of the residuals (see `kurtosis` from the `moments` package) is > 3, report inference for $\beta_{1,n}$.
- If the kustosis of the residuals is ≤ 3, fit a linear regression

$$log\{E(Y|A, W)\} = \alpha_0 + \alpha_1 A + \alpha_2 W \ .$$

- Compute the residual from this regression

$$\varepsilon_{2,i} = Y_i - \hat{Y}_i = Y_i - exp(\alpha_{0,n} + \alpha_{1,n} A_i + \alpha_{2,n} W_i)$$

- Using this residual, fit the following linear regression model:

$$E(\varepsilon_{2,i}^2 | \hat{Y}) = \lambda_0 + \lambda_1 log(\hat{Y})$$

  That is, regress the squared residuals from the raw-scale regression on the log-transformed predicted values from the raw-scale regression.
- (slightly simplified from original paper) If $\lambda_{1,n} < 1.5$, fit a `glm` with `family = possion(link="log")`; if $1.5 \leq \lambda_{1,n} < 2.5$ fit a `glm` with `family = Gamma(link="log")`; if $\lambda_{1,n} > 2.5$ fit a `glm` with `family = inverse.gaussian(link="log")`.
- Report the inference associated with the A-coefficient in the selected GLM.

Note: You may need to add some stability checks to make this algorithm run smoothly for all of these choices of `family`. You are welcome to use any method you like; just report what you chose. One example could be to include checking for model convergence (i.e., `glm.object$converged`) and if not report inference on log-transformed scale.

The authors show simulations illustrating this algorithm leads to increases in bias and variance over choosing a single model when the regression formula is correctly specified. They go on to report inference based on the final model for real data examples without a discussion of how model selection may affect inference. Your task is to evaluate the effect of this model selection through simulation.

Consider data generated as follows:

$$W_1 \sim Uniform(-4, 4)$$
$$W_2 \sim Bernoulli(1/2)$$
$$A \sim Bernoulli(expit[-2 + W_1/2 + W_2 + W_1 W_2/4])$$
$$log(Y) \sim Normal(5 + \psi A + W_1 + W_2 - W_1 W_2 - W_1^2/8, 1)$$

1. Set $\psi = 0$ so A has no effect on Y. Evaluate the type-1 error of the hypothesis test that rejects whenever the p-value associated with the Manning algorithm-selected model is less than 0.05. First, consider the case where the regression formula is correctly specified (i.e., your `glm` uses `formula = Y ~ A + W1*W2 + I(W1^2)`) for sample sizes $n \in \{100, 500, 1000\}$.

