Simulation Study
Homework 9
ST758

Meng Yang

December 8, 2014

Instructor: Hua Zhou

1 Empirical Bayesian vs MLE

1.1 Statement of problem

In HW6/HW7, we implemented algorithms for the MLE of the Dirichlet-Multinomial distribution. The professor talks about possible use of this for empirical Bayes (EB) estimation of the multinomial parameters from multiple populations. Many people will suspect why shall we borrow information across populations even when those populations are totally unrelated? This question seems to be very simple at first glance, but very difficult to prove mathematically. In this simulation study, Monte Carlo method will be used to make a numerical comparison between Empirical Bayesian estimator and MLE.

1.2 Questions to be addressed

How to make an appropriate design? We generate S replicates in total with I populations in each replicate under some specific setting (see figure 1).

How to choose appropriate S? In other word, How well do the Monte Carlo quantities approximate properties of the true sampling distribution of the estimator? Each data set yields a draw from the true sampling distribution, so S is the sample size on which estimates of mean, bias, SD, etc. of this distribution are based. We can select appropriate sample size (number of data sets S) in order to achieve acceptable precision of the approximation in the usual way. Acceptable precision can make Monte Carlo quantities significant which means the estimates we get are convincing.

How to estimate the estimation error from MLE and Empirical Bayesian? Here, we introduce the concept of the total variance:

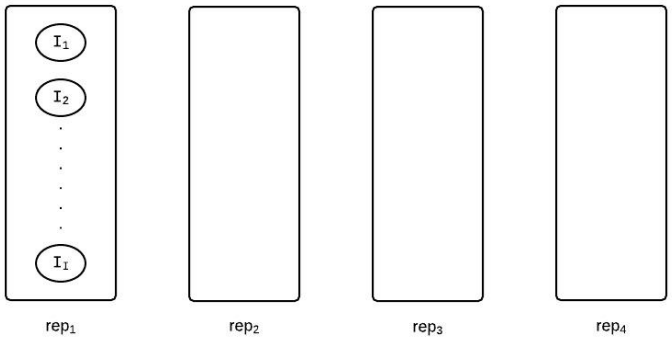

Figure 1: Monte Carlo design

$$Err_s^{MLE} = \frac{1}{I}\sum_{i=1}^{I} ||\hat{\boldsymbol{p}}_i^{MLE} - \boldsymbol{p}_i||_{TV}$$

$$Err_s^{EB} = \frac{1}{I}\sum_{i=1}^{I} ||\hat{\boldsymbol{p}}_i^{EB} - \boldsymbol{p}_i||_{TV}$$

I means the number of populations, s means the number of Monte Carlo replicates.

1.3 Details

Description of the design of the experiment We generate S replicates in total with I populations in each replicate under some specific setting (see figure 1). Within each combination of S and I, we choose a fixed batch size $N = 20$ for multinomial distribution, from which we can get a MLE and Empirical Bayesian estimator.

Computational details For MLE, $\hat{\boldsymbol{p}}_i^{MLE} = \dfrac{\boldsymbol{x}_i}{N}, i = 1, 2, 3, ..., I$ for each replicate. For Empirical Bayesian :$\hat{\boldsymbol{p}}_i^{EB} = \dfrac{\boldsymbol{x}_i + \hat{\boldsymbol{\alpha}}}{N + |\hat{\boldsymbol{\alpha}}|}, i = 1, 2, 3, ..., I.$ $\boldsymbol{x}_i$ is the count vector for each population, N the batch size. We implement the design in figure 1 under every possible combination of I $\times$ d, where I = 10, 20, 50 and d= 2,3,5,10. For each population, in order to get Empirical Bayesian estimator, we use Newton algorithm to get an estimated $\alpha's$ of the prior Dirichlet distribution and get the bayesian rule. And then we can get an estimation error for each population and each replicate. Take the mean across the all the populations in for each replicate, then we have S estimation errors. This is our estimation error sample distribution. The main part code is below and the full code is in the Appendix 1.

R code: (Main Part)

```
MC_MLE <- function(I, d, S , N = 20) {
this is a function to calculate estimator for P using MLE method
# Args: I: number of populations
# d: number of categories
# S: number of replicates
# N: batch size
# Output Err_MLE: estimated error for MLE estimator
#
set.seed(100)
Err_MLE <- matrix(0, S)
temp <- matrix(0, I, 1)
for ( s in 1:S) {
p <- matrix(runif(d * I), I, d)
temp <- rowSums(p)
p <- p / temp
data <- rmultinomial(I, N, p)
pr_MLE <- data / N
Err_MLE[s] <- sum(abs(pr_MLE - p)) / (2 * I)
}
return(Err_MLE)
}

MC_EB <- function(I, d, S, N = 20) {
#
# This is a function to calculate estimator for P using EB method
# Args: I: number of populations
# d: number of categories
# S: number of replicates
# N: batch size
# Output Err_EB: estimated error for EB estimator
#
set.seed(100)
# initialize matrix
```

```
data < − matrix(0, d, I)
temp1 < − matrix(0, I, 1)
temp < − matrix(0, S, 1)
# loop over different replicate
for(s in 1:S) {
p < − matrix(runif(d * I), I, d)
temp1 < − rowSums(p)
p < − p / temp1
data < − rmultinomial(I, N, p)
# Newton method to estimate alpha
alpha < − dirmultfit(data)
pr_EB < − t((data + alpha) / (N + sum(t(alpha))))
temp[s] < − sum(abs(pr_EB - t(p)))/(2 * I)
}
Err_EB < − temp
return(Err_EB)
}

MLE < − function(S, I, d) {
# This is a function to calculate estimator error for different I and d
# Args: I: number of populations (vector)
# d: number of categories (vector)
# S: number of replicates
# Output Err_MLE: estimated error for MLE estimator for different replicate
#
tabledata_MLE < − matrix(0, S, length(I) * length(d))
k = 1
for (i in 1 : length(I)){
for (j in 1 : length(d)) {
tabledata_MLE[, k] < − MC_MLE(I[i], d[j], S)
k < − k + 1
}
}
return(tabledata_MLE)
```

```
}

EB <- function(S, I, d){
# This is a function to calculate estimator error for different I and d
# Args: I: number of populations (vector)
# d: number of categories (vector)
# S: number of replicates
# Output Err_EB: estimated error for EB estimator for different replicate
#
tabledata_EB <- matrix(0, S, length(I)*length(d))
k = 1
for (i in 1:length(I)){
for (j in 1:length(d)) {
tabledata_EB[, k]<- MC_EB(I[i], d[j], S)
k <- k + 1
}
}
return(tabledata_EB)
}
```

Results and Analysis

In this study, we round our error to two digits. Hence, we need to choose an appropriate S to guarantee standard error of estimated error is less than 0.005. This indicates that the estimation errors we get from Monte Carlo approximation for both estimators are reliable. We choose S=100, so that every standard error of estimated error in our result is less than 0.005.

EB, d=2: the error is exact the same across all values of I.
MLE, d=2: the error is almost the same across all values of I.
EB, d=3: the error is much bigger when I=10 than the error when I=20, 50.
MLE, d=3: the error is almost the same across all values of I.
EB, d=5: the error is almost the same across all values of I.
MLE, d=5: the error is exact the same across all values of I.
EB, d=10: the error is exact the same across all values of I.
MLE, d=10: the error is almost the same across all values of I.

d=2, I=10,20, EB and MLE will give the same error, but d=2, I=50, EB seems to give smaller error.

	I=10	I=20	I=50
d=2(EB)	0.070	0.070	0.070
d=2, SE(EB)	0.002	0.002	0.001
d=2(MLE)	0.070	0.070	0.080
d=2, SE(MLE)	0.002	0.002	0.001
d=3(EB)	0.110	0.100	0.100
d=3, SE(EB)	0.002	0.001	0.001
d=3(MLE)	0.110	0.120	0.110
d=3, SE(MLE)	0.003	0.001	0.001
d=5(EB)	0.150	0.140	0.140
d=5, SE(EB)	0.002	0.001	0.001
d=5(MLE)	0.160	0.160	0.160
d=5, SE(MLE)	0.002	0.001	0.001
d=10(EB)	0.180	0.180	0.180
d=10, SE(EB)	0.001	0.001	0.001
d=10(MLE)	0.250	0.240	0.250
d=10, SE(MLE)	0.002	0.001	0.001

Figure 2: Numerical result

d=3 I=20 EB and MLE will give the same error, but d=3 I=20, 50, MLE will give larger error than EB does.
d=5 I=10, 20, 50, EB and MLE will give the same error.
d=10 I=10 20, 50, EB will give smaller error than MLE does.

Figure 3 gives a more clear result.

Conclusion
d=2, I=10, 20, the accuracy of EB and MLE are the competitive, but for I=50 d=2, EB seems to be slightly accurate than MLE.

d=3, I=10, the accuracy of EB and MLE are the almost the same, but for I=20, 50, EB seems to be more accurate than MLE.

d=5, I=10, 20 ,50, the accuracy of EB and MLE are same.

d=10, I=10, 20, 50, the accuracy of EB are more accurate than MLE.

Thus, we can see, $d = 5$ seems to be a border line. When d is larger than or equal to 5, the EB seems to be a more desirable estimator than MLE. When d is smaller than 5, the accuracy of MLE and EB are competitive, even though

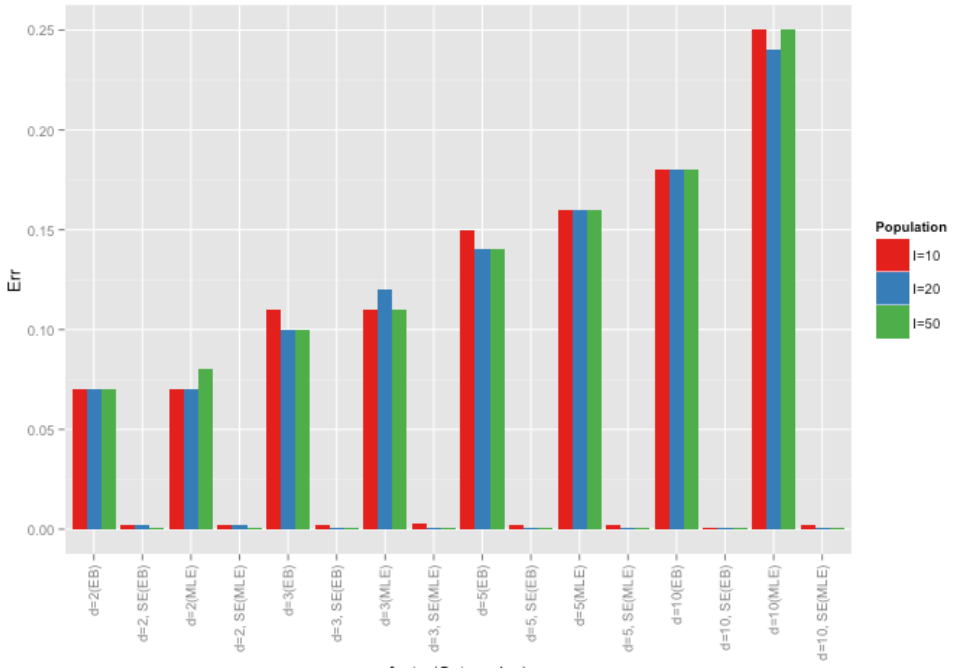

Figure 3: Numerical result

there are still some times EB is slightly better than MLE. Only based on our cases, it is not bad to borrow information from some uncorrelated populations. This is reasonable because Empirical Bayesian method provides an compromise between the multinomial model and the information from our past experiences. Since our design is quite limited, it is still hard to claim one method dominate the other one in all the cases.

