STAT-S 610 Final Project

BJKill

12/3/2020

When simulating our data for our linear model, we need to know

1. How many observations or data points we have, n, such that $n \in \mathbb{N}$
2. How many predictor variables we have, p, such that $p \in \mathbb{N}$, $1 \leq p \leq n-2$
3. How many of those predictor variables are the "good" ones, k, such that $k \in \mathbb{N}$, $1 \leq k \leq n$
4. How many times we generate the data and run backwards elimination on it, m, such that $m \in \mathbb{N}$
5. The significance level we will be using, α, such that $\alpha \in (0, 1)$

Let's say we have

```r
n <- 100        # observations
p <- 30         # predictor vars
k <- 15         # valid predictor vars
m <- 125        # simulations
alpha <- 0.05   # sig level
```

The first thing we must do is to generate the data.

```r
make_model_matrix <- function(n, p) {
    if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {
        return("n and p must be positive integers")
    }
    if (n < p + 2) {
        return("n must be at least p+2")
    } else {
        X <- matrix(nrow = n, ncol = p)
        for (i in 1:p) {
            X[, i] <- rnorm(n)
        }
        return(X)
    }
}
```

Here is what it gives us:

```r
X_mat <- make_model_matrix(n,p)
dim(X_mat)
```

```
## [1] 100  30
```

```r
head(X_mat)
```

```
##            [,1]        [,2]        [,3]       [,4]        [,5]       [,6]       [,7]       [,8]        [,9]      [,10]
## [1,]  0.1747586 -0.05254192  0.47830613 -1.2310275 -1.54278893 -0.9524904 -1.7969004  1.3295294 -2.38292903 -1.1048912
## [2,]  0.2365746 -0.46942066 -0.54783276 -1.6273152 -2.10839868  2.4441648  0.1673256  0.6381959 -0.80536704  1.5395851
## [3,]  0.4325620 -0.10327028 -1.28806698  0.4813901 -1.37370418  0.2722537  0.2088436 -0.8913172  0.08918844 -0.2441417
## [4,] -0.2323792 -0.10931170 -0.09311896  0.5767637 -0.03291628  0.7378521  0.7673974  1.3371961  0.30797753  1.6822135
## [5,] -0.2364654  1.46978617 -0.98173027  1.1266691 -0.79515595 -0.6783235 -0.1983495 -1.2080685 -0.31562239  0.2400831
## [6,] -0.1136910  1.57666031  0.52488692 -0.4526341  0.02778976 -0.4598396  1.4142117  0.8469955 -1.30294409  1.9786457
##             [,11]       [,12]       [,13]      [,14]      [,15]      [,16]      [,17]      [,18]      [,19]       [,20]
## [1,] -0.514164037 -1.38348609 -0.01383100  0.2091635 -1.0850205 -1.2424000  0.1994239 -0.4283364 -0.3427272 -0.02608416
## [2,]  1.023726864 -0.06599645 -1.60416221  0.4961675  0.3948371  0.9871316 -1.5098757 -0.8820266 -0.3432497 -0.19441284
## [3,] -0.723658866 -0.58025852 -1.45557447  0.7127541  0.2019290 -0.3979097  0.1097398 -1.1409616  1.1979760 -0.59746618
## [4,] -0.904602494  1.75606079 -0.22673291  1.0828511  0.4979133  0.3657793 -0.6465394  0.7360509 -2.0218968  0.57620179
## [5,]  0.007071082 -0.74639498  0.82170357 -0.5925242 -1.2140874 -1.0883782  1.7582422 -0.2723028 -0.9843749 -3.55475964
## [6,] -0.038530609 -0.89399410  0.05729878  0.6217276  0.3975097  1.4493062 -0.1785410 -0.1089013  0.3370182  0.72945468
##            [,21]      [,22]      [,23]      [,24]       [,25]       [,26]       [,27]       [,28]       [,29]
## [1,]  0.51107626 -0.5418214 -0.6136873 -1.5376819  0.892817672  1.11056281  0.13784296 -1.09798939 -0.05084035
```

```
## [2,]  0.58535294 -0.3635223  1.0433424  0.7623727  0.319415554 -0.40875476  0.93579841 -0.83418927 -0.86307524
## [3,] -0.89348762  0.3215639 -0.1978076 -1.3741644  0.693880571  1.40828587 -0.03967933  0.64680197 -0.73834547
## [4,]  0.77500433  0.6387293 -0.5658923  0.5749469 -1.297826479  0.03489504  0.24044938 -1.00887084 -0.95762371
## [5,] -2.18968497  1.0870134 -0.3995424  0.1767418 -0.283683338 -1.54195339 -0.19074854  0.56013932  0.48722747
## [6,] -0.07413536  1.6445188  0.6556672 -0.5618878 -0.004725164  0.89677999 -0.06349086  0.02067925 -0.93516181
##           [,30]
## [1,]  0.2330487
## [2,]  0.2305697
## [3,]  0.5100310
## [4,]  0.3522231
## [5,] -2.0095807
## [6,] -1.4208852
```

