Results

Harpeth Lee

4/2/2022

```
## Loading required package: ggpp
```

```
##
## Attaching package: 'ggpp'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

Results

Data Simulation

Our data simulations are done based on the latent variable model that PLS methods assume. We generate $\mathbf{X}$ and $\mathbf{Y}$ using the following equations, $\mathbf{X} = \boldsymbol{\phi}\mathbf{A} + \boldsymbol{\epsilon}_X$ and $\mathbf{Y} = \boldsymbol{\phi}\mathbf{D} + \boldsymbol{\epsilon}_Y$. The $\mathbf{A}$ and $\mathbf{D}$ matrices are fixed, providing most of the structure seen in $\mathbf{X}$ and $\mathbf{Y}$. $\boldsymbol{\phi}$, $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$ are randomly generated. The method by which $\boldsymbol{\phi}$ is generated is less important as we want to approximate $\boldsymbol{\phi}$ with our final model. $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$ are sampled from a normal distribution.

Although $\boldsymbol{\phi}$ is generated using a similar method as $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$, it is important to note that $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ are noise while $\boldsymbol{\phi}$ is not. Ideally, our models will be able to identify $\boldsymbol{\phi}$ while ignoring $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$. $\boldsymbol{\phi}$ is generated sampling from a multivariate normal distribution with uncorrelated samples(using the `mvrnorm` function in the `MASS` package in order to make sure columns are uncorrelated). $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ are generated by samples from random distributions (using `rnorm`) as this will cause some random correlation in the noise. This random correlation is included as we want to test model's ability to distinguish true correlation from random correlation.

When generating the data, there are four variables related to the dimensionality of the matrices which must be chosen, n, p, q, and R. n is the number of observations generated, p is the number of predictors, and q is the number of response variables. Of these variables, R is the only one of these variables that is not clear from the generated data. R is the size of the dimension of $\mathbf{A}$,$\mathbf{D}$, and $\boldsymbol{\phi}$ that is lost when matrix multiplication is performed. Since we want $\mathbf{X}$ to be of dimension $n \times p$ and $\mathbf{Y}$ to be of dimension $n \times q$, $\boldsymbol{\phi}$ will be of dimension $n \times R$, $\mathbf{A}$ will be of dimension $R \times p$, and $\mathbf{D}$ will be of dimension $R \times q$. Although it may seem like we will want our final model to have R total components, we will want K to be the number of eigenvectors of the variance matrix of $\mathbf{X}$ that have non-null projections onto the covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$. [Helland and Almoy]

There were a number of methods used to generate these matrices. $\boldsymbol{\phi}$ was always generated from uncorrelated normal distributions. Two methods were used to generate $\mathbf{A}$, the "simple" method and the "complex" method. In the simple method,

$$\mathbf{A} = \begin{pmatrix} \mathbf{1}_{\frac{3}{5}R \times 5} & \mathbf{0}_{\frac{3}{5}R \times 10} & \mathbf{0}_{\frac{3}{5}R \times (p-15)} \\ \mathbf{0}_{\frac{2}{5}R \times 5} & \mathbf{1}_{\frac{2}{5}R \times 10} & \mathbf{0}_{\frac{2}{5}R \times (p-15)} \end{pmatrix}$$

Note that $p \geq 15$ in order for this method to work. The dimensions of $\frac{3}{5}R$ and $\frac{2}{5}R$ are not exact, we round and then take the difference in order to make sure values are integers that add up to R.

Using the "complex" method,

$$\mathbf{A} = \begin{pmatrix} \mathbf{1}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \\ \mathbf{0}_{2\times 5} & \mathbf{1}_{2\times 10} & \mathbf{0}_{2\times 5} & \mathbf{0}_{2\times 1} & \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times (p-23)} \\ \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{1}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \\ \mathbf{0}_{2\times 5} & \mathbf{0}_{2\times 10} & \mathbf{0}_{2\times 5} & \mathbf{1}_{2\times 1} & \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times (p-23)} \\ \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \end{pmatrix}$$

Note that $p \geq 23$ in order for this method to work. Furthermore, $R = 13$ whenever this method is used.