2 LASSO regression, Ridge regression and OLS

2.1 Statement of problem

As usual , we assume the model: $y = f(X)+\epsilon$. In regression analysis, our major goal is to come up with some good regression functions so that $X\hat{\beta}$ is equal to response vector. As we all know, this problem can be solved by least square method easily and it has well-known properties (Gauss-Markov, ML). But can we do better? We can decompose our prediction error into two parts, one part is $bias^2$, the other is variance. Such decomposition is called bias-variance tradeoff. From figure 4 we find that, sometimes it might be better to introduce a little bias in our estimate for β, which will lead to a substantial decrease in variance, and hence to a substantial decrease in prediction error. This why Lasso and Ridge are introduced. To control variance, we might regularize the coefficients. LASSO and Ridge regression are two ways to regularize the coefficients. The question is which method is better? In this study, we will implement a naive simulation study comparing Ridge regression, LASSO, and OLS regression.

2.2 Questions to be addressed

How to make an appropriate design We generate S replicates in total. For each replicate, we generate a list of data(X and y) to do the regression by each method.

How to choose appropriate S In other word, How well do the Monte Carlo quantities approximate properties of the true sampling distribution of the estimator? Each data set yields a draw from the true sampling distribution, so S is the sample size on which estimates of mean, bias, SD, etc. of this distribution are based. We can select appropriate sample size (number of data sets S) that will achieve acceptable precision of the approximation in the usual way. Acceptable precision can make Monte Carlo quantities significant which means the estimated error and prediction error are convincing.

How to estimate the estimation error and prediction errorr For each method

$$Err_s^{method} = \sqrt{\frac{\sum_{j=1}^{p}(\hat{\beta}_j - \beta_j)^2}{p}},\ \beta_j \text{ is the j-th component of } \boldsymbol{\beta}_s$$

$$PredictionErr_s^{method} = \sqrt{\frac{\sum_{j=1}^{p}(\hat{y}_j - y_j)^2}{n}},\ y_j \text{ is the j-th component of } \boldsymbol{y}$$

n means the number of data, s means Monte Carlo replicate and p means the number parameters.

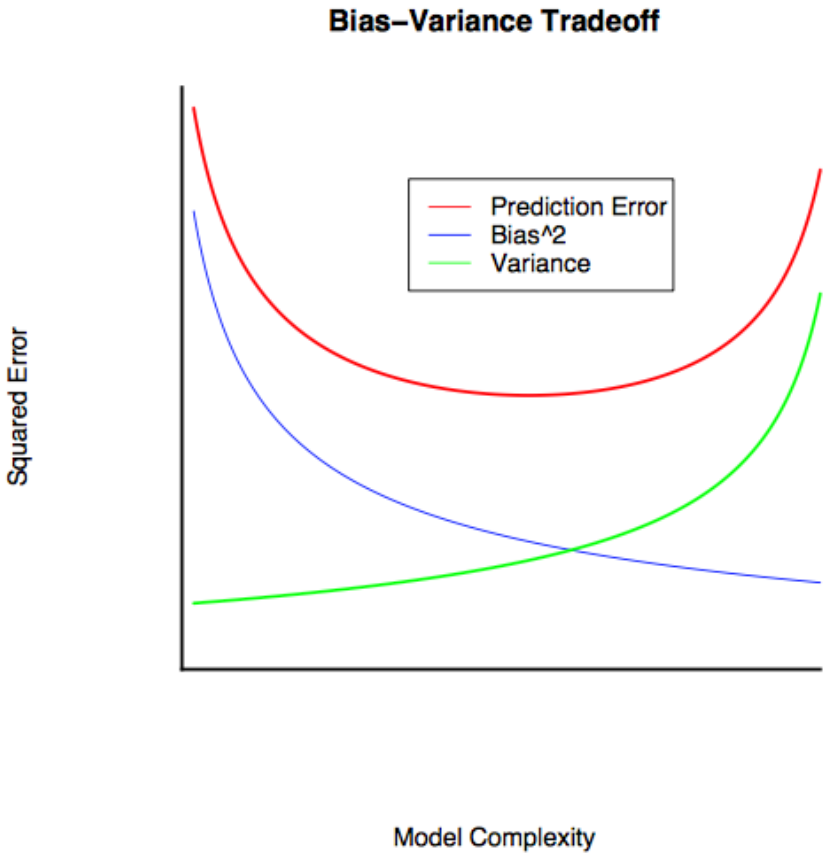

Figure 4: variance-bias tradeoff

2.3 Details

Description of the design of the experiment We generate S replicates in total under each possible combination of the number of data and the level of noise (see figure 5). Within each replicate, a data sets will be generated and used to do regression analysis. Then we can get three parameter estimates from three regression methods with which we can calculate the estimation error and prediction error for each method respectively. We repeat this procedure S times, and end up with S numbers for both estimation errors and prediction errors for three methods. Then we get the sample distribution of estimation error and prediction error.

In this study, we round our error to two digits. Hence, we need to choose an appropriate S to guarantee standard error of estimated error is less than 0.005. This indicates that the estimated errors we get from Monte

Carlo approximation for both estimators are reliable. We choose S=700, so that every standard error of estimated error in our result is less than 0.005.

we will break down this simulation study into several tasks:

1 Generating the data.

2 Carrying out Ridge regression.

3 Carrying out the LASSO .

4 Carrying out OLS .

5 Comparing the three methods.

Computational details We use cv.glmnet() in R to implement LASSO regression and Ridge regression and use lm() to fit OLS regression. In this study we choose a fixed dimension of parameter which is 10. We will consider the case when N=50, 100, 150, 200 and $\sigma = \sqrt{1}$, $\sqrt{0.8}$, $\sqrt{0.5}$, $\sqrt{0.1}$. Also, to simplify our study, we assume the correlation between different predictors are zero. The main part is below and the full code is in the Appendix 2.

R code: (Main Part)

```
# Generating data
genData <- function(n ,beta, sigma) {
# This is a function to generate data with size n, beta and noise level of
# sigma
# Args: n : data size
# beta: true parameter
# sigma: noise level
# Outputs: list contains X and Y
p <- length(beta)
X <- matrix(rnorm(n * p), n, p)
X[, 1] <- rep(1, n)
y <- rnorm(n, X %*% beta, sigma ^ 2)
return(list(X = X, y = y))
}

ridge <- function(data, beta) {
# This is a function to find beta using ridge regression
# Args: data: X and Y
```

```
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
require(glmnet)
cvfit <- cv.glmnet(data$X, data$y, family = "gaussian", alpha = 0)
n <- length(data$y)
#BIC <- n*log(residual) + log(n)*fit$df
coeff <- as.numeric(coef(cvfit,s = cvfit$lambda.min))[-2]
residual <- predict(cvfit, newx=data$X) - data$y
prederr <- sqrt(crossprod(residual) / n)
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err"=sqrt(sum(coeff-
beta)^2) / length(beta)) ))
}

lasso <- function(data, beta) {
# This is a function to find beta using LASSO regression
# Args: data: X and Y
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
require(glmnet)
cvfit <- cv.glmnet(data$X, data$y, family = "gaussian", alpha = 1)
n <- length(data$y)
#BIC <- n*log(residual) + log(n)*fit$df
coeff <- as.numeric(coef(cvfit, s = cvfit$lambda.min))[-2]
residual <- predict(cvfit,newx = data$X) - data$y
prederr <- sqrt(crossprod(residual) / n)
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err" = sqrt(sum((coeff
- beta) ^ 2) / length(beta)) ))
}

OLS <- function(data, beta) {
# This is a function to find beta using OLS regression
# Args: data: X and Y
```

```
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
fit <- lm(y~X, data)
coeff <- coef(fit)[-2]
prederr <- sqrt(crossprod(fit$residuals) / length(data$y))
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err"=sqrt(sum((coeff
- beta)^2) / length(beta)) ))
}

set.seed(100)
beta <- runif(10,-5,5)
simulation1 <- function(n, sigma, S, beta, i, j) {
# This is the function do simulation across different combination of
# sigma and n give specific value of S and beta
# Args: n : data size
# sigma: noise level
# S : number of replicates
# beta : true value
# i, j : index to set seed
# Outputs : list contains the estimation
# Err matrix for different method and replicates,
# the prediction err matrix for different method and replicates,
# true beta
p <- length(beta)
results <- array(NA, dim = c(S, p, 3),
dimnames = list(1 : S, 1 : p, c("Lasso", "Ridge", "OLS")))
predictionErr <- Err <- matrix(NA, S, 3)
for (s in 1:S)
{
set.seed((i - 1) * i + (j - 1) * j + s * (s - 1))
# generate data
Data <- genData(n, beta, sigma)
# fit model using three methods
```

```
lassfit <- lasso(Data, beta)
ridgefit <- ridge(Data, beta)
OLSfit <- OLS(Data, beta)
# store results
results[s, , 1] <- lassfit$coefficient
results[s, , 2] <- ridgefit$coefficient
results[s, , 3] <- OLSfit$coefficient
Err[s, 1] <- lassfit$Err
Err[s, 2] <- ridgefit$Err
Err[s, 3] <- OLSfit$Err
predictionErr[s, 1] <- lassfit$PredictionErr
predictionErr[s, 2] <- ridgefit$PredictionErr
predictionErr[s, 3] <- OLSfit$PredictionErr
}
return(list("Err"= Err, "PredictionErr" = predictionErr, "estiamte" =
results, "true" = beta)) # return(Err_s vector) }
```

Results and Analysis

Estimated Error for LASSO	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	0.15	0.12	0.08	0.09
N=50(SE)	0.002	0.001	0.001	0
N=100	0.1	0.08	0.05	0.08
N=100(SE)	0.001	0.001	0	0
N=150	0.08	0.07	0.04	0.08
N=150(SE)	0.001	0.001	0	0
N=200	0.07	0.06	0.04	0.08
N=200(SE)	0.001	0.001	0	0

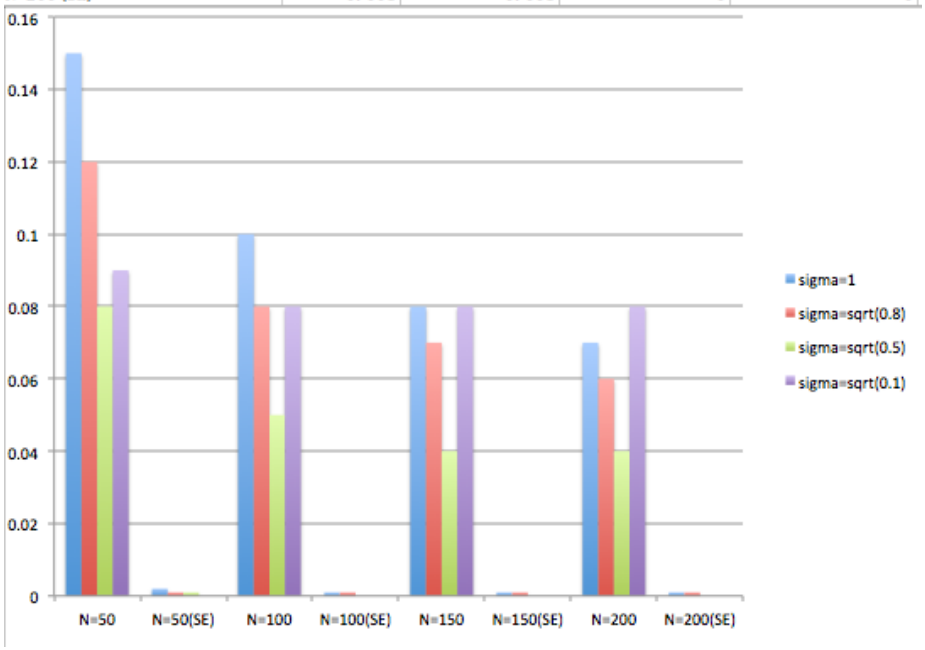

Figure 5: Estimation error and standard error of estimation error for LASSO

From the figure 5, we find that for LASSO regression at each value of N, the estimation error first decreases and then increases when σ is smaller than $\sqrt{0.5}$. All of standard errors of estimation error are very small which means that the estimate of estimation errors are reliable. When $\sigma = 1$, $\sqrt{0.8}$ and $\sqrt{0.5}$, the estimation error is decreasing as N become larger. However, when $\sigma = \sqrt{0.1}$, the estimation error seems to be constant no matter what N we choose.