Then, we will use the first k predictor variables as the basis for generating our y values. For simplicity, we will not have an intercept, we will give each predictor variable the same coefficient as its index, and we will use a standard normal error term, like so:

$$Y \sim N(0,1) + \sum_{i=1}^{k} k * X_k$$

$$or,$$

$$Y \sim 1X_1 + 2X_2 + 3X_3 + \ldots + kX_k + \epsilon$$

Here is how we'll do it:

```
make_response_vector <- function(pred_mat, k) {
    if (k <= 0 | k > ncol(pred_mat) | is.wholenumber(k) == FALSE) {
        return("k must be a positive integer not exceeding p")
    }
    Y <- vector(length = nrow(pred_mat))
    for (i in 1:nrow(pred_mat)) {
        Y[i] <- rnorm(1)
        for (j in 1:k) {
            Y[i] <- Y[i] + pred_mat[i, j] * j
        }
    }
    return(Y)
}
```

Using our example `X_mat` from before, here is what we get for our response values.

```
Y_vec <- make_response_vector(X_mat,k)
length(Y_vec)
```

```
## [1] 100
```

```
Y_vec
```

```
##   [1]  -87.15757637   13.04571094  -35.12297631   72.70197602  -41.81243778   30.30450536  -22.23496808   32.45272545
##   [9]   30.74884058  -30.94225808  -15.17465385    7.65618498  -64.66499011   -3.12253338   28.02656225  -26.71177582
##  [17]   33.11520145   -5.47777006   49.70387600  -31.77859278    2.19806927    5.88640338   -9.29506934  -68.82464792
##  [25]   -0.09158119  -14.63303582  -64.17586670   64.41847263  -41.86421208   66.17278313   -1.39802783  -11.92637291
##  [33]   74.87077668   24.07607710  -72.62295988  -15.73828232  -13.03269682   53.57749750   18.51976346   83.76924027
##  [41]    7.50569763    2.41552383   17.50638777  -30.32831551  -37.87907865   32.73297235    0.26883649   31.54073884
##  [49]  -16.90655723  -13.76750416   96.10028136  -51.47360961  -10.04761692  -10.62908365   -4.54994601   43.15622819
##  [57]  -40.30980306  -43.17354094  -42.31577260  -32.02023539   37.00351996   32.06693979  -16.48263773   29.25395227
##  [65]   -1.81520270   10.31532079   38.02873013   20.57737554   59.98917634    1.29805857  -41.25532923   41.74306280
##  [73]  -45.62767601   25.40832614 -116.15926501   25.92126857   -4.58215401   -8.71021051   26.32766280   35.01627691
##  [81]  -17.19385770  -31.58284127   22.59792422  -26.15119374  -37.46165305    9.27410360   39.10838460  -25.18661922
##  [89]  -35.73513934  -21.79789588   -8.64401851  -29.69186023   39.89364322  -59.71368552  -17.19768812  -49.08587349
##  [97]  -35.95316759  -68.16073938  -21.89042945   23.20973348
```

We will then create a function that can generate the data and combine the response and the predictors into a single data frame in order for us to use R's built-in `lm` function.

```
make_data_frame <- function(n, p, k) {
    if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {
        return("n and p must be positive integers")
    }
    if (n < p + 2) {
        return("n must be at least p+2")
    }
    if (k <= 0 | k > p | is.wholenumber(k) == FALSE) {
        return("k must be a positive integer not exceeding p")
    } else {
        X <- make_model_matrix(n, p)
        Y <- make_response_vector(X, k)
        df <- data.frame(cbind(X, Y))
        return(df)
    }
}
```