A wider number of methods were used to generate the $\mathbf{D}$ matrix. We will call these methods "basic", "diagonal", "simple", "2-block", and "4-block". In the basic method,

$$\mathbf{D} = \mathbf{1}_{R\times q}$$

Using the "diagonal" method,

$$\mathbf{D} = \mathbf{I}_{R\times q}$$

Using the "simple" method

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{R\times 1} & \mathbf{0}_{R\times (q-1)} \end{pmatrix}$$

Using the "2-block" method

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{\frac{3}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{3}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{3}{5}\times \frac{1}{2}q} \\ \mathbf{0}_{\frac{2}{5}R\times \frac{1}{4}q} & \mathbf{1}_{\frac{2}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{2}{5}\times \frac{1}{2}q} \end{pmatrix}$$

Note the two blocks of 1s hence the name "2-block" method.

Using the "4-block" method

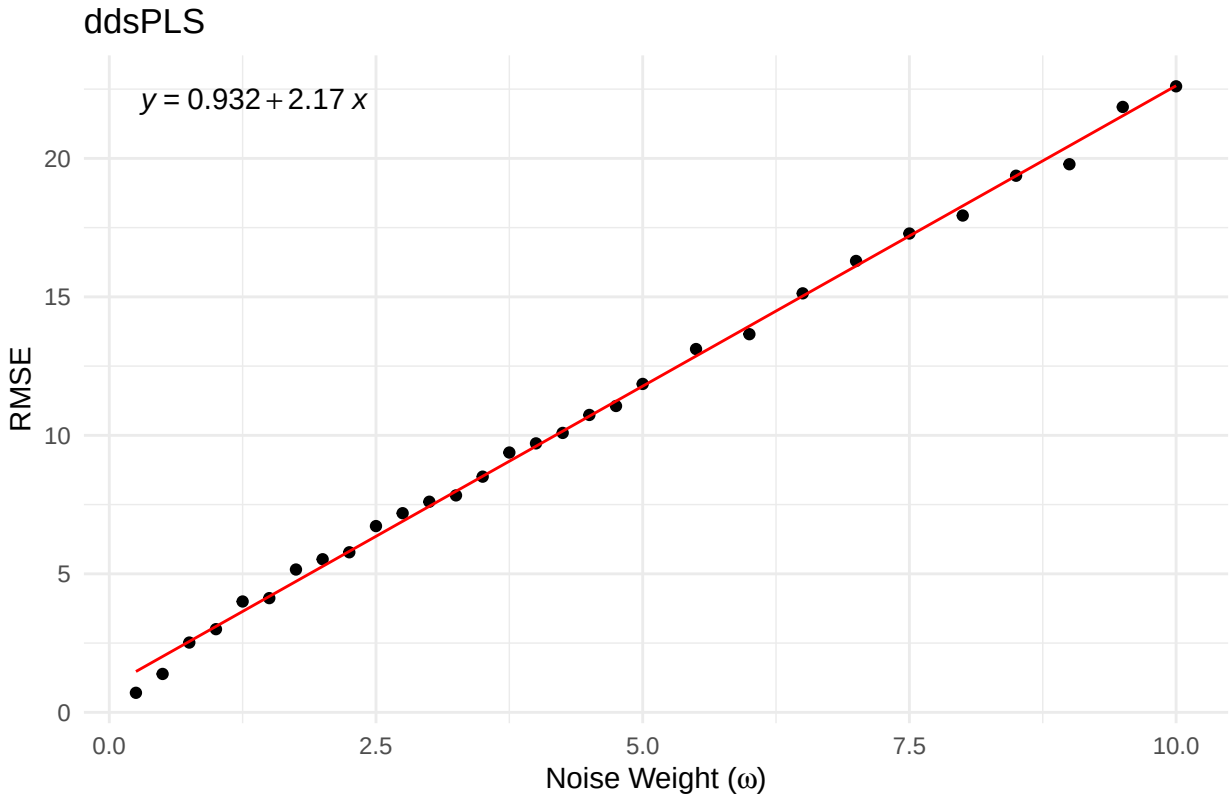
$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times (q-4)} \end{pmatrix}$$

Note that $R = 13$ at all times when the "4-block" method is used.