Estimated Error for Ridge	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	0.24	0.22	0.2	0.19
N=50(SE)	0.002	0.002	0.002	0.001
N=100	0.19	0.18	0.16	0.16
N=100(SE)	0.001	0.001	0.001	0.001
N=150	0.17	0.17	0.15	0.16
N=150(SE)	0.001	0.001	0.001	0.001
N=200	0.16	0.16	0.16	0.15
N=200(SE)	0.001	0.001	0.001	0.001

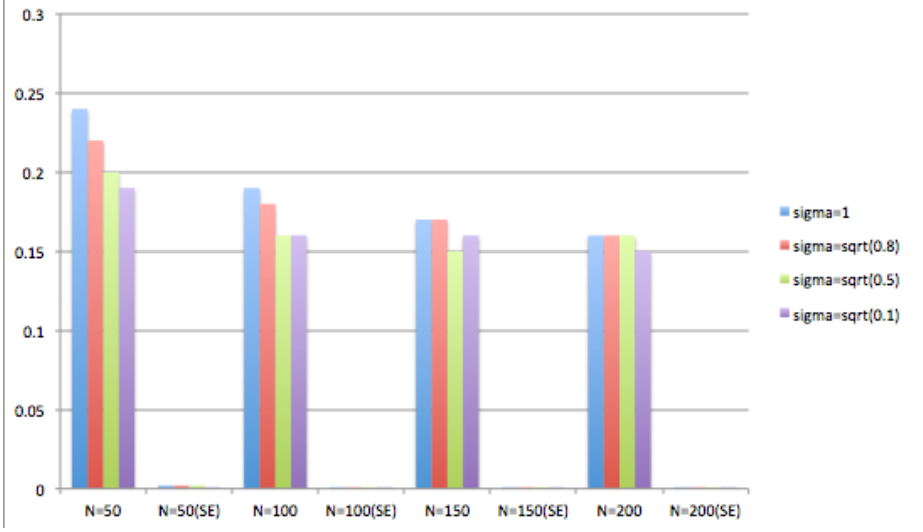

Figure 6: Estimation error and standard error of estimation error for Ridge

From the figure 6, we find that for Ridge regression : For fixed N, the estimation error decreases when σ becomes smaller and smaller. All of standard errors are very small which means that the estimate of estimation errors are reliable. For $\sigma = 1$ and $\sqrt{0.8}$, the estimation error is decreasing as N increasing. For N=100, 150, 200, the estimation error seems to be constant when $\sigma = \sqrt{0.5}$ and $\sqrt{0.1}$.

Estimated Error for OLS	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	0.15	0.12	0.08	0.02
N=50(SE)	0.002	0.001	0.001	0
N=100	0.1	0.08	0.05	0.01
N=100(SE)	0.001	0.001	0	0
N=150	0.08	0.07	0.04	0.01
N=150(SE)	0.001	0.001	0	0
N=200	0.07	0.06	0.04	0.01
N=200(SE)	0.001	0	0	0

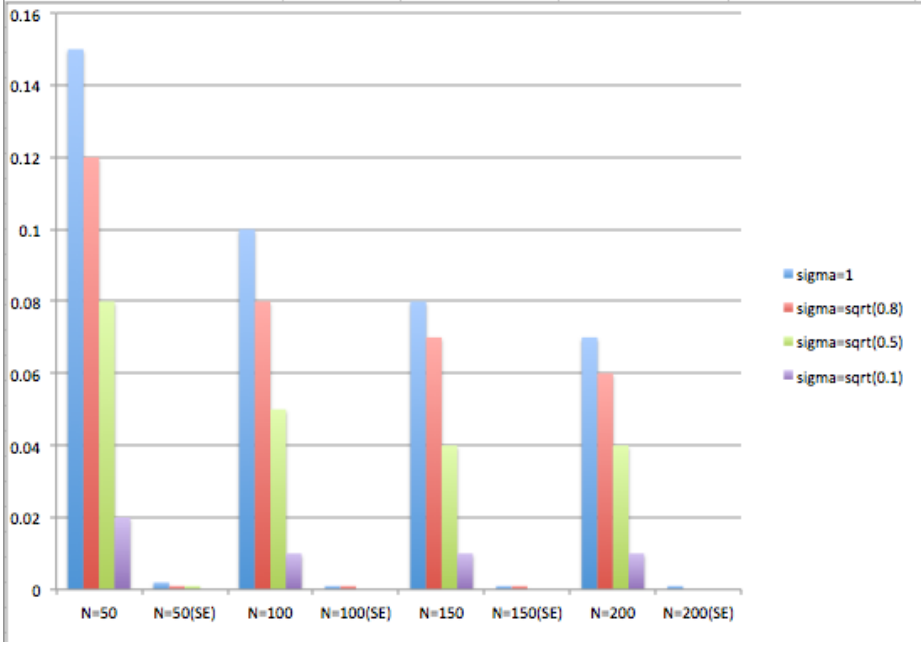

Figure 7: Estimation error and standard error of estimation error for OLS

From the figure 7, we find that for OLS regression, the estimation error decreases as N increase and . In addition, the estimation error is decreasing as σ is decreasing. All of standard errors are very small which means that the estimate of estimation errors are reliable.

Prediction Error for LASSO	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	0.98	0.79	0.49	0.27
N=50(SE)	0.004	0.003	0.002	0.001
N=100	1.01	0.82	0.51	0.27
N=100(SE)	0.003	0.002	0.002	0.001
N=150	1.02	0.82	0.51	0.26
N=150(SE)	0.002	0.002	0.001	0.001
N=200	1.02	0.82	0.51	0.26
N=200(SE)	0.002	0.002	0.001	0.001

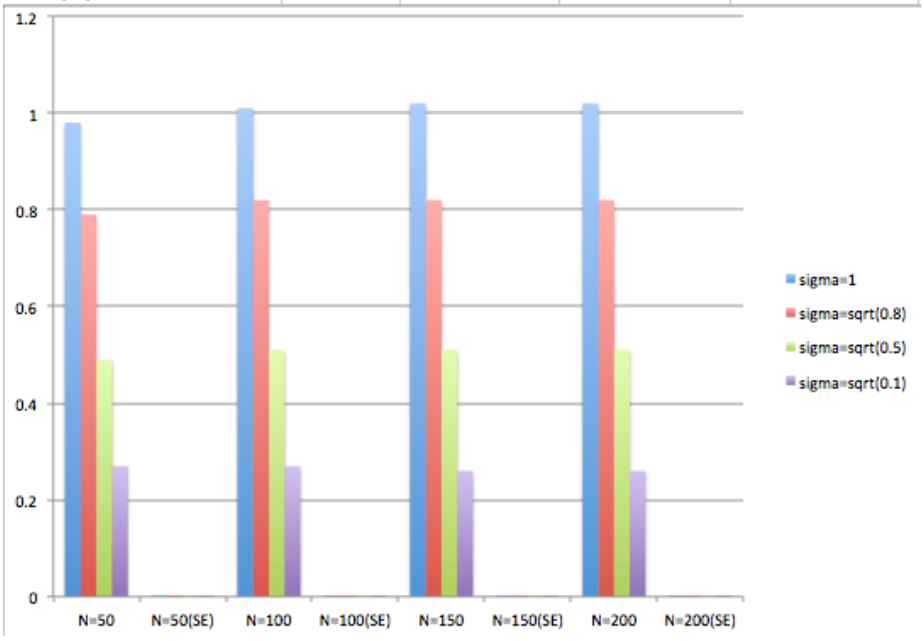

Figure 8: Prediction error and standard error of prediction error for LASSO

From figure 8, we found that for LASSO regression, the prediction error is decreasing when σ is decreasing, but the prediction error seems to be an constant as a function of number of data. All of standard errors are very small and less than 0.005 which means that the estimation of prediction errors are reliable.

Prediction Error for Ridge	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	1.11	0.95	0.73	0.56
N=50(SE)	0.004	0.003	0.003	0.003
N=100	1.12	0.95	0.71	0.51
N=100(SE)	0.003	0.002	0.002	0.002
N=150	1.12	0.94	0.7	0.49
N=150(SE)	0.002	0.002	0.001	0.002
N=200	1.12	0.94	0.7	0.48
N=200(SE)	0.002	0.002	0.001	0.001

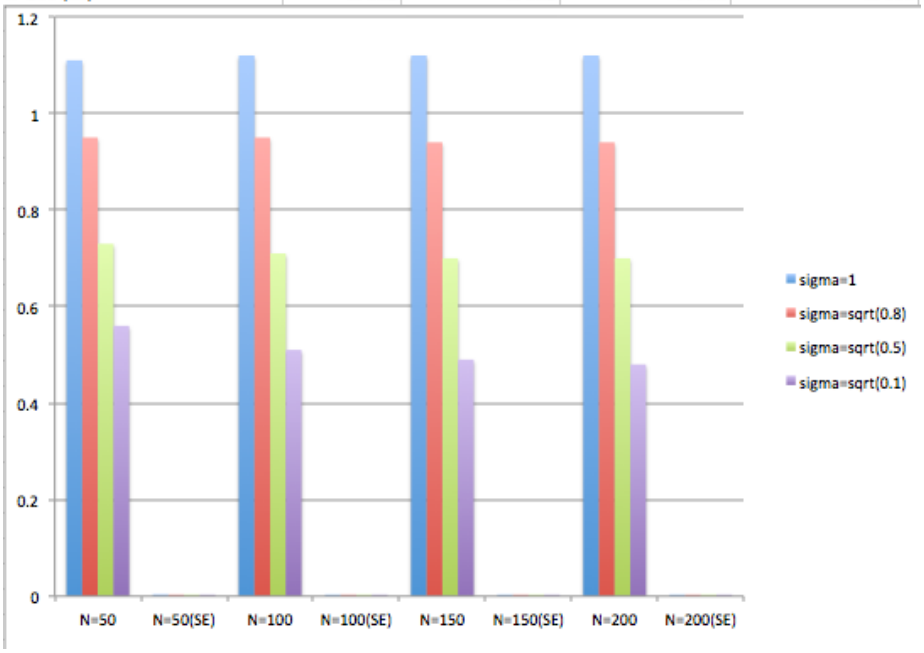

Figure 9: Prediction error and standard error of prediction error for Ridge

From figure 9, we found that for Ridge regression, the prediction error is decreasing when σ is decreasing but the prediction error seems to be constant when the number of data is increasing. All of standard errors are very small and less than 0.005 which means that the estimate of prediction errors are reliable.