Let's use this to create a new data frame and see what we get.

```
our_df <- make_data_frame(n,p,k)
head(our_df)
```

```
##           V1          V2         V3         V4         V5         V6         V7         V8          V9         V10
## 1 -1.2406155 -1.40715139 -0.3432231 -1.3675110 -2.3971812  0.9576927  0.3819475  0.7196053 -0.41838967  0.07445884
## 2 -0.4220567 -0.05303418 -0.7733127  0.8254116  1.5539868  1.0393288 -3.0602948 -0.4939182  0.34543357  0.85778009
## 3  0.2466436 -0.46477627  1.9498946 -0.9120776 -0.1360423  0.9788080 -0.5591876  0.1100531  0.09854595 -0.34858143
## 4  0.5839272  1.24225994 -0.7502044 -0.2282489  0.3615824 -1.1132163 -1.5027194  1.6413133 -1.59607665 -0.35785517
## 5 -0.6024883  0.40205382 -0.9664566  0.2965336 -0.6648975 -0.4143441  1.1402606  1.0581789 -0.34984908 -1.63933113
## 6  0.6558081 -2.46670449  0.9673687 -0.8286355  0.4197715 -1.6403726  0.3634385  0.3890005 -0.43789581 -0.62594111
##          V11        V12        V13        V14        V15         V16         V17        V18         V19        V20
## 1  0.7849373  0.8771865 -0.4710165 -1.2577753 -0.5051762  0.73316342  0.08494604 -0.8694604  0.66915633 -1.1300401
## 2 -1.1198962 -1.6790957 -0.9180590  0.0100869 -0.4640676 -0.09462316 -1.48187325  0.2130133 -0.16323018  0.2063944
## 3 -0.5125818  1.2044140 -0.4439713 -0.5964563 -0.6288333 -1.01992691 -0.80173821  0.7520616  2.31882695 -1.5060475
## 4  0.7437480 -1.0803456 -0.7091208 -0.4167284  0.5130059 -0.25986638 -0.10595880 -0.7006098  0.15653986  1.2775001
## 5 -0.7886760  1.1763541 -0.8004908  1.3507017  0.8982431  0.59753816 -0.62403105  0.1856587  0.01606657  0.1715003
## 6 -0.5454980  0.1896130 -0.9773442 -0.9826253  1.8873511  0.59142422 -1.46723425  1.8498005 -0.78014011 -1.0552998
##          V21        V22        V23        V24         V25        V26         V27         V28        V29        V30
## 1 -1.3681528  0.3643881  0.4879378 -0.6238744  0.33442073  1.0596087 -1.19667103 -1.09755832 -0.6928483  0.8535413
## 2 -0.7209215 -0.3205987  0.1549657 -0.0037715 -0.09109815  2.0748754 -0.75661541 -1.21364218  0.6395436 -0.4283085
## 3 -0.9328638 -0.5390973  0.0576816 -0.5202623  0.17306773  1.2006128  0.17075067  1.12127709 -0.4948487  1.3620662
## 4 -0.3463196  0.2878621 -1.1111626  0.8734812 -1.47093349 -0.1226029 -0.06969246  1.12950478  0.1437377 -0.2598692
## 5  0.6206598 -3.1065744 -0.3144966  2.4232515 -0.53179352  1.1328664  0.93084693 -0.79383800 -0.3336540 -0.9194541
## 6  0.3620863  0.8961537 -0.8863287  1.3962633  0.19235530  0.7273367 -0.92170148  0.03893721 -1.7325641  0.4930915
##           Y
## 1 -24.43545
## 2 -49.62960
## 3 -15.35291
## 4 -34.39467
## 5  16.14298
## 6 -17.87270
```