The manning algorithm is implemented as a function `manning_Q1.R`

```
#=====================================================
# Adaptive GLM algorithm of Manning (2001)
#=====================================================
SL.manningGLM <- function(Y, X, newX, family, obsWeights,
                                                    kCut = 3, # kurtosis cutpoint
                                                    lambdaCut = c(0.5,1.5,2.5), # skew cutpoint
                                                    startNLS=0, # starting values for NLS?
                                                    ...){
    if(family$family=="gaussian"){
        require(moments)

        # step 1: first do ols on log scale
        logY <- log(Y)
        fit.logGLM <- glm(logY ~ A + W1*W2 + I(W1^2), data=X,
                          family=family, weights=obsWeights)

        mu <- predict(fit.logGLM, type="response", newdata=X)
        # step 2: compute residual
        resid <- logY - mu

        # step 3: check kurtosis of residuals
        k <- kurtosis(resid)

        # by default use these methods
        # some of the other GLMs are unstable and if they fail, this
        # algorithm returns log OLS + smearning estimate
        # the smearing estimate is used to go back to the original from the
        # log-transformed scale -- Duan (1983)
        pred <- exp(predict(fit.logGLM, type="response",newdata=newX))*
            mean(exp(resid))
        fit <- list(object=fit.logGLM, mean(exp(resid)))
        class(fit) <- "SL.logOLS.smear"

        try({
            if(k < kCut){
                # step 4: kurtosis < 3

                # park test
            fit.initGLM <- glm(Y ~ A + W1*W2 + I(W1^2), data=X,
                               weights=obsWeights, family=gaussian(link="log"))

                # step 5: compute residual
                muPark <- predict(fit.initGLM, type="response", newdata=X)
                resid2Park <- (Y - muPark)^2

                # step 5: fit epsilon^2 on Y^hat
                fit.parkGLM <- glm(resid2Park ~ muPark,
                                   family=gaussian(link="log"))
                lambda1 <- fit.parkGLM$coef[2]

                if(lambda1 < lambdaCut[2]){
                    # 1) lamba_1 (0, 1.5)
                    # use poisson glm
```

```
                    suppressWarnings(
                        fit.poisGLM <- glm(Y ~ A + W1*W2 + I(W1^2), data=X,
                                           weights=obsWeights,
                                          family=poisson(link='log'),
                                           control=list(maxit=100))
                    )
                    pred <- predict(fit.poisGLM, newdata=newX, type="response")
                    fit <- list(object=fit.poisGLM)
                    class(fit) <- "SL.manningGLM"
                }else
                if(lambda1 < lambdaCut[3] & lambda1 >= lambdaCut[2]){
                    # 2) lamba_1 (1.5, 2.5)
                    # use gamma glm
                    fit.gammaGLM <- glm(Y ~ A + W1*W2 + I(W1^2), data=X,
                                        weights=obsWeights,
                                        family=Gamma(link='log'),
                                        control=list(maxit=100))
                    pred <- predict(fit.gammaGLM, newdata=newX, type="response")
                    fit <- list(object=fit.gammaGLM)
                    class(fit) <- "SL.manningGLM"
                }else if(lambda1 > lambdaCut[3]){
                    # 3) lamba_1 > 2.5

                    # use inverse gaussian glm -- not very stable
                    fit.invGaussianGLM <- glm(Y ~ A + W1*W2 + I(W1^2), data=X,
                                              weights=obsWeights,
                                              family=inverse.gaussian(link="log"),
                                              control=list(maxit=100))
                    pred <- predict(fit.invGaussianGLM, newdata=newX, type="response")
                    fit <- list(object=fit.invGaussianGLM)
                    class(fit) <- "SL.manningGLM"
                }
            }
        }, silent=TRUE)
    }else{
        stop("SL.manningGLM doesn't work with binomial family.")
    }
    out <- list(pred = pred, fit=fit)
    return(out)
}

# predict function
predict.SL.manningGLM <- function(object, newdata,...){
    if(!is.list(object$object)){
        pred <- predict(object=object$object, newdata=newdata, type="response")
    }else{
        pred <- predict(object=object$object, newdata=newdata, type="response")
    }
    pred
}
```

The simulation code for data generating distribution is `simu_Q1.R`

```r
simu_Q1 <- function(n.sim, Psi) {
    library(rje)
    W1 <- runif(n.sim, -4, 4)
    W2 <- rbinom(n.sim, 1, prob = .5)
    A <- rbinom(n.sim, 1, prob = expit(-2 + W1/2 + W2 +W1*W2/4))
    logY <- 5 + Psi*A + W1 + W2 - W1*W2 - W1^2/8 + rnorm(n.sim)
    Y <- exp(logY)

    return(data.frame(Y, A, W1, W2))
}
```

Check the coverage of the fitting manning model using `check_manning.R`

```r
check_manning <- function(manning_fit, robustVar = FALSE) {
    # infernce on final fit from manning

    # class(manning_fit$fit)

    if(!robustVar){
        pval <- coef(summary(manning_fit$fit$object))[,4]
    }else{
        se <- sqrt(vcovHC(manning_fit$fit$object)[2,2])
        pval <- 2*pnorm(-abs(manning_fit$fit$object$coefficients[2]/se))
    }
    return(pval['A'] <= 0.05)
}
```