Prediction Error for OLS	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50	0.88	0.71	0.44	0.09
N=50(SE)	0.004	0.003	0.002	0.001
N=100	0.94	0.76	0.48	0.09
N=100(SE)	0.003	0.002	0.001	0
N=150	0.97	0.77	0.48	0.1
N=150(SE)	0.002	0.002	0.001	0
N=200	0.97	0.77	0.49	0.1
N=200(SE)	0.002	0.001	0.001	0

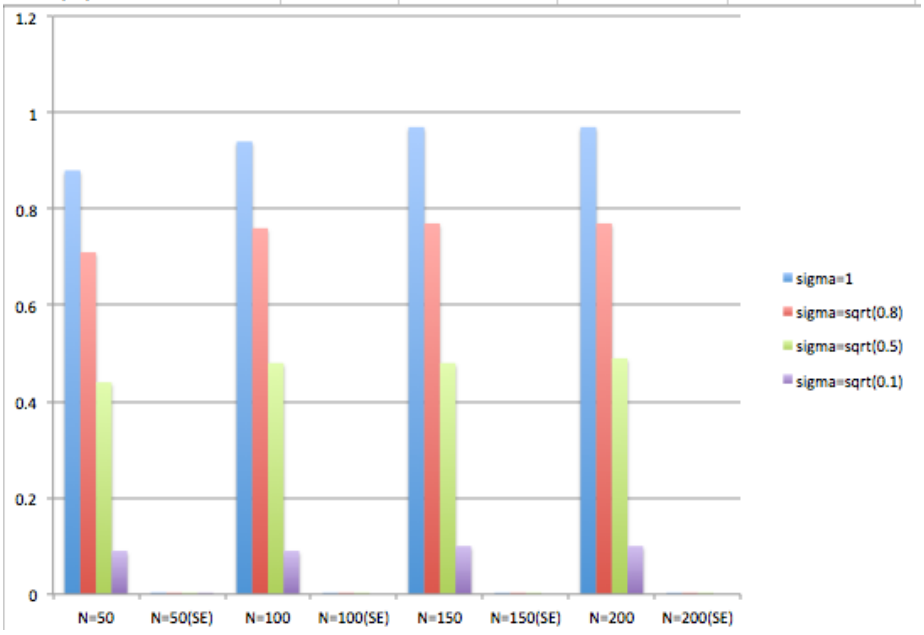

Figure 10: Prediction error and standard error of prediction error for OLS

From figure 10, we found that for Ridge regression, the prediction error is decreasing when σ is decreasing but the prediction error seems to be a slightly increase when the number of data is increasing. All of standard errors are very small and less than 0.005 which means that the estimate of prediction errors are reliable.

Estimated Error	N=50 sigma=1	N=50, sigma=sqrt(0.8)	N=50, sigma=sqrt(0.5)	N=50, sigma=sqrt(0.1)
LASSO	0.15	0.12	0.08	0.09
Ridge	0.24	0.22	0.2	0.19
OLS	0.15	0.12	0.08	0.02

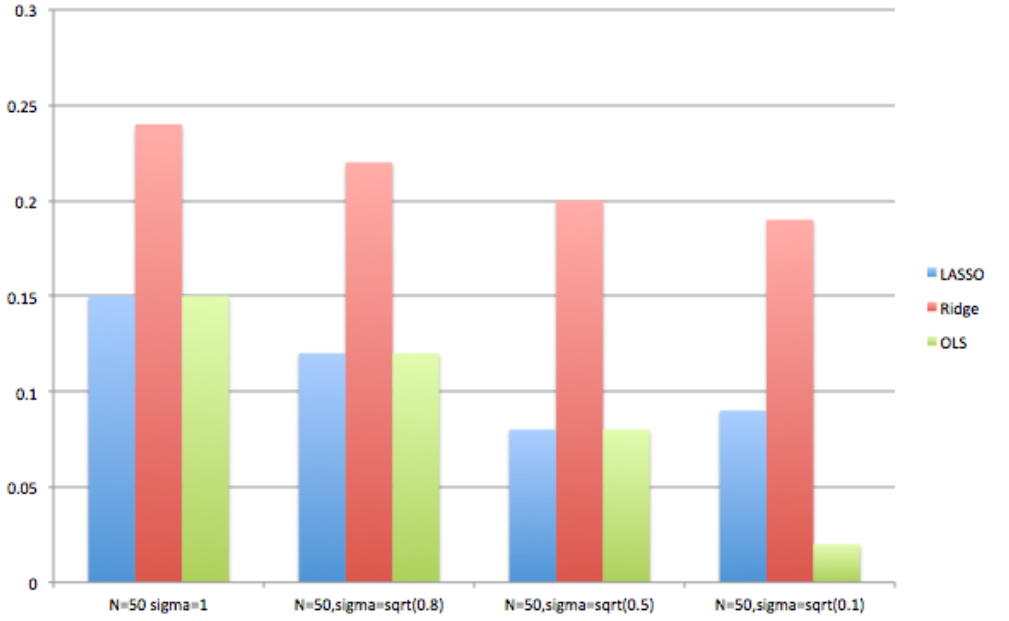

Figure 11: Estimation error N=50 with different values of sigma

From figure 11, When N=50, Ridge regression has the biggest estimation error across all different values of σ When $\sigma = 1$, $\sqrt{0.8}$, $\sqrt{0.5}$, OLS and LASSO will provide the same estimation error, but for $N = 50$ $\sigma = \sqrt{0.1}$ the OLS regression provides the smallest variance.

Estimated Error	N=100 sigma=1	N=100, sigma=sqrt(0.8)	N=100, sigma=sqrt(0.5)	N=100, sigma=sqrt(0.1)
LASSO	0.1	0.08	0.05	0.08
Ridge	0.19	0.18	0.17	0.16
OLS	0.1	0.08	0.05	0.01

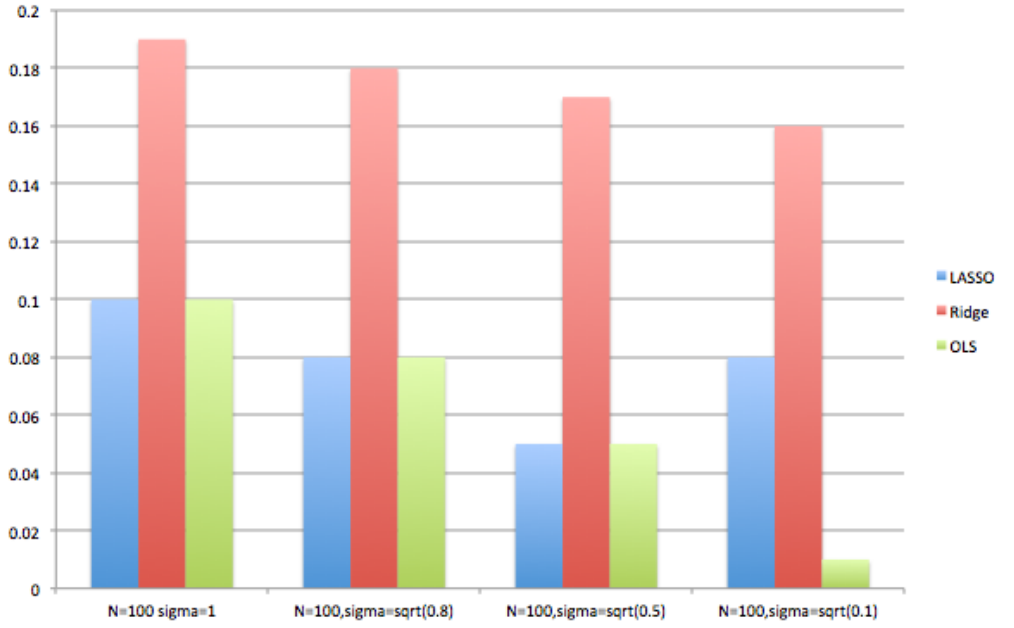

Figure 12: Estimation error N=100 with different values of sigma

From figure 12, When N=100, ridge regression has the biggest estimated error across all different σ When $\sigma = 1$, $\sqrt{0.8}$, $\sqrt{0.5}$, OLS and LASSO will provide the same estimated error, but for $N = 50$ $\sigma = \sqrt{0.1}$ the OLS regression provides the smallest variance.

Estimated Error	N=150 sigma=1	N=150, sigma=sqrt(0.8)	N=150, sigma=sqrt(0.5)	N=150, sigma=sqrt(0.1)
LASSO	0.08	0.07	0.04	0.08
Ridge	0.17	0.17	0.16	0.16
OLS	0.08	0.07	0.04	0.01

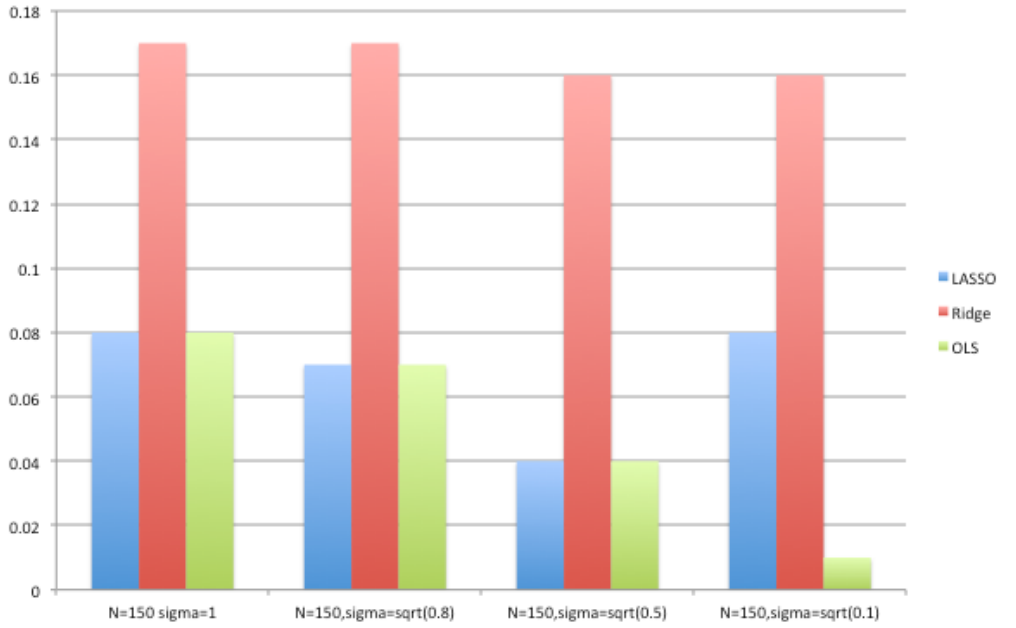

Figure 13: Estimation error N=150 with different values of sigma

From figure 13, When N=150, Ridge regression still has the biggest estimation error across all different values of σ When $\sigma = 1$, $\sqrt{0.8}$, $\sqrt{0.5}$, OLS and LASSO will provide the same estimation error, but for $N = 50$ $\sigma = \sqrt{0.1}$ the OLS regression provides the smallest variance.