Now that we can generate a data frame just the way we like it, we can create a function that generates a data frame and systematically eliminates the least significant variable (highest p-value) from the linear model one at a time until all of the variables left have p-values that are at most our pre-determined significance level, α. It will return the coefficient matrix of the final linear model along with the $100(1-\alpha)\%$ CI for each parameter and an indicator of whether the CI for that parameter contained the known parameter.

```
run_BE <- function(n, p, k, alpha) {
    if (alpha < 0 | alpha > 1) {
        return("alpha must be in the interval (0,1)")
    }
    if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {
        return("n and p must be positive integers")
    }
    if (n < p + 2) {
        return("n must be at least p+2")
    }
    if (k <= 0 | k > p | is.wholenumber(k) == FALSE) {
        return("k must be a positive integer not exceeding p")
    } else {
        df <- make_data_frame(n, p, k)
        while (summary(lm(Y ~ ., df))$coefficients[1 + which.max(summary(lm(Y ~ ., df))$coefficients[-1, 4]), 4] > alpha) {
            rem_inx <- which.max(summary(lm(Y ~ ., df))$coefficients[-1, 4])
            df <- df[, -rem_inx]
        }
        lm1 <- lm(Y ~ ., df)
        display <- cbind(summary(lm1)$coefficients, confint(lm1))
        display <- cbind(display, vector(length = nrow(display)))
        colnames(display)[7] <- "Known Param in CI?"
        display[1, 7] <- (0 >= display[1, 5]) & (0 <= display[1, 6])
        for (i in 2:nrow(display)) {
            index <- as.numeric(str_sub(rownames(display)[i], 2, -1))
            display[i, 7] <- (index >= display[i, 5]) & (index <= display[i, 6])

        }
        return(display)
    }
}
```

To take a quick peek under the hood, let's create a data frame and see what the while loop is checking for.

```r
our_df2 <- make_data_frame(n, p, k)
our_lm <- lm(Y ~ ., our_df2)
summary(our_lm)$coefficients
```

```
##                Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  0.08369441  0.1150891   0.7272139 4.695552e-01
## V1           0.94893906  0.1126730   8.4220621 3.380375e-12
## V2           1.98189410  0.1186698  16.7009199 6.118775e-26
## V3           2.86394927  0.1199515  23.8758943 4.509846e-35
## V4           3.85433913  0.1220197  31.5878550 9.259210e-43
## V5           4.99895989  0.1086663  46.0028674 1.641043e-53
## V6           5.77910533  0.1478179  39.0961147 8.161683e-49
## V7           7.14495657  0.1123353  63.6038559 5.383261e-63
## V8           7.94134529  0.1031276  77.0050524 1.203678e-68
## V9           9.10651095  0.1222110  74.5146531 1.132706e-67
## V10          9.93700667  0.1285873  77.2782696 9.453138e-69
## V11         11.14951356  0.1177059  94.7234805 8.560375e-75
## V12         11.84439641  0.1311502  90.3117167 2.241036e-73
## V13         12.98921463  0.1145839 113.3598963 3.840857e-80
## V14         13.97840198  0.1218013 114.7639762 1.649920e-80
## V15         15.18194300  0.1160864 130.7813850 2.089741e-84
## V16         -0.25348570  0.1131778  -2.2397124 2.833458e-02
## V17          0.01483556  0.1106943   0.1340227 8.937748e-01
## V18         -0.06415616  0.1153952  -0.5559692 5.800303e-01
## V19          0.10845377  0.1196202   0.9066511 3.677467e-01
## V20         -0.08569465  0.1155533  -0.7416027 4.608448e-01
## V21          0.08445823  0.1077823   0.7836003 4.359563e-01
## V22         -0.03260947  0.1135547  -0.2871696 7.748431e-01
## V23          0.14536310  0.1405160   1.0344953 3.045168e-01
## V24         -0.13949381  0.1203951  -1.1586337 2.505999e-01
## V25          0.11556310  0.1298875   0.8897167 3.767096e-01
## V26         -0.04386926  0.1057928  -0.4146716 6.796687e-01
## V27          0.03132658  0.1166940   0.2684507 7.891534e-01
## V28         -0.08143632  0.1217150  -0.6690738 5.056809e-01
## V29          0.18756763  0.1104730   1.6978590 9.404042e-02
## V30         -0.26053825  0.1143442  -2.2785430 2.579376e-02
```

```r
which.max(summary(our_lm)$coefficients[-1, 4])
```

```
## V17
##  17
```

```r
summary(our_lm)$coefficients[1 + which.max(summary(our_lm)$coefficients[-1, 4]), 4]
```

```
## [1] 0.8937748
```

```r
summary(our_lm)$coefficients[1 + which.max(summary(our_lm)$coefficients[-1, 4]), 4] > alpha
```