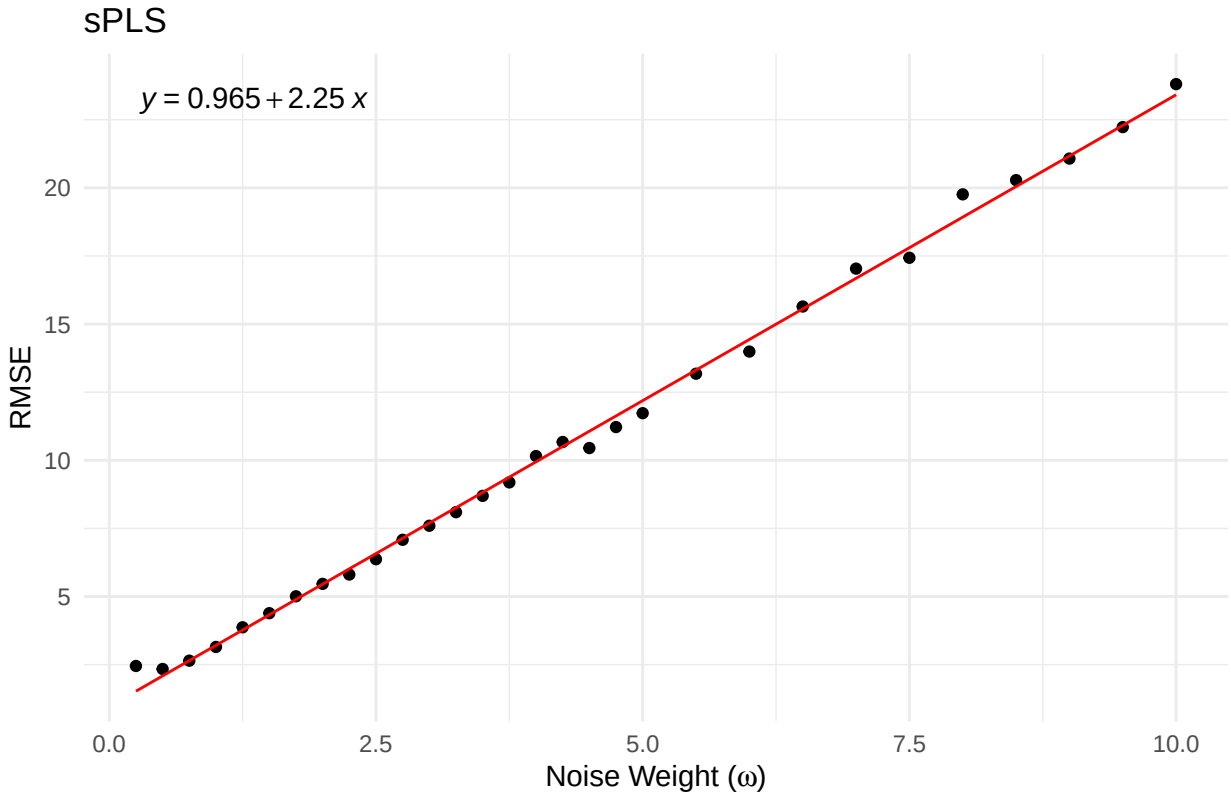
The $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ matrices are generated from samples from a standard normal distribution and then scaled by a factor ω, called the noise weight. $\boldsymbol{\epsilon}_X$ is of dimension $n \times p$ and $\boldsymbol{\epsilon}_Y$ is of dimension $n \times q$. Since the noise is randomly generated we expect to see some correlation occur both between predictors and between the predictors and the responses.

Noise

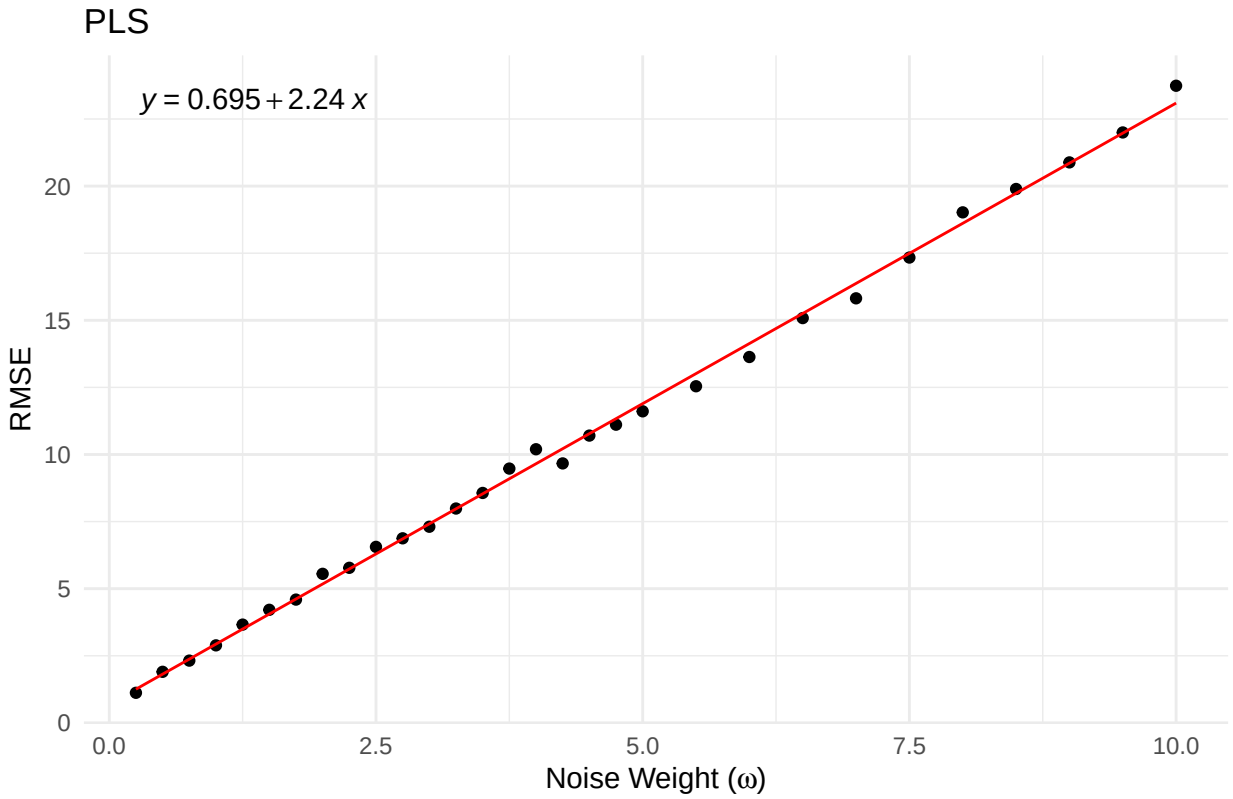
When examining model performance, it is important to consider how it performs under conditions where the amount of noise varies. The following graphs show the relationship between ω and the RMSE for the ddsPLS, sPLS, and PLS methods. Note the extremely linear relation displayed in all three.



Relation between ω and the test set RMSE for the ddsPLS model. n=100, p=100, q=5