Estimated Error	N=150 sigma=1	N=150, sigma=sqrt(0.8)	N=150, sigma=sqrt(0.5)	N=150, sigma=sqrt(0.1)
LASSO	0.07	0.05	0.04	0.08
Ridge	0.17	0.16	0.16	0.15
OLS	0.07	0.06	0.04	0.01

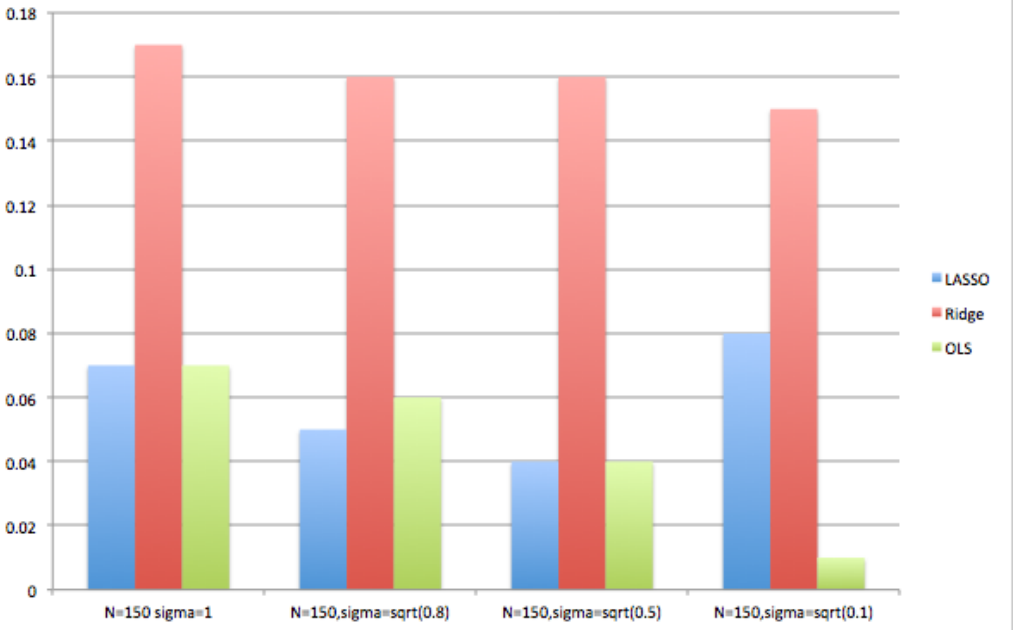

Figure 14: Estimation error N=200 with different values of sigma

From figure 14, When N=200, Ridge regression still has the biggest estimated error across all different values of σ When $\sigma = 1$, $\sqrt{0.8}$, $\sqrt{0.5}$, OLS and LASSO will provide the same estimation error, but for $N = 50$ $\sigma = \sqrt{0.1}$ the OLS regression provides the smallest variance.

Estimated Error	N=50 sigma=1	N=100, sigma=1	N=150, sigma=1	N=200, sigma=1
LASSO	0.15	0.1	0.08	0.07
Ridge	0.24	0.19	0.17	0.17
OLS	0.15	0.1	0.08	0.07

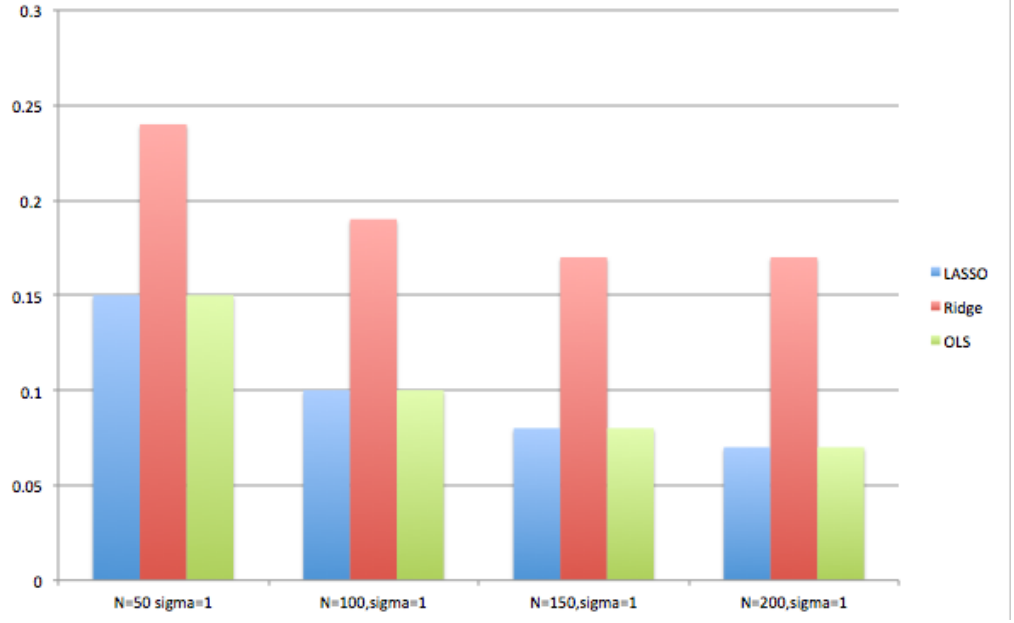

Figure 15: Estimation error sigma=1 with different values of N

From figure 15, when $\sigma = 1$, Ridge regression still has the biggest estimation error across all different values of N. But other two methods will provide the same estimation error.

Estimated Error	N=50 sigma=sqrt(0.8)	N=100, sigma=sqrt(0.8)	N=150, sigma=sqrt(0.8)	N=200, sigma=sqrt(0.8)
LASSO	0.12	0.08	0.07	0.06
Ridge	0.22	0.18	0.17	0.16
OLS	0.12	0.08	0.07	0.06

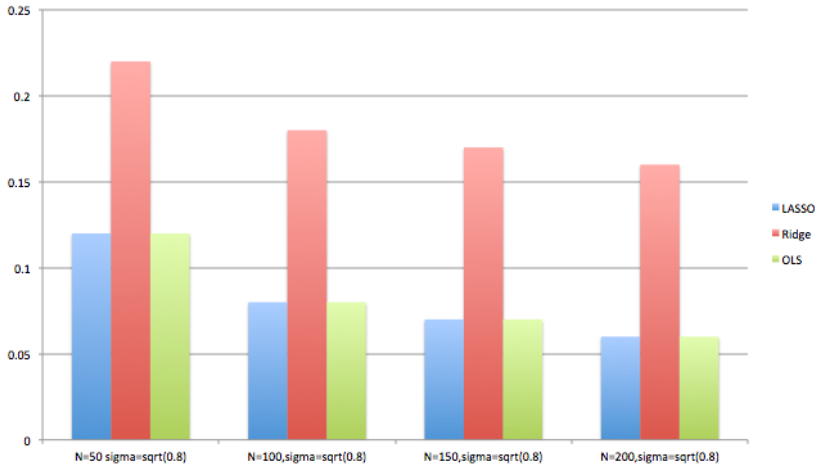

Figure 16: Estimation error sigma=sqrt(0.8) with different values of N

From figure 16, when $\sigma = \sqrt{0.8}$, Ridge regression still has the biggest estimated error across all different N. But other two methods will provide the same estimated error across all values of N

Estimated Error	N=50 sigma=sqrt(0.5)	N=100, sigma=sqrt(0.5)	N=150, sigma=sqrt(0.5)	N=200, sigma=sqrt(0.5)
LASSO	0.08	0.05	0.04	0.04
Ridge	0.2	0.17	0.16	0.16
OLS	0.08	0.05	0.04	0.04

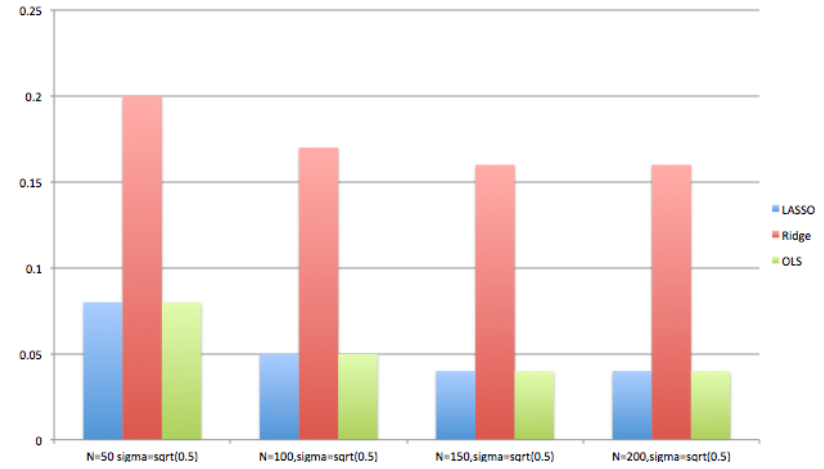

Figure 17: Estimation error sigma=sqrt(0.5) with different values of N

From figure 17, when $\sigma = \sqrt{0.5}$, Ridge regression still has the biggest estimation error across all different values of N. But other two methods will provide the same estimation error.

Estimated Error	N=50 sigma=sqrt(0.1)	N=100, sigma=sqrt(0.1)	N=150, sigma=sqrt(0.1)	N=200, sigma=sqrt(0.1)
LASSO	0.09	0.08	0.08	0.08
Ridge	0.19	0.16	0.16	0.15
OLS	0.02	0.01	0.01	0.01

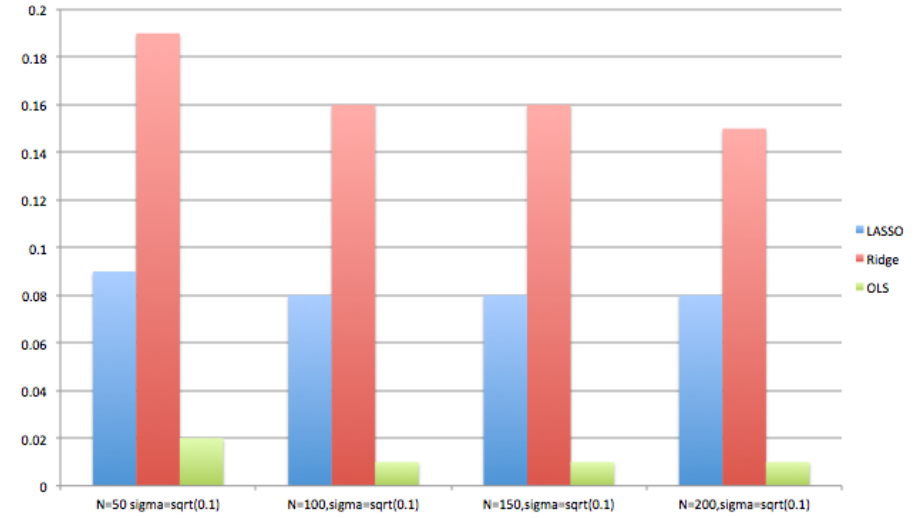

Figure 18: Estimation error sigma=sqrt(0.1) with different values of N

From figure 18, when $\sigma = \sqrt{0.1}$, Ridge regression still has the biggest estimation error across all different values of N. But OLS regression is way better than the LASSO regression in terms of estimation error.

PredictionErr	N=50, sgima=1	N=50, sigma=sqrt(0.8)	N=50, sigma=sqrt(0.5)	N=50, sigma=sqrt(0.1)
LASSO	0.98	0.79	0.49	0.27
Ridge	1.11	0.95	0.73	0.56
OLS	0.88	0.71	0.44	0.09

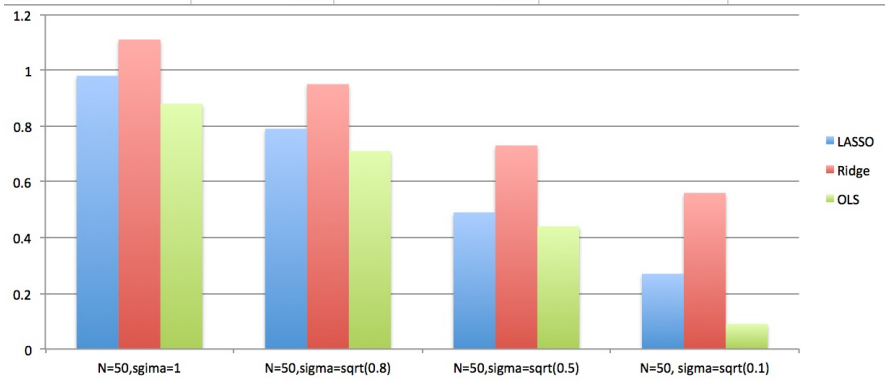

Figure 19: Prediction err N=50 with different values of sigma

From figure 19, when $N = 50$, Ridge regression has the prediction error across all different values of σ. But OLS regression provides smaller prediction error than LASSO regression does.