For a single sample size n, write an R function to perform simulating and checking coverage. There may be warnings raised by the algorithm, where the glm's with non-canonical link functions could be highly unstable.

```r
sim_for_one_n <- function(n.sim, Psi, n.repeat) {
    reject_holder <- rep(NA, n.repeat)
    for (it in 1:n.repeat) {
        df_sim <- simu_Q1(n.sim, Psi)
        newX <- data.frame(Y = 1, A = 1, W1 = 1, W2 = 1)

        manning_fit <-
            SL.manningGLM(Y = df_sim$Y, X = df_sim[,2:4],
                                    newX = newX,
                                  family = gaussian(), kCut = 3,
                                   obsWeights = rep(1, n.sim),
                                   lambdaCut = c(0.5, 1.5, 2.5))

        reject_holder[it] <- check_manning(manning_fit, robustVar = FALSE)
    }
    reject_holder[is.na(reject_holder)] <- FALSE
    alpha <- mean(reject_holder)
    alpha
}

# simulate for a grid of n, I added more sample sizes out of curiousity
n.grid <- c(100, 500, 1000, 5000, 10000)

alpha_holder <- rep(NA, length(n.grid))
for (it2 in 1:length(n.grid)) {
```

```
    print(it2)
    alpha_holder[it2] <- sim_for_one_n(n.sim = n.grid[it2],
                                       Psi = 0, n.repeat = 1e3)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
alpha_holder
```

```
## [1] 0.577 0.564 0.529 0.502 0.526
```

We find that there is a large type-1 error for each sample size. However, in fairness to Dr. Manning, we have here used model-based variance estimates, which are not valid in this heteroscedastic setting. A more fair evaluation would use sandwich-based variance estimators (which the author of this homework assignment did not think of until he sat down to make the key). Let's see what happens when we use sandwich variance estimators from the `sandwich` package.

```
library(sandwich)
sim_for_one_n <- function(n.sim, Psi, n.repeat) {
    reject_holder <- rep(NA, n.repeat)
    for (it in 1:n.repeat) {
        df_sim <- simu_Q1(n.sim, Psi)
        newX <- data.frame(Y = 1, A = 1, W1 = 1, W2 = 1)

        manning_fit <-
            SL.manningGLM(Y = df_sim$Y, X = df_sim[,2:4],
                                    newX = newX,
                                   family = gaussian(), kCut = 3,
                                    obsWeights = rep(1, n.sim),
                                    lambdaCut = c(0.5, 1.5, 2.5))

        reject_holder[it] <- check_manning(manning_fit, robustVar = TRUE)
    }
    reject_holder[is.na(reject_holder)] <- FALSE
    alpha <- mean(reject_holder)
    alpha
}

# simulate for a grid of n, I added more sample sizes out of curiousity
n.grid <- c(100, 500, 1000, 5000, 10000)

alpha_holder <- rep(NA, length(n.grid))
for (it2 in 1:length(n.grid)) {
    print(it2)
    alpha_holder[it2] <- sim_for_one_n(n.sim = n.grid[it2],
                                       Psi = 0, n.repeat = 1e3)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
## [1] 5
```

```
alpha_holder
```

```
## [1] 0.058 0.047 0.066 0.050 0.043
```

We can add Monte Carlo 95% confidence intervals to these estimates:

```
cbind(
    alpha_holder,
    alpha_holder + sqrt(alpha_holder*(1-alpha_holder)/1000)%o%c(-1.96,1.96)
    )
```

```
##      alpha_holder
## [1,]        0.058 0.04351243 0.07248757
## [2,]        0.047 0.03388249 0.06011751
## [3,]        0.066 0.05061132 0.08138868
## [4,]        0.050 0.03649163 0.06350837
## [5,]        0.043 0.03042679 0.05557321
```

Surprisingly, the adaptive selection of the link function and family does not appear to have a large effect on the type-1 error of the procedure.