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

```r
rem_inx <- which.max(summary(our_lm)$coefficients[-1, 4])
our_df2 <- our_df2[, -rem_inx]
head(our_df2)
```

```
##           V1          V2         V3         V4         V5          V6         V7         V8          V9        V10
## 1  0.3224893  0.25030239 -1.9636473 -0.1908958 -2.0698426  0.03007867 -0.3465919 -1.6387077 -1.11728829 -0.9176115
## 2  1.0393369  1.05237277  0.3497962  0.6707269 -0.5611060  0.34186776 -0.8532761 -0.1989156  0.86606228 -0.4679466
## 3  0.3620937  1.01229628 -0.8086659  0.2226521  0.6335299  0.52950537 -0.1756332  1.4316855 -0.67505447 -0.3019786
## 4  2.6952495 -0.62641386 -2.1724179 -0.2056112  0.7893220 -0.83824784 -0.5901817  0.1965325  0.39182686  0.4058325
## 5 -0.2220306 -0.09457502  0.7716704 -0.2585643  2.2205917 -0.48503468  0.5533806  1.8576283  0.66402339 -0.1985734
## 6 -0.3634943 -1.10253092  0.6916973 -0.1093809  0.2822249  1.06281705  2.2744877 -0.4048488  0.09959621 -0.9593692
##          V11        V12        V13          V14        V15        V16        V18        V19        V20         V21
## 1 -0.5830068  0.6304618 -0.8654048  1.509689693  0.5412971  1.8302474 -0.4728529  0.5889361 -1.2615153 -0.04870460
## 2  0.4492400  1.9564701 -1.2102455 -0.137096127 -0.7377183 -0.4491893  0.6796610  0.4137103 -0.8198740  0.68883600
## 3 -1.3379892 -0.4745657 -0.6164566  0.003067593 -0.4524744 -0.4871044 -0.9203098 -0.3989354 -1.6969118 -0.02751696
## 4  1.5107565  0.8944676 -0.5179551 -0.483952462  1.1033589 -0.6707489 -0.8145664 -0.7623925  0.6187703  0.05931249
## 5 -1.4624820  0.4559436 -1.1146398 -1.296972340  0.1997645  0.5987922  0.7043557 -0.2605895 -0.7210284 -1.41696491
## 6  0.6244589  0.4900047 -0.7109354  0.549266964 -0.2901797  0.6634210 -1.0958304 -0.4731946  2.1173039 -0.38024019
##          V22         V23        V24         V25        V26        V27        V28        V29         V30          Y
## 1 -1.3431972  0.64856203  0.5797339 -0.03215037 -1.5896355 -1.1321206  0.7047769 -0.6423863 -1.39824172 -30.584415
## 2 -0.6080119  1.21763716  1.2330384  0.16903728 -0.4077991  0.5755276 -0.1304502  0.9132925 -0.10033043   1.805559
## 3 -0.8826985  1.17527072 -0.8356647  1.09887973 -0.9023938  1.6324757 -0.3029581 -1.4975017 -0.63631529 -26.930862
## 4 -1.4094642  0.04196612  0.4802486 -0.72219856 -0.1497689 -0.2537769  0.2544970 -0.5918419 -0.52831015  27.381626
## 5 -1.4724190 -0.37445812  0.1518078  0.73959767 -1.0544945  1.6979354 -0.8621776 -0.1989525  0.02213683  -9.843954
## 6  2.3518774  0.74641089  0.1916580  0.84116729  0.8117128 -1.2624314 -1.4752433 -0.1463498 -0.09039265  18.746758
```

If any of the variables have a p-value greater than alpha, the `run_BE` function will repeat that process of removing the variable with the highest p-value until it settles on a model where all of the variables have significant p-values.