PredictionErr	N=100, sgima=1	N=100, sigma=sqrt(0.8)	N=100, sigma=sqrt(0.5)	N=100, sigma=sqrt(0.1)
LASSO	1.01	0.82	0.51	0.27
Ridge	1.12	0.95	0.71	0.51
OLS	0.94	0.76	0.48	0.09

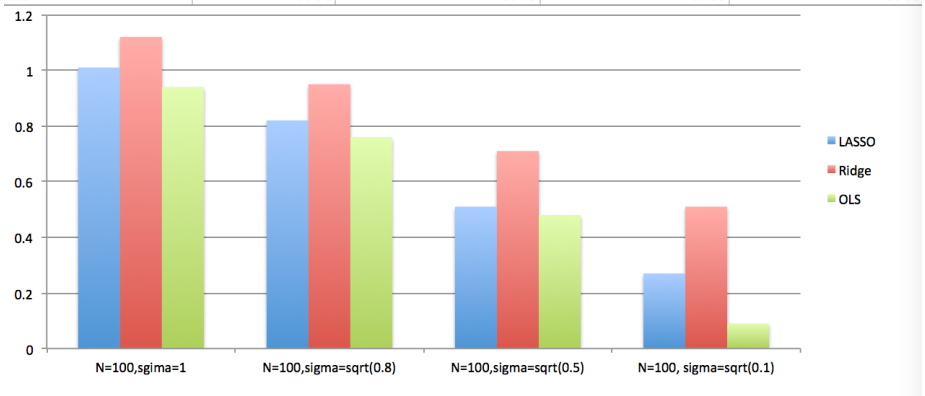

Figure 20: Prediction err N=100 with different values of sigma

From figure 20, when $N = 100$, Ridge regression still has the biggest prediction error across all different values of σ. But OLS regression provides smaller prediction error than LASSO regression does.

PredictionErr	N=150, sgima=1	N=150, sigma=sqrt(0.8)	N=150, sigma=sqrt(0.5)	N=150, sigma=sqrt(0.1)
LASSO	1.02	0.82	0.51	0.26
Ridge	1.12	0.94	0.7	0.49
OLS	0.97	0.77	0.48	0.1

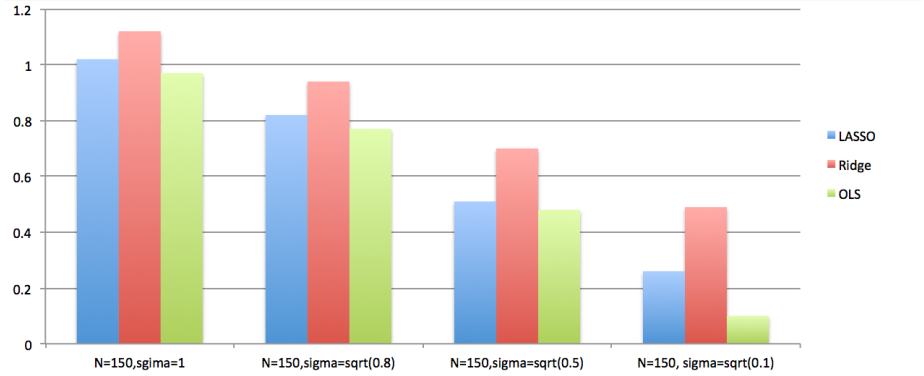

Figure 21: Prediction err N=150 with different values of sigma

From figure 21, when $N = 150$, Ridge regression still has the biggest prediction error across all different values of σ. But OLS regression provides smaller prediction error than LASSO regression does, especially when $\sigma = \sqrt{0.1}$.

PredictionErr	N=200, sgima=1	N=200, sigma=sqrt(0.8)	N=200, sigma=sqrt(0.5)	N=200, sigma=sqrt(0.1)
LASSO	1.02	0.82	0.51	0.26
Ridge	1.12	0.94	0.7	0.48
OLS	0.97	0.78	0.49	0.1

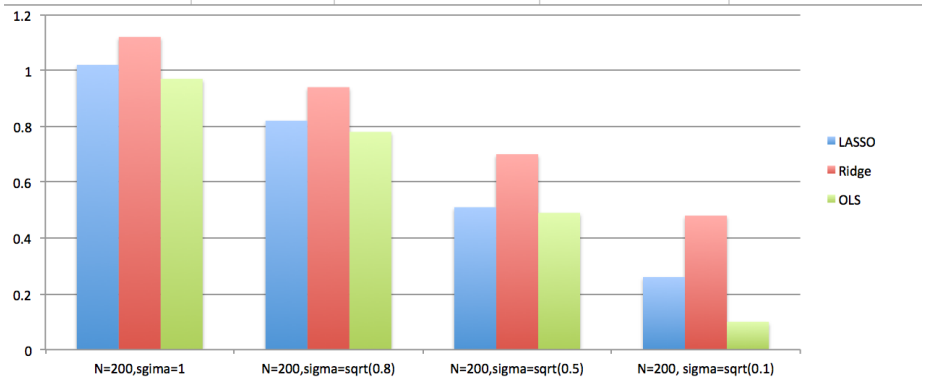

Figure 22: Prediction err N=200 with different values of sigma

From figure 22, when $N = 200$, Ridge regression still has the biggest prediction error across all different values of σ. But OLS regression provides smaller prediction error than LASSO regression does. When $\sigma = \sqrt{0.1}$, OLS regression's prediction is way accurate than other two methods..

PredictionErr	N=50, sgima=1	N=100, sigma=1	N=150, sigma=1	N=200, sigma=1
LASSO	0.98	1.01	1.02	1.02
Ridge	1.11	1.12	1.12	1.12
OLS	0.88	0.94	0.97	0.97

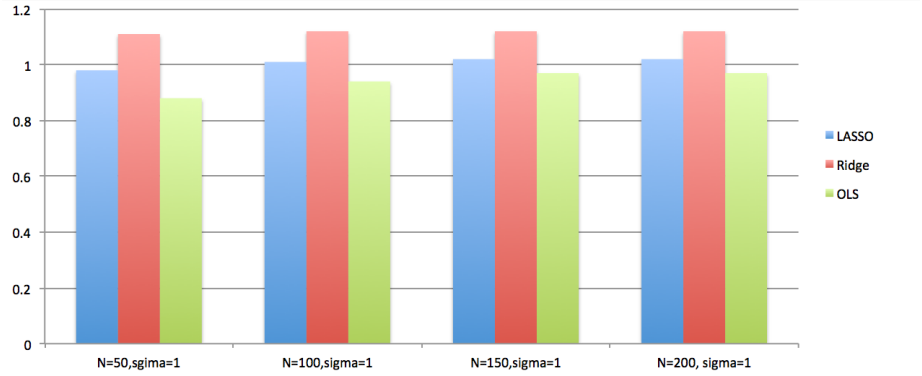

Figure 23: Prediction err sigma=1 with different values of N

From figure 23, when $\sigma = 1$, Ridge regression still has the biggest prediction error across all different values of N. But OLS regression provides smaller prediction error than LASSO regression across all different values of N.

PredictionErr	N=50, sgima=sqrt(0.8)	N=100, sigma=sqrt(0.8)	N=150, sigma=sqrt(0.8)	N=200, sigma=sqrt(0.8)
LASSO	0.79	0.82	0.82	0.82
Ridge	0.95	0.95	0.94	0.94
OLS	0.71	0.76	0.77	0.78

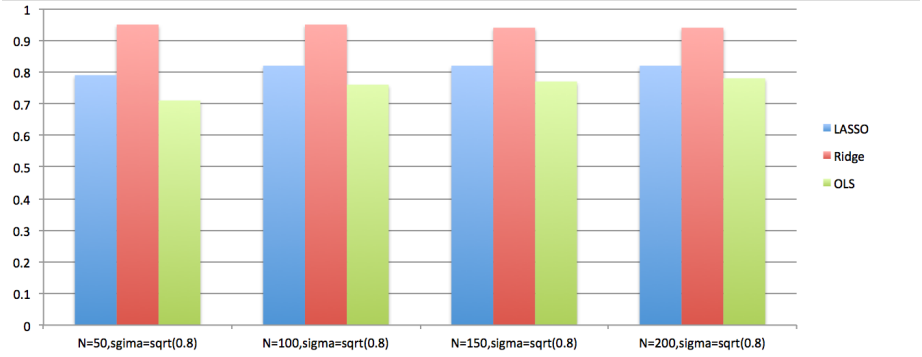

Figure 24: Prediction err sigma=sqrt(0.8) with different values of N

From figure 24, when $\sigma = \sqrt{0.8}$, rRdge regression still has the biggest prediction error across all different values of N. But OLS regression provides smaller prediction error than LASSO regression across all different values of N.

PredictionErr	N=50, sgima=sqrt(0.5)	N=100, sigma=sqrt(0.5)	N=150, sigma=sqrt(0.5)	N=200, sigma=sqrt(0.5)
LASSO	0.49	0.51	0.51	0.51
Ridge	0.73	0.71	0.7	0.7
OLS	0.44	0.48	0.48	0.49

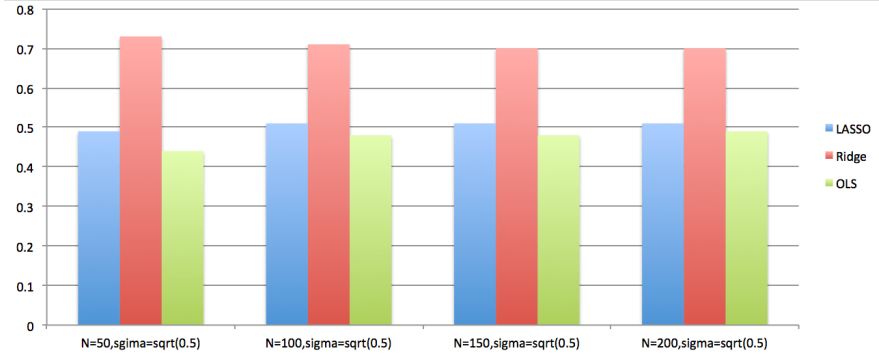

Figure 25: Prediction err sigma=sqrt(0.5) with different values N

From figure 25, when $\sigma = \sqrt{0.5}$, Ridge regression still has the biggest prediction error across all different values of N. But OLS regression provides slightly smaller prediction error than LASSO regression does across all different values of N.

PredictionErr	N=50, sgima=sqrt(0.1)	N=100, sigma=sqrt(0.1)	N=150, sigma=sqrt(0.1)	N=200, sigma=sqrt(0.1)
LASSO	0.27	0.27	0.26	0.26
Ridge	0.56	0.51	0.49	0.48
OLS	0.09	0.09	0.1	0.1

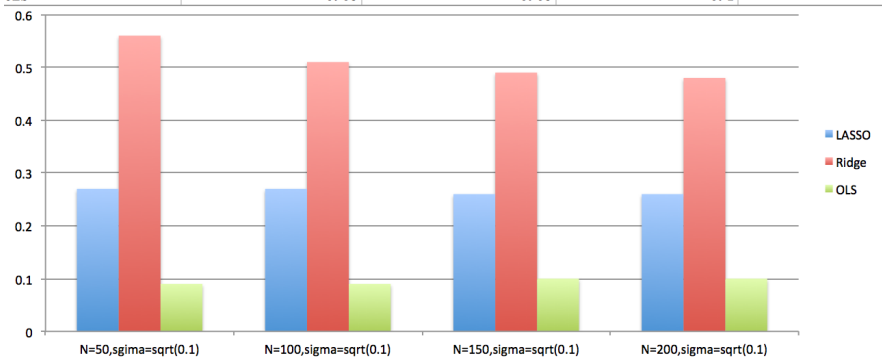

Figure 26: Prediction Err sigma=sqrt(0.1) with different N

From figure 26, when $\sigma = \sqrt{0.1}$, ridge regression still has the biggest estimated error across all different values of N. But OLS regression provides much smaller prediction error than LASSO regression does across all different values of N.