2. Repeat question 1 using the incorrect main terms regression `formula = Y ~ A + W1 + W2` in your calls to `glm`.

The new function for this question is `manning_Q2.R`, wherein the regression formulas have been modified.

```
#=====================================================
# Adaptive GLM algorithm of Manning (2001)
#=====================================================
SL.manningGLM <- function(Y, X, newX, family, obsWeights,
                          kCut = 3, # kurtosis cutpoint
                          lambdaCut = c(0.5,1.5,2.5), # skew cutpoint
                          startNLS=0, # starting values for NLS?
                          ...){
    if(family$family=="gaussian"){
        require(moments)

        # step 1: first do ols on log scale
        logY <- log(Y)
        fit.logGLM <- glm(logY ~ A + W1 + W2, data=X,
                          family=family, weights=obsWeights)

        mu <- predict(fit.logGLM, type="response", newdata=X)
        # step 2: compute residual
        resid <- logY - mu

        # step 3: check kurtosis of residuals
        k <- kurtosis(resid)

        # by default use these methods
        # some of the other GLMs are unstable and if they fail, this
        # algorithm returns log OLS + smearning estimate
        # the smearing estimate is used to go back to the original from the
        # log-transformed scale -- Duan (1983)
        pred <- exp(predict(fit.logGLM, type="response",newdata=newX))*
            mean(exp(resid))
```

```
        fit <- list(object=fit.logGLM, mean(exp(resid)))
        class(fit) <- "SL.logOLS.smear"

        try({
            if(k < kCut){
                # step 4: kurtosis < 3

                # park test
                fit.initGLM <- glm(Y ~ A + W1 + W2, data=X,
                                   weights=obsWeights, family=gaussian(link="log"))

                # step 5: compute residual
                muPark <- predict(fit.initGLM, type="response", newdata=X)
                resid2Park <- (Y - muPark)^2

                # step 5: fit epsilon^2 on Y^hat
                fit.parkGLM <- glm(resid2Park ~ muPark,
                                   family=gaussian(link="log"))
                lambda1 <- fit.parkGLM$coef[2]

                if(lambda1 < lambdaCut[2]){
                    # 1) lamba_1 (0, 1.5)
                    # use poisson glm
                    suppressWarnings(
                        fit.poisGLM <- glm(Y ~ A + W1 + W2, data=X,
                                           weights=obsWeights,
                                           family=poisson(link='log'),
                                           control=list(maxit=100))
                    )
                    pred <- predict(fit.poisGLM, newdata=newX, type="response")
                    fit <- list(object=fit.poisGLM)
                    class(fit) <- "SL.manningGLM"
                }else
                    if(lambda1 < lambdaCut[3] & lambda1 >= lambdaCut[2]){
                        # 2) lamba_1 (1.5, 2.5)
                        # use gamma glm
                        fit.gammaGLM <- glm(Y ~ A + W1 + W2, data=X,
                                            weights=obsWeights,
                                            family=Gamma(link='log'),
                                            control=list(maxit=100))
                        pred <- predict(fit.gammaGLM, newdata=newX, type="response")
                        fit <- list(object=fit.gammaGLM)
                        class(fit) <- "SL.manningGLM"
                    }else if(lambda1 > lambdaCut[3]){
                        # 3) lamba_1 > 2.5

                        # use inverse gaussian glm -- not very stable
                        fit.invGaussianGLM <- glm(Y ~ A + W1 + W2, data=X,
                                                  weights=obsWeights,
                                                  family=inverse.gaussian(link="log"),
                                                  control=list(maxit=100))
                        pred <- predict(fit.invGaussianGLM, newdata=newX, type="response")
                        fit <- list(object=fit.invGaussianGLM)
```

```
                        class(fit) <- "SL.manningGLM"
                    }
                }
            }, silent=TRUE)
    }else{
        stop("SL.manningGLM doesn't work with binomial family.")
    }
    out <- list(pred = pred, fit=fit)
    return(out)
}

# predict function
predict.SL.manningGLM <- function(object, newdata,...){
    if(!is.list(object$object)){
        pred <- predict(object=object$object, newdata=newdata, type="response")
    }else{
        pred <- predict(object=object$object, newdata=newdata, type="response")
    }
    pred
}
```