Let's try it on for size and see what happens.

```
BE <- run_BE(n,p,k,alpha)
BE
```

```
##                Estimate Std. Error     t value      Pr(>|t|)      2.5 %      97.5 % Known Param in CI?
## (Intercept)  0.04800017 0.10641783   0.4510538  6.531270e-01 -0.1636606  0.25966096                  1
## V1           1.09133063 0.11479535   9.5067496  6.330583e-15  0.8630073  1.31965396                  1
## V2           1.94643138 0.10615131  18.3363862  6.280812e-31  1.7353007  2.15756206                  1
## V3           2.93917958 0.10778728  27.2683331  3.453934e-43  2.7247950  3.15356416                  1
## V4           3.89647488 0.11345437  34.3439814  7.870633e-51  3.6708187  4.12213106                  1
## V5           5.06157529 0.10533804  48.0507823  2.358589e-62  4.8520622  5.27108842                  1
## V6           5.86670065 0.10237039  57.3085705  1.607229e-68  5.6630901  6.07031123                  1
## V7           6.73074186 0.12492797  53.8769828  2.367102e-66  6.4822652  6.97921856                  0
## V8           8.05005697 0.09805593  82.0965875  2.991217e-81  7.8550277  8.24508627                  1
## V9           9.11176843 0.11201749  81.3423762  6.375238e-81  8.8889702  9.33456670                  1
## V10          9.89245406 0.11379211  86.9344434  2.724995e-83  9.6661261 10.11878200                  1
## V11         10.89255897 0.11107753  98.0626669  1.364440e-87 10.6716302 11.11348771                  1
## V12         11.97748928 0.10670253 112.2512217  1.995421e-92 11.7652622 12.18971632                  1
## V13         13.12411071 0.10589207 123.9385608  5.632705e-96 12.9134956 13.33472578                  1
## V14         14.00384582 0.09495546 147.4780527 3.240556e-102 13.8149832 14.19270840                  1
## V15         15.08448887 0.09373288 160.9305975 2.370489e-105 14.8980579 15.27091980                  1
## V24         -0.23233522 0.10685045  -2.1743964  3.252226e-02 -0.4448565 -0.01981398                  0
```

Now that we know our BE program works, we can have it run `m` times and compute aggregate data of our `m` simulations. We want to know the proportion of times our model creates confidence intervals that contain the known parameter, as well as the proportion of the simulations that our model found each variable significant.

```r
run_simulation <- function(n, p, k, alpha, m) {
    if (m <= 0 | is.wholenumber(m) == FALSE) {
        return("m must be a positive integer")
    }
    if (alpha < 0 | alpha > 1) {
        return("alpha must be in the interval (0,1)")
    }
    if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {
        return("n and p must be positive integers")
    }
    if (n < p + 2) {
        return("n must be at least p+2")
    }
    if (k <= 0 | k > p | is.wholenumber(k) == FALSE) {
        return("k must be a positive integer not exceeding p")
    } else {
        CI_freq <- vector(length = p + 1)
        sig_freq <- vector(length = p + 1)
        full_df <- make_data_frame(n, p, k)
        var_names <- rownames(summary(lm(Y ~ ., full_df))$coefficients)
        names(CI_freq) <- var_names
        names(sig_freq) <- var_names
        for (i in 1:m) {
            display <- run_BE(n, p, k, alpha)
            CI_freq[1] <- CI_freq[1] + display[1, 7]
            sig_freq[1] <- sig_freq[1] + as.numeric(display[1, 4] <= alpha)
            for (j in 2:nrow(display)) {
                index <- as.numeric(str_sub(rownames(display)[j], 2, -1)) + 1
                CI_freq[index] <- CI_freq[index] + display[j, 7]
                sig_freq[index] <- sig_freq[index] + 1
            }
        }
        CI_perc <- CI_freq/m
        sig_perc <- sig_freq/m
        accuracy_mat <- cbind(round(CI_perc * 100, 2), round((sig_freq/m) * 100, 2))
        colnames(accuracy_mat) <- c("% Param in CI", "% Param Significant")
        return(accuracy_mat)
    }
}
```