Conclusion

In this study, we do the simulation study for different combinations of $N = 50, 100, 150, 200$ and $\sigma = 1$, $\sqrt{0.8}$, $\sqrt{0.5}$, $\sqrt{0.1}$. We find that Ridge regression always gives the biggest prediction error and estimation error in our study. I think this is reasonable, since the original motivation of introducing Ridge regression is to introduce a little bias to make smaller variance in a very complex model. Our model here is very simple with only 10 parameters. I think for $p = 100$ or200, Ridge regression will be much better than OLS.

For prediction error, in most cases, OLS is better than the other two methods. This is also reasonable, since the model in this study is just a very simple model with a very dense true parameter. I think if the true parameter is very sparse, the LASSO will provide the better prediction than OLS. It is hard to compare LASSO with Ridge since it is possible that the very complicated model might contain a very sparse true parameter β.

For estimation error, OLS have done a better job when σ is very small. But in other cases, LASSO will provide almost the same estimated error as OLS does in our study.

Thus, in sum, it is hard to make a final conclusion which method is the best. The best way to choose method depends on the complexity of model, the purpose of your study and the structure of the true parameter β

Appendix 1

R code:

```
ddirmult <- function(x, alpha, log = FALSE) {
# check x is count data
if (any(x ¡ 0))
stop("Error: x should be nonnegative count data")
# check positivity of alpha
if (any(alpha ¡ 0))
stop("Error: Dirichlet-multinomial parameter alpha should be nonnegative")
# check dimensions of input arguments
if (is.vector(alpha) && length(alpha) ¿ 1) {
if (is.vector(x) && length(x) ¿ 1) {
if (length(x) != length(alpha)) {
stop("Error: sizes of x and alpha do not match.")
} else {
# expand x to a matrix of matching size with alpha
x <- matrix(x, 1, length(x))
}
} else if (is.vector(x) && length(x) <= 1) {
stop("Error: x can not be a scalar")
}
# expand alpha to a matrix of matching size with x
alpha <- matrix(alpha, nrow = nrow(x), ncol = length(alpha), byrow =
TRUE)
}
if (any(dim(alpha) != dim(x)))
stop("Error: dimensions of alpha and x do not match")

# compute log-densities
alphaRowSums <- rowSums(alpha)
xRowSums <- rowSums(x)
# lgamma(0 + 0) - lgamma(0) will produce NaNs.
# This fixes x_ij = alpha_ij = 0 cases.
# Is there better way to deal with this?
alpha[(x == 0) & (alpha == 0)] <- 1
# assemble log-likelihood
# lgamma(0) throws a lot of warnings but they are valid
logl <- suppressWarnings(
lfactorial(xRowSums) + rowSums(lgamma(x + alpha)) +
lgamma(alphaRowSums) - (rowSums(lfactorial(x)) +
rowSums(lgamma(alpha)) + lgamma(alphaRowSums + xRowSums))
)
# Deal with alphaRowSums == 0 cases
# fix the lgamma(0 + 0) - lgamma(0) produces NaN here
logl[(xRowSums == 0) & (alphaRowSums == 0)] <- 0
```

```
logl[(xRowSums > 0) & (alphaRowSums == 0)] < − - Inf

# output
if (log)
return(logl)
else
return(exp(logl))
}
dirmultfit < − function(X, weights = NULL, alpha0 = NULL,
tolfun = 1e-6, maxiters = 100, display = FALSE) {
# remove data points with batch size 0
rsum < − rowSums(X)
if (any(rsum == 0)) {
rmv < − sum(rsum == 0)
message(paste( "Warning: ", rmv,
" rows are removed because the row sums are 0"))
}
# remove the bins with no observations
csum < − colSums(X)
if (any(csum == 0)) {
rmv < − sum(csum == 0)
message(paste("Warning: ", rmv,
" columns are removed because the column sums are 0"))
}
# cleaned up data
data < − X[rsum != 0, csum != 0]
N < − nrow(data) ## Sample size
d < − ncol(data) ## Number of parameters
m < − rowSums(data) ## batch sizes
# set starting points
if (is.null(alpha0)) {
# method of moment estimate
rho < − sum(colSums((data / m)^2) / (colSums(data / m)))
alpha0 < − as.vector(colSums(data / m) * (d - rho) / (rho - 1) / N)
alpha0[alpha0 ¡= 0] < − 1e-6
} else {
alpha0 < − alpha0[csum != 0]
if (!is.vector(alpha0) && length(alpha0) != d) {
stop("Error: dimension of alpha0 does not match data")
} else if (any(alpha0 ¡= 0)) {
# user provided starting values
stop("Error: starting values should be positive")
}
}
# prepare some variables before looping
alphaHat < − alpha0
```

```
alphaSum < − sum(alphaHat)
loglIter < − sum(ddirmult(data, alpha0, log = TRUE))
# backtrack max iterations
backtrackMaxiters < − 10

## The Newton loop
if (maxiters == 1) iter < − 1
else {
for (iter in 2:maxiters) {

# score vector
alphaMat < − matrix(alphaHat, nrow = N, ncol = d, byrow = TRUE)
score < − colSums(digamma(data + alphaMat)) -
N * (digamma(alphaHat) - digamma(alphaSum)) -
sum(digamma(alphaSum + m))
# observed info. matrix = diag(obsinfoDvec) - obsinfoC
obsinfoDvec < − N * trigamma(alphaHat) -
colSums(trigamma(data + alphaMat))
obsinfoDvecInv < − 1 / obsinfoDvec
obsinfoC < − N * trigamma(alphaSum) -
sum(trigamma(m + alphaSum))
# shrink c if necessary to make obs. info. pos def
if (obsinfoC * sum(obsinfoDvecInv) >= 1) {
obsinfoC < − 0.95 / sum(obsinfoDvecInv)
}
# compute Newton direction
newtondir < − obsinfoDvecInv * score
newtondir < − newtondir +
(sum(newtondir) / (1 / obsinfoC - sum(obsinfoDvecInv))) * obsinfoDvecInv
# line search by step halving
if (any(newtondir < 0)) {
# make sure Newton iterate always lands within boundary
stepsize < − min(- alphaHat[newtondir < 0] / newtondir[newtondir < 0])
stepsize < − min(0.95 * stepsize, 1)
} else {
stepsize < − 1
}
for (btiter in 1:backtrackMaxiters) {
alphaNew < − alphaHat + stepsize * newtondir
loglNew < − sum(ddirmult(data, alphaNew, log = TRUE))
# line search successful if improving log-L
if (loglNew > loglIter) break
else if (btiter == backtrackMaxiters) {
warning("line search failed")
} else {
stepsize < − stepsize / 2
```

```
}
}
alphaHat < − alphaNew
alphaSum < − sum(alphaHat)
loglOld < − loglIter
loglIter < − loglNew
# check convergence criterion
if (abs(loglIter - loglOld) < tolfun * (abs(loglOld) + 1)) break
}
}

# score, i.e., gradient
alphaMat < − matrix(alphaHat, nrow = N, ncol = d, byrow = TRUE)
score < −
colSums(digamma(data + alphaMat)) -
N * (digamma(alphaHat) - digamma(alphaSum)) -
sum(digamma(alphaSum + m))
# restore to original data size
if (any(csum == 0)) {
colidx < − (csum != 0)
# parameter estimate
tmp < − alphaHat
alphaHat < − rep(0, ncol(X))
alphaHat[colidx] < − tmp
}
# output
return(alphaHat)
}
MC_MLE < − function(I, d, S , N = 20) {
this is a function to calculate estimator for P using MLE method
# Args: I: number of populations
# d: number of categories
# S: number of replicates
# N: batch size
# Output Err_MLE: estimated error for MLE estimator
#
set.seed(100)
Err_MLE < − matrix(0, S)
temp < − matrix(0, I, 1)
for ( s in 1:S) {
p < − matrix(runif(d * I), I, d)
temp < − rowSums(p)
p < − p / temp
data < − rmultinomial(I, N, p)
pr_MLE < − data / N
Err_MLE[s] < − sum(abs(pr_MLE - p)) / (2 * I)
```

```
}
return(Err_MLE)
}

MC_EB <- function(I, d, S, N = 20) {
#
# This is a function to calculate estimator for P using EB method
# Args: I: number of populations
# d: number of categories
# S: number of replicates
# N: batch size
# Output Err_EB: estimated error for EB estimator
#
set.seed(100)
# initialize matrix
data <- matrix(0, d, I)
temp1 <- matrix(0, I, 1)
temp <- matrix(0, S, 1)
# loop over different replicate
for(s in 1:S) {
p <- matrix(runif(d * I), I, d)
temp1 <- rowSums(p)
p <- p / temp1
data <- rmultinomial(I, N, p)
# Newton method to estimate alpha
alpha <- dirmultfit(data)
pr_EB <- t((data + alpha) / (N + sum(t(alpha))))
temp[s] <- sum(abs(pr_EB - t(p)))/(2 * I)
}
Err_EB <- temp
return(Err_EB)
}

MLE <- function(S, I, d) {
# This is a function to calculate estimator error for different I and d
# Args: I: number of populations (vector)
# d: number of categories (vector)
# S: number of replicates
# Output Err_MLE: estimated error for MLE estimator for different replicate
#
tabledata_MLE <- matrix(0, S, length(I) * length(d))
k = 1
for (i in 1 : length(I)){
for (j in 1 : length(d)) {
tabledata_MLE[, k] <- MC_MLE(I[i], d[j], S)
k <- k + 1
```

```
}
}
return(tabledata_MLE)
}

EB <- function(S, I, d){
# This is a function to calculate estimator error for different I and d
# Args: I: number of populations (vector)
# d: number of categories (vector)
# S: number of replicates
# Output Err_EB: estimated error for EB estimator for different replicate
#
tabledata_EB <- matrix(0, S, length(I)*length(d))
k = 1
for (i in 1:length(I)){
for (j in 1:length(d)) {
tabledata_EB[, k]<- MC_EB(I[i], d[j], S)
k <- k + 1
}
}
return(tabledata_EB)
}
library(MASS)
library(mc2d)
library(compiler)
library(knitr)
# start simulation
I <- c(10, 20, 50)
d <- c(2, 3, 5, 10)
tabledata_MLE <- MLE(S, I, d)
tabledata_EB <- EB(S, I, d)
S<-100

# Compute stardard error of estimated error
SE_MLE <- matrix(apply(tabledata_MLE, 2, var), 4, 3)/S
SE_EB <- matrix(apply(tabledata_EB, 2, var), 4, 3)/S
SE_MLE <- round(sqrt(SE_MLE), 3)
SE_EB <- round(sqrt(SE_EB), 3)
# Genearate table
rownames(SE_EB) <- c("d = 2, SE", "d = 3, SE", "d = 5, SE", "d = 10,
SE")
rownames(SE_MLE) <- c("d=2, SE", "d=3, SE", "d = 5, SE", "d = 10, SE")
row.names(SE_EB) <- paste0(row.names(SE_EB), "(EB)")
row.names(SE_MLE) <- paste0(row.names(SE_MLE), "(MLE)")
table_Mean_MLE <- colMeans(tabledata_MLE)
table_Mean_MLE <- round(matrix(table_Mean_MLE, 4, 3), 2)
```

```
colnames(table_Mean_MLE) < − c("I=10", "I=20", "I=50")
rownames(table_Mean_MLE) < − c("d=2", "d=3", "d=5", "d=10")
table_Mean_EB < − colMeans(tabledata_EB)
table_Mean_EB < − round(matrix(table_Mean_EB, 4, 3), 2)
colnames(table_Mean_EB) < − c("I=10", "I=20", "I=50")
rownames(table_Mean_EB) < − c("d=2", "d=3", "d=5", "d=10")
row.names(table_Mean_EB) < − paste0(row.names(table_Mean_EB), "(EB)")
row.names(table_Mean_MLE) < − paste0(row.names(table_Mean_MLE), "(MLE)")
tableNew < − rbind(table_Mean_EB, SE_EB, table_Mean_MLE  SE_MLE)
tableNew1 < − tableNew[c(matrix(1:nrow(tableNew), nrow=4, byrow=TRUE)),
]
library(knitr)
kable(tableNew1, format='pandoc')
```