Let's go ahead and give the algorithm the best shot at return correct inference by setting `robustVar = TRUE` when we compute the p-value. This will isolate the effect of model misspecification on bias (vs. incorrect standard error estimation).

```
sim_for_one_n <- function(n.sim, Psi, n.repeat) {
    reject_holder <- rep(NA, n.repeat)
    for (it in 1:n.repeat) {
        df_sim <- simu_Q1(n.sim, Psi)
        newX <- data.frame(Y = 1, A = 1, W1 = 1, W2 = 1)

        manning_fit <- SL.manningGLM(Y = df_sim$Y, X = df_sim[,2:4], newX = newX,
                                     family = gaussian(), kCut = 3, obsWeights = rep(1, n.sim),
                                      lambdaCut = c(0.5, 1.5, 2.5))
        reject_holder[it] <- check_manning(manning_fit, robustVar = TRUE)
    }
    reject_holder[is.na(reject_holder)] <- FALSE
    alpha <- mean(reject_holder)
    alpha
}

# simulate for a grid of n
n.grid <- c(100, 500, 1000, 5000, 10000)

alpha_holder <- rep(NA, length(n.grid))
for (it2 in 1:length(n.grid)) {
    alpha_holder[it2] <- sim_for_one_n(n.sim = n.grid[it2], Psi = 0, n.repeat = 1e2)
}
```

```
# type-1 err and 95% MC CI
cbind(
    alpha_holder,
    alpha_holder + sqrt(alpha_holder*(1-alpha_holder)/1000)%o%c(-1.96,1.96)
)
```

```
##      alpha_holder
## [1,]         0.22 0.1943247 0.2456753
## [2,]         0.49 0.4590159 0.5209841
## [3,]         0.63 0.6000755 0.6599245
## [4,]         0.97 0.9594269 0.9805731
## [5,]         1.00 1.0000000 1.0000000
```

We now clearly see that the type-1 error converges to one with sample size. Again, this is due to the bias induced by using an incorrectly-specified parameteric model.

3. Use the above data generating mechanism for W_1, W_2, and Y, but now suppose that $A \sim Bernoulli(1/2)$. Using simulations, assess the probability of rejecting the null hypothesis of no treatment effect for all combinations of $\psi \in \{0, 0.05, 0.1, 0.25, 0.5\}$ and $n \in \{100, 500, 1000\}$. Compare these probabilities to the probability of rejecting the null hypothesis using a simple two-sample t-test with unequal variances (i.e., reject when `t.test(Y~A)$p.value < 0.05`).

The data generating distribution is changed for A. The simulation code for generating data.frame is changed accordingly in `simu_Q3.R`.

```
simu_Q3 <- function(n.sim, Psi) {
    library(rje)
    W1 <- runif(n.sim, -4, 4)
    W2 <- rbinom(n.sim, 1, prob = .5)
    A <- rbinom(n.sim, 1, prob = .5)
    logY <- 5 + Psi*A + W1 + W2 - W1*W2 - W1^2/8 + rnorm(n.sim)
    Y <- exp(logY)

    return(data.frame(Y, A, W1, W2))
}
```

```
sim_for_one_n <- function(n.sim, Psi, n.repeat) {
    reject_holder <- rep(NA, n.repeat)
    reject_holder_t <- rep(NA, n.repeat)
    for (it in 1:n.repeat) {
        df_sim <- simu_Q3(n.sim, Psi)
        newX <- data.frame(Y = 1, A = 1, W1 = 1, W2 = 1)

        manning_fit <- SL.manningGLM(Y = df_sim$Y, X = df_sim[,2:4], newX = newX,
                                     family = gaussian(), kCut = 3,
                                    obsWeights = rep(1, n.sim),
                                      lambdaCut = c(0.5, 1.5, 2.5))
        reject_holder[it] <- check_manning(manning_fit, robustVar = TRUE)
        reject_holder_t[it] <- t.test(df_sim$Y~df_sim$A)$p.value < 0.05
    }
    reject_holder[is.na(reject_holder)] <- FALSE
    alpha <- mean(reject_holder)

    reject_holder_t[is.na(reject_holder_t)] <- FALSE
    alpha_t <- mean(reject_holder_t)

    c(alpha, alpha_t)
}

# simulate for a grid of n
# and Psi
n.grid <- c(100, 500, 1000, 5000, 10000)
```

```
Psi.grid <- c(0,0.05,0.1,0.25,0.5)

alpha_holder <- matrix(NA, nrow = length(n.grid), ncol = length(Psi.grid))
alpha_holder_t <- matrix(NA, nrow = length(n.grid), ncol = length(Psi.grid))

for (it3 in 1:length(Psi.grid)) {
    for (it2 in 1:length(n.grid)) {
        print(it3)
        alpha_holder[it2, it3] <- sim_for_one_n(n.sim = n.grid[it2],
                                                Psi = Psi.grid[it3],
                                                n.repeat = 1e3)[1]
        alpha_holder_t[it2, it3] <- sim_for_one_n(n.sim = n.grid[it2],
                                                  Psi = Psi.grid[it3],
                                                  n.repeat = 1e3)[2]
    }
}
```

```
## [1] 1
## [1] 1
## [1] 1
## [1] 1
## [1] 1
## [1] 2
## [1] 2
## [1] 2
## [1] 2
## [1] 2
## [1] 3
## [1] 3
## [1] 3
## [1] 3
## [1] 3
## [1] 4
## [1] 4
## [1] 4
## [1] 4
## [1] 5
## [1] 5
## [1] 5
## [1] 5
## [1] 5
```