Finally, let's go ahead and give it a whirl. We'll do a few different simulations with different n, p, k, α, and m each time. We'll start with our current values of 100, 30, 15, 0.05, and 125, respectively.

```
output <- run_simulation(n, p, k, alpha, m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)          93.6                 6.4
## V1                   96.8               100.0
## V2                   91.2               100.0
## V3                   88.8               100.0
## V4                   95.2               100.0
## V5                   89.6               100.0
## V6                   92.8               100.0
## V7                   95.2               100.0
## V8                   95.2               100.0
## V9                   94.4               100.0
## V10                  92.0               100.0
## V11                  89.6               100.0
## V12                  94.4               100.0
## V13                  92.0               100.0
## V14                  96.8               100.0
## V15                  92.0               100.0
## V16                   0.0                 8.0
## V17                   0.0                 8.8
## V18                   0.0                 9.6
## V19                   0.0                 4.8
## V20                   0.0                 1.6
## V21                   0.0                 7.2
## V22                   0.0                 4.0
## V23                   0.0                 8.0
## V24                   0.0                 4.8
## V25                   0.0                10.4
## V26                   0.0                 4.8
## V27                   0.0                 8.0
## V28                   0.0                 6.4
## V29                   0.0                 4.0
## V30                   0.0                 4.0
```

```
c(mean(output[2:(k + 1), 1]), mean(output[c(1, (k + 2):(p + 1)), 2]))
```

```
## [1] 93.06667  6.30000
```

One would expect that the model will be more accurate if you give it more data. We originally gave it 100 data points. Let's see what happens if we halve that to $n = 50$.

```
n <- 50
output <- run_simulation(n, p, k, alpha, m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)          91.2                 8.8
## V1                   88.8               100.0
## V2                   96.0               100.0
## V3                   88.0               100.0
## V4                   89.6               100.0
## V5                   92.8               100.0
## V6                   91.2               100.0
## V7                   92.8               100.0
## V8                   90.4               100.0
## V9                   90.4               100.0
## V10                  93.6               100.0
## V11                  91.2               100.0
## V12                  92.0               100.0
## V13                  91.2               100.0
## V14                  89.6               100.0
## V15                  92.0               100.0
## V16                   0.0                10.4
## V17                   0.0                13.6
## V18                   0.0                 6.4
## V19                   0.0                12.8
## V20                   0.0                 3.2
## V21                   0.0                 5.6
## V22                   0.0                10.4
## V23                   0.0                11.2
## V24                   0.0                 6.4
## V25                   0.0                11.2
## V26                   0.0                 8.8
## V27                   0.0                 8.8
## V28                   0.0                 8.0
## V29                   0.0                 8.0
## V30                   0.0                11.2
```

```
c(mean(output[2:(k + 1), 1]), mean(output[c(1, (k + 2):(p + 1)), 2]))
```

```
## [1] 91.30667  9.05000
```

Here, let's give the model less variables to work with, but let's make most of them "good".

```
p <- 10
k <- 7
output <- run_simulation(n, p, k, alpha, m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)          95.2                 4.8
## V1                   93.6               100.0
## V2                   92.8               100.0
## V3                   95.2               100.0
## V4                   94.4               100.0
## V5                   97.6               100.0
## V6                   92.8               100.0
## V7                   95.2               100.0
## V8                    0.0                 5.6
## V9                    0.0                 4.8
## V10                   0.0                 4.8
```

```
c(mean(output[2:(k + 1), 1]), mean(output[c(1, (k + 2):(p + 1)), 2]))
```

```
## [1] 94.51429  5.00000
```

Now, let's revert back to our original input parameters and change the alpha to see what happens.

```r
n<-100; p<-30; k<-15; alpha<-0.01; m<-125
output <- run_simulation(n,p,k,alpha,m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)          95.2                 1.6
## V1                   89.6               100.0
## V2                   97.6               100.0
## V3                   97.6               100.0
## V4                   95.2               100.0
## V5                   95.2               100.0
## V6                   95.2               100.0
## V7                   97.6               100.0
## V8                   92.8               100.0
## V9                   92.8               100.0
## V10                  92.8               100.0
## V11                  97.6               100.0
## V12                  95.2               100.0
## V13                  93.6               100.0
## V14                  96.0               100.0
## V15                  92.8               100.0
## V16                   0.0                 0.0
## V17                   0.0                 0.8
## V18                   0.0                 0.8
## V19                   0.0                 0.8
## V20                   0.0                 3.2
## V21                   0.0                 0.8
## V22                   0.0                 0.8
## V23                   0.0                 2.4
## V24                   0.0                 0.0
## V25                   0.0                 1.6
## V26                   0.0                 0.8
## V27                   0.0                 1.6
## V28                   0.0                 0.8
## V29                   0.0                 0.8
## V30                   0.0                 0.8
```

```r
c(mean(output[2:(k+1),1]),mean(output[c(1,(k+2):(p+1)),2]))
```

```
## [1] 94.77333  1.10000
```