Appendix 2

```
# Generating data
genData <- function(n ,beta, sigma) {
# This is a function to generate data with size n, beta and noise level of
# sigma
# Args: n : data size
# beta: true parameter
# sigma: noise level
# Outputs: list contains X and Y
p <- length(beta)
X <- matrix(rnorm(n * p), n, p)
X[, 1] <- rep(1, n)
y <- rnorm(n, X %*% beta, sigma ^ 2)
return(list(X = X, y = y))
}

ridge <- function(data, beta) {
# This is a function to find beta using ridge regression
# Args: data: X and Y
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
require(glmnet)
cvfit <- cv.glmnet(data$X, data$y, family = "gaussian", alpha = 0)
n <- length(data$y)
#BIC <- n*log(residual) + log(n) * fit$df
coeff <- as.numeric(coef(cvfit,s = cvfit$lambda.min))[-2]
residual <- predict(cvfit, newx = data$X) - data$y
prederr <- sqrt(crossprod(residual) / n)
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err"=sqrt(sum(coeff-
beta)^2) / length(beta)) )
}

lasso <- function(data, beta) {
# This is a function to find beta using LASSO regression
# Args: data: X and Y
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
require(glmnet)
cvfit <- cv.glmnet(data$X, data$y, family = "gaussian", alpha = 1)
n <- length(data$y)
coeff <- as.numeric(coef(cvfit, s = cvfit$lambda.min))[-2]
residual <- predict(cvfit, newx = data$X) - data$y
```

```
prederr <- sqrt(crossprod(residual) / n)
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err" = sqrt(sum((coeff
- beta) ^ 2) / length(beta)) ))
}

OLS <- function(data, beta) {
# This is a function to find beta using OLS regression
# Args: data: X and Y
# beta: true parameter
# Outputs: list contains estimated coefficients
# prediction error and estimation error
#
fit <- lm(y~X, data)
coeff <- coef(fit)[-2]
prederr <- sqrt(crossprod(fit$residuals) / length(data$y))
return(list("coefficient" = coeff, "PredictionErr" = prederr, "Err"=sqrt(sum((coeff
- beta)^2) / length(beta))))
}

set.seed(100)
beta <- runif(10, -5, 5)
simulation1 <- function(n, sigma, S, beta, i, j) {
# This is the function do simulation across different combination of
# sigma and n give specific value of S and beta
# Args: n : data size
# sigma: noise level
# S : number of replicates
# beta : true value
# i, j : index to set seed
# Outputs : list contains the estimation
# Err matrix for different method and replicates,
# the prediction err matrix for different method and replicates,
# true beta
p <- length(beta)
results <- array(NA, dim = c(S, p, 3),
dimnames = list(1 : S, 1 : p, c("Lasso", "Ridge", "OLS")))
predictionErr <- Err <- matrix(NA, S, 3)
for (s in 1:S)
{
set.seed((i - 1) * i + (j - 1) * j + s * (s - 1))
# generate data
Data <- genData(n, beta, sigma)
# fit model using three methods
lassfit <- lasso(Data, beta)
ridgefit <- ridge(Data, beta)
OLSfit <- OLS(Data, beta)
```

```
# store results
results[s, , 1] <- lassfit$coefficient
results[s, , 2] <- ridgefit$coefficient
results[s, , 3] <- OLSfit$coefficient
Err[s, 1] <- lassfit$Err
Err[s, 2] <- ridgefit$Err
Err[s, 3] <- OLSfit$Err
predictionErr[s, 1] <- lassfit$PredictionErr
predictionErr[s, 2] <- ridgefit$PredictionErr
predictionErr[s, 3] <- OLSfit$PredictionErr
}
return(list("Err"= Err, "PredictionErr" = predictionErr, "estiamte" = results,
"true" = beta)) # return(Err_s vector) }
##################################################################
S <- 700 # S=700 can make error and prediction error significant
# different settings we want in our study
sigma <- c(sqrt(1), sqrt(0.8), sqrt(0.5), sqrt(0.1))
N <- c(50, 100, 150, 200)
k <- 1
lasso_Err <- matrix(0, S, length(N) * length(sigma))
ridge_Err <- lasso_Err
OLS_Err <- lasso_Err
lasso_pdErr <- matrix(0, S, length(N) * length(sigma))
ridge_pdErr <- lasso_Err
OLS_pdErr <- lasso_Err

for ( sig in 1:length(sigma)){
for (Num in 1:length(N)) {
sim <- simulation1(N[Num], sigma[sig], S, beta, sig, Num)
lasso_Err[, k] <- sim$Err[, 1]
ridge_Err[, k] <- sim$Err[, 2]
OLS_Err[, k] <- sim$Err[, 3]
lasso_pdErr[, k] <- sim$PredictionErr[, 1]
ridge_pdErr[, k] <- sim$PredictionErr[, 2]
OLS_pdErr[, k] <- sim$PredictionErr[, 3]
k <- k + 1
}
}

table1_se_lassoErr <- matrix(apply(lasso_Err, 2, sd) / sqrt(S), 4, 4)
table2_se_ridgeErr <- matrix(apply(ridge_Err, 2, sd) / sqrt(S), 4, 4)
table3_se_OLSErr <- matrix(apply(OLS_Err, 2, sd) / sqrt(S), 4, 4)
table4_se_lassopderr <- matrix(apply(lasso_pdErr,2,sd) / sqrt(S), 4, 4)
table5_se_ridgepderr <- matrix(apply(ridge_pdErr,2,sd) / sqrt(S), 4, 4)
table6_se_OLSpderr <- matrix(apply(OLS_pdErr,2,sd) / sqrt(S), 4, 4)
```

```
table1_lassoErr <- matrix(colMeans(lasso_Err), 4, 4)
table2_ridgeErr <- matrix(colMeans(ridge_Err), 4, 4)
table3_OLSErr <- matrix(colMeans(OLS_Err), 4, 4)
table4_lassopderr <- matrix(colMeans(lasso_pdErr), 4, 4)
table5_ridgepderr <- matrix(colMeans(ridge_pdErr), 4, 4)
table6_OLSpderr <- matrix(colMeans(OLS_pdErr), 4, 4)
```

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(LASSO)(SE of err)	0.0015718927	0.0011420875	0.0007439472	0.0004005936
N=100(LASSO)(SE of err)	0.0009355473	0.0007448055	0.0004638757	0.0002645828
N=150(LASSO)(SE of err)	0.0007290767	0.0005804646	0.0004004586	0.0002000516
N=200(LASSO)(SE of err)	0.0006253207	0.0005143224	0.0003141784	0.0001799039

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(ridge)(SE of err)	0.002176625	0.0019217678	0.0016126640	0.0013741364
N=100(ridge)(SE of err)	0.001401327	0.0011716404	0.0009607174	0.0008124437
N=150(ridge)(SE of err)	0.001112252	0.0009558223	0.0007731985	0.0005939201
N=200(ridge)(SE of err)	0.000957661	0.0008563401	0.0006327612	0.0005431187

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(OLS)(SE of err)	0.0015294295	0.0011358628	0.0007589443	1.510234e-04
N=100(OLS)(SE of err)	0.0009172869	0.0007482264	0.0004603656	9.394758e-05
N=150(OLS)(SE of err)	0.0007072984	0.0005761593	0.0003871185	7.424674e-05
N=200(OLS)(SE of err)	0.0006126488	0.0004996728	0.0002869649	6.275764e-05

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(LASSO)(SE of pderr)	0.004379188	0.003400165	0.0022307743	0.0011310544
N=100(LASSO)(SE of pderr)	0.003049849	0.002385665	0.0015456312	0.0008337341
N=150(LASSO)(SE of pderr)	0.002472421	0.001854733	0.0011760520	0.0006624303
N=200(LASSO)(SE of pderr)	0.002087547	0.001563400	0.0009915576	0.0005621467

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(ridge)(SE of pderr)	0.003938123	0.003171778	0.002500618	0.002588270
N=100(ridge)(SE of pderr)	0.002876214	0.002216064	0.001755613	0.001942893
N=150(ridge)(SE of pderr)	0.002330437	0.001833359	0.001422733	0.001555088
N=200(ridge)(SE of pderr)	0.001976515	0.001575544	0.001284435	0.001282722

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(OLS)(SE of pderr)	0.003760394	0.002997457	0.0019320598	0.0003659929
N=100(OLS)(SE of pderr)	0.002725846	0.002166862	0.0014154806	0.0002742102
N=150(OLS)(SE of pderr)	0.002239490	0.001718111	0.0010883494	0.0002184989
N=200(OLS)(SE of pderr)	0.001900215	0.001456012	0.0009225482	0.0001873213

Figure 27: Output: standard error for prediction error and estimation error

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(LASSO)(Err)	0.15	0.12	0.08	0.09
N=100(LASSO)(Err)	0.10	0.08	0.05	0.08
N=150(LASSO)(Err)	0.08	0.07	0.04	0.08
N=200(LASSO)(Err)	0.07	0.06	0.04	0.08

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(ridge)(Err)	0.24	0.22	0.20	0.19
N=100(ridge)(Err)	0.19	0.18	0.17	0.16
N=150(ridge)(Err)	0.17	0.17	0.16	0.16
N=200(ridge)(Err)	0.17	0.16	0.16	0.15

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(OLS)(Err)	0.15	0.12	0.08	0.02
N=100(OLS)(Err)	0.10	0.08	0.05	0.01
N=150(OLS)(Err)	0.08	0.07	0.04	0.01
N=200(OLS)(Err)	0.07	0.06	0.04	0.01

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(LASSO)(pderr)	0.98	0.79	0.49	0.27
N=100(LASSO)(pderr)	1.01	0.82	0.51	0.27
N=150(LASSO)(pderr)	1.02	0.82	0.51	0.26
N=200(LASSO)(pderr)	1.02	0.82	0.51	0.26

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(ridge)(pderr)	1.11	0.95	0.73	0.56
N=100(ridge)(pderr)	1.12	0.95	0.71	0.51
N=150(ridge)(pderr)	1.12	0.94	0.70	0.49
N=200(ridge)(pderr)	1.12	0.94	0.70	0.48

	sigma=1	sigma=sqrt(0.8)	sigma=sqrt(0.5)	sigma=sqrt(0.1)
N=50(OLS)(pderr)	0.88	0.71	0.44	0.09
N=100(OLS)(pderr)	0.94	0.76	0.48	0.09
N=150(OLS)(pderr)	0.97	0.77	0.48	0.10
N=200(OLS)(pderr)	0.97	0.78	0.49	0.10

Figure 28: Output: prediction error and estimation error