The probability of rejecting the null hypothesis is shown in the output below. Each row corresponds to sample size $n = \{100, 500, 1000, 5000, 10000\}$, and each column corresponds to $\psi = \{0, 0.05, 0.1, 0.25, 0.5\}$. Thus, the first column illustrates the type-1 error, while subsequent columns represent the power of the test under the alternative hypothesis that $\psi_0 = \psi$.

```
alpha_holder
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 0.064 0.071 0.089 0.146 0.346
## [2,] 0.063 0.061 0.118 0.369 0.875
## [3,] 0.051 0.079 0.164 0.599 0.981
## [4,] 0.055 0.162 0.464 0.996 1.000
```

```
## [5,] 0.039 0.258 0.724 1.000 1.000
```

Here is the same output for the t-test.

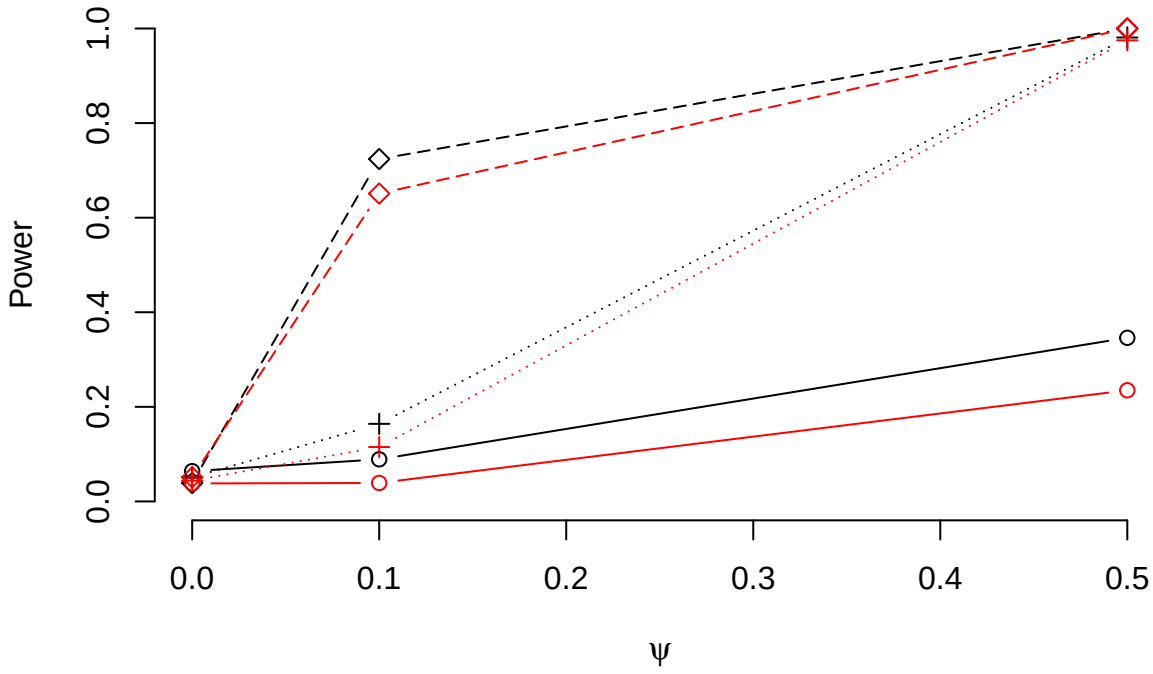
```
alpha_holder_t
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 0.038 0.041 0.039 0.087 0.235
## [2,] 0.046 0.041 0.081 0.290 0.780
## [3,] 0.044 0.056 0.115 0.518 0.975
## [4,] 0.058 0.125 0.417 0.986 1.000
## [5,] 0.051 0.233 0.651 1.000 1.000
```