Finally, let's make it really work. Let's say we have 200 data points on 100 predictor variables, of which 35 of them are "valid". We will run the simulation 1,000 times using $\alpha = 0.02$. Let's see how it plays out!

```
n<-200; p<-100; k<-35; alpha<-0.02; m<-1000
output <- run_simulation(n,p,k,alpha,m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)          93.3                 2.9
## V1                   94.3               100.0
## V2                   94.1               100.0
## V3                   93.8               100.0
## V4                   94.2               100.0
## V5                   91.9               100.0
## V6                   92.1               100.0
## V7                   92.0               100.0
## V8                   93.1               100.0
## V9                   94.2               100.0
## V10                  92.1               100.0
## V11                  92.9               100.0
## V12                  93.4               100.0
## V13                  92.1               100.0
## V14                  93.5               100.0
## V15                  94.1               100.0
## V16                  93.4               100.0
## V17                  92.7               100.0
## V18                  93.6               100.0
## V19                  93.9               100.0
## V20                  92.8               100.0
## V21                  92.4               100.0
## V22                  94.7               100.0
## V23                  93.0               100.0
## V24                  93.0               100.0
## V25                  90.9               100.0
## V26                  93.1               100.0
## V27                  92.2               100.0
## V28                  93.5               100.0
## V29                  91.5               100.0
## V30                  93.6               100.0
## V31                  93.7               100.0
## V32                  94.8               100.0
## V33                  93.0               100.0
## V34                  93.1               100.0
## V35                  93.5               100.0
```

```
## V36                    0.0              3.2
## V37                    0.0              4.0
## V38                    0.0              2.4
## V39                    0.0              2.3
## V40                    0.0              2.4
## V41                    0.0              2.9
## V42                    0.0              2.6
## V43                    0.0              2.5
## V44                    0.0              2.9
## V45                    0.0              1.8
## V46                    0.0              2.9
## V47                    0.0              3.2
## V48                    0.0              3.7
## V49                    0.0              3.2
## V50                    0.0              2.6
## V51                    0.0              2.7
## V52                    0.0              2.1
## V53                    0.0              2.8
## V54                    0.0              2.1
## V55                    0.0              2.3
## V56                    0.0              2.7
## V57                    0.0              2.8
## V58                    0.0              2.7
## V59                    0.0              1.8
## V60                    0.0              3.1
## V61                    0.0              2.3
## V62                    0.0              2.3
## V63                    0.0              2.3
## V64                    0.0              2.6
## V65                    0.0              3.2
## V66                    0.0              3.1
## V67                    0.0              2.7
## V68                    0.0              2.0
## V69                    0.0              2.2
## V70                    0.0              2.4
## V71                    0.0              1.7
## V72                    0.0              3.0
## V73                    0.0              2.2
## V74                    0.0              3.3
## V75                    0.0              2.5
## V76                    0.0              2.1
## V77                    0.0              2.9
## V78                    0.0              2.9
## V79                    0.0              2.5
```

```
## V80                    0.0                 2.1
## V81                    0.0                 2.9
## V82                    0.0                 1.8
## V83                    0.0                 2.5
## V84                    0.0                 2.8
## V85                    0.0                 2.9
## V86                    0.0                 3.9
## V87                    0.0                 2.4
## V88                    0.0                 3.1
## V89                    0.0                 2.9
## V90                    0.0                 2.3
## V91                    0.0                 2.9
## V92                    0.0                 2.6
## V93                    0.0                 3.3
## V94                    0.0                 2.8
## V95                    0.0                 3.4
## V96                    0.0                 2.8
## V97                    0.0                 2.0
## V98                    0.0                 3.1
## V99                    0.0                 2.7
## V100                   0.0                 1.9
```

```
c(mean(output[2:(k+1),1]),mean(output[c(1,(k+2):(p+1)),2]))
```

```
## [1] 93.148571  2.665152
```