We can plot the results together. Here I'm plotting $n = \{100, 1000, 10000\}$ for simplicity.

```
plot(alpha_holder[1,c(1,3,5)] ~ Psi.grid[c(1,3,5)], bty="n",
     xlim=range(Psi.grid),pch=1,
     ylim=c(0,1), xlab=expression(psi), ylab="Power", type="b",lty=1)
for(i in c(3,5)){
    points(alpha_holder[i,c(1,3,5)] ~ Psi.grid[c(1,3,5)], type="b",
           lty=i,pch=i)
}

# add t-test points in red
for(i in c(1,3,5)){
    points(alpha_holder_t[i,c(1,3,5)] ~ Psi.grid[c(1,3,5)], type="b",
           lty=i,col=2,pch=i)
}
```



Here we find that, as above, in this circumstance the inflation of type-1 error due to the adaptive selection of the glm is small – the Manning algorithm has approximately correct type-1 error at all sample sizes. The t-test too has correct type-1 error at all sample sizes. However, as we move away from the null hypothesis, we find that, even though the Manning algorithm uses a misspecified parametric model, the power of the Manning method (which adjusts for covariates) is larger than the unadjusted t-test. This example is an illustration of the power of randomization. Even with a misspecified parametric model, we may gain efficiency

by adjusting for covariates. The adaptive selection of the parametric model in this case had little effect on the type-1 error (once the proper standard error estimates were used). However, we cannot expect this to generally be the case. Model selection procedures that use more ad-hoc methods (e.g., selecting models based on p-values) will increase type-1 error.

4. Comment on your findings. What can you conclude about inference based on an adaptively-selected GLM in an observational study? What can you conclude about inference based on an adaptively-selected GLM in a randomized trial?

In observational studies it is necessary to adjust for covariates to control for confounding. However, adjusting for covariates through parametric regression comes at a large risk. If the parametric model can be properly specified (as in part 1), then we will draw proper inference. However, in practice correctly specifying a parametric model is nearly impossible when covariates are continuous and/or high dimensional. Therefore these methods come at substantial risk. This problem also – and I admit I am a bit surprised about this – illustrated that not all adaptive selection of models will lead to incorrect inference. However, I stress again that this is not a general statement, but is specific to this particular algorithm. An interesting exercise would be to repeat the analysis of part 1, but each time after step 1 of the algorithm, only proceed if the test of the coefficient for A is not significant. One could imagine that this is a reasonable guess at how this algorithm could be abused in practice, i.e., researchers would only look to select a different GLM if they did not find 'a significant' result based on the first one they looked at.

Part 3 illustrated that randomization can buy you a lot in terms of flexibility to adjust for covariates. Even misspecified parametric models may draw more powerful inference than methods that ignore covariates. However, this is not true in the case of GLMs with other link functions (e.g., logistic). We also do not advocate for adaptive selection of a GLM in general, though the simulation appears to bear out that Mannings proposed algorithm does not drastically increase type-1 error (provided proper standard error estimators are used).