Relation between ω and the test set RMSE for the sPLS model. n=100, p=100, q=5



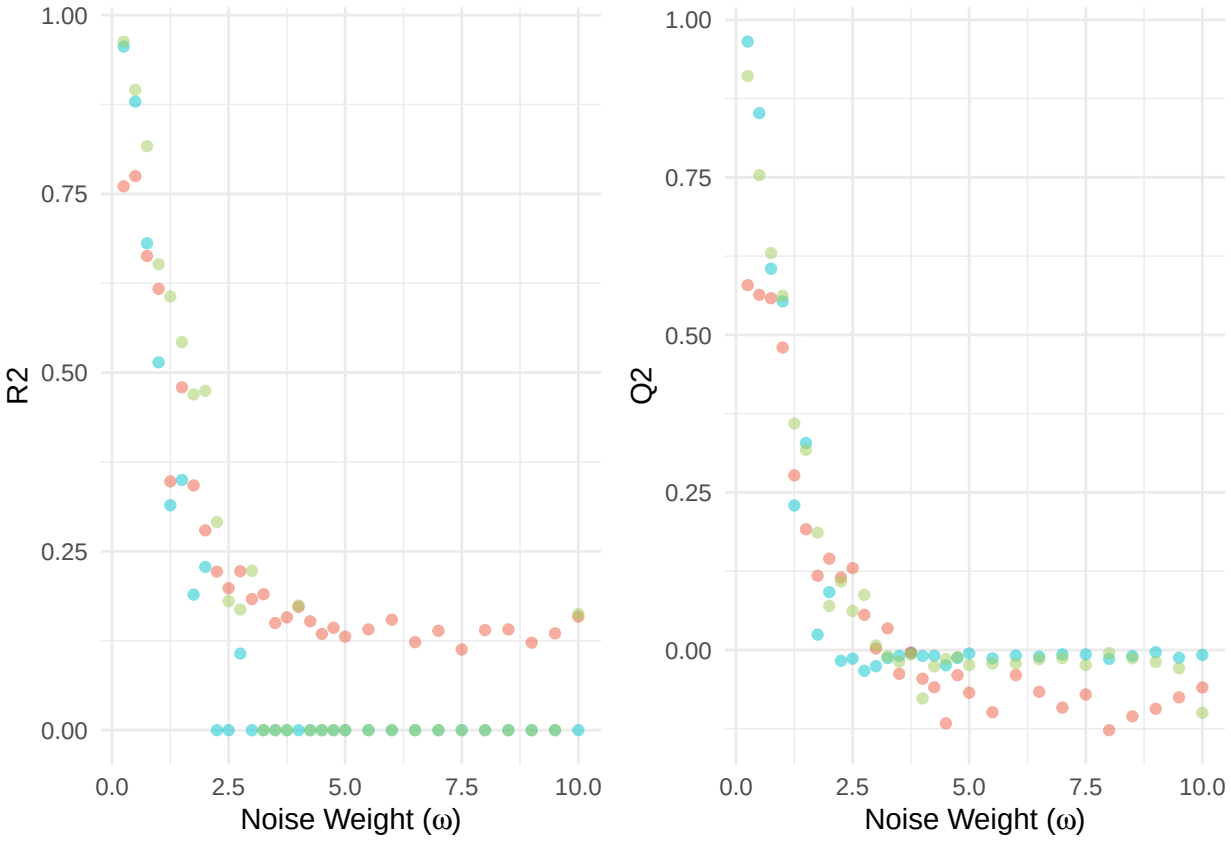
Relation between ω and the test set RMSE for the PLS model. n=100, p=100, q=5

Note the similar slopes of the line of best fit for all model types. Ultimately, these plots suggest that there is not a large difference in performance between model types as the amount of noise is increased. We should be careful about drawing any conclusions from results at higher values of ω as these models are usually just performing mean estimation and are not as dependent on the model used.

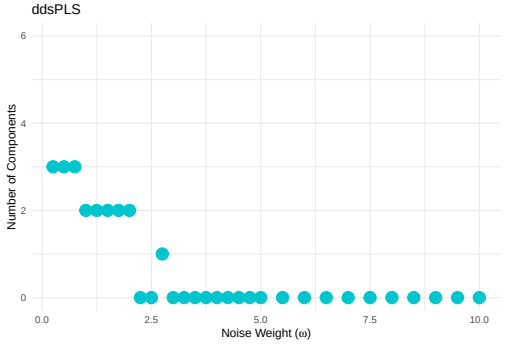
Now lets compare R^2 and Q^2 as ω changes.

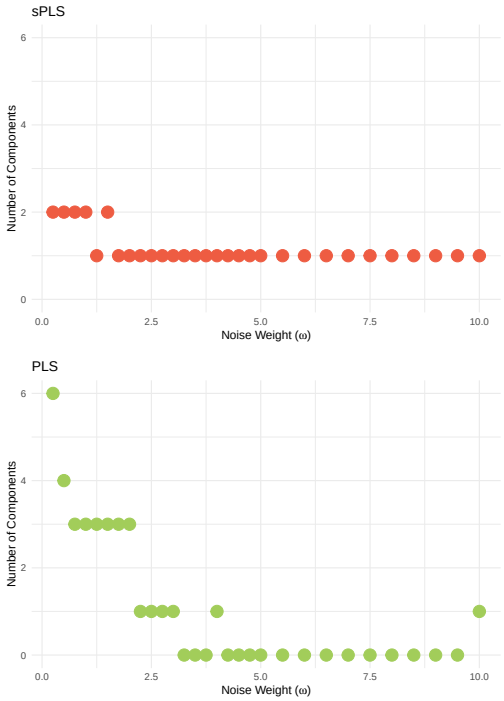
```
R2_plot <- ggplot(samp_test_ddspls,
       aes(x = noise_weight, y = R2)) +
  geom_point(color = "turquoise3",
             alpha = 0.5) +
  geom_point(data = samp_test_spls,
             aes(x = noise_weight, y = R2),
             color = "tomato2",
             alpha = 0.5) +
  geom_point(data = samp_test_pls,
              mapping = aes(x = noise_weight, y = R2),
             color = "darkolivegreen3",
             alpha = 0.5) +
  labs(x = unname(TeX("Noise Weight ($\\omega$)"))) +
  theme_minimal()

Q2_plot <- ggplot(samp_test_ddspls,
       aes(x = noise_weight, y = Q2)) +
  geom_point(color = "turquoise3",
             alpha = 0.5) +
  geom_point(data = samp_test_spls,
             aes(x = noise_weight, y = Q2),
```

```
              color = "tomato2",
              alpha = 0.5) +
  geom_point(data = samp_test_pls,
              mapping = aes(x = noise_weight, y = Q2),
              color = "darkolivegreen3",
              alpha = 0.5) +
  labs(x = unname(TeX("Noise Weight ($\\omega$)"))) +
  theme_minimal()

grid.arrange(R2_plot, Q2_plot, nrow = 1)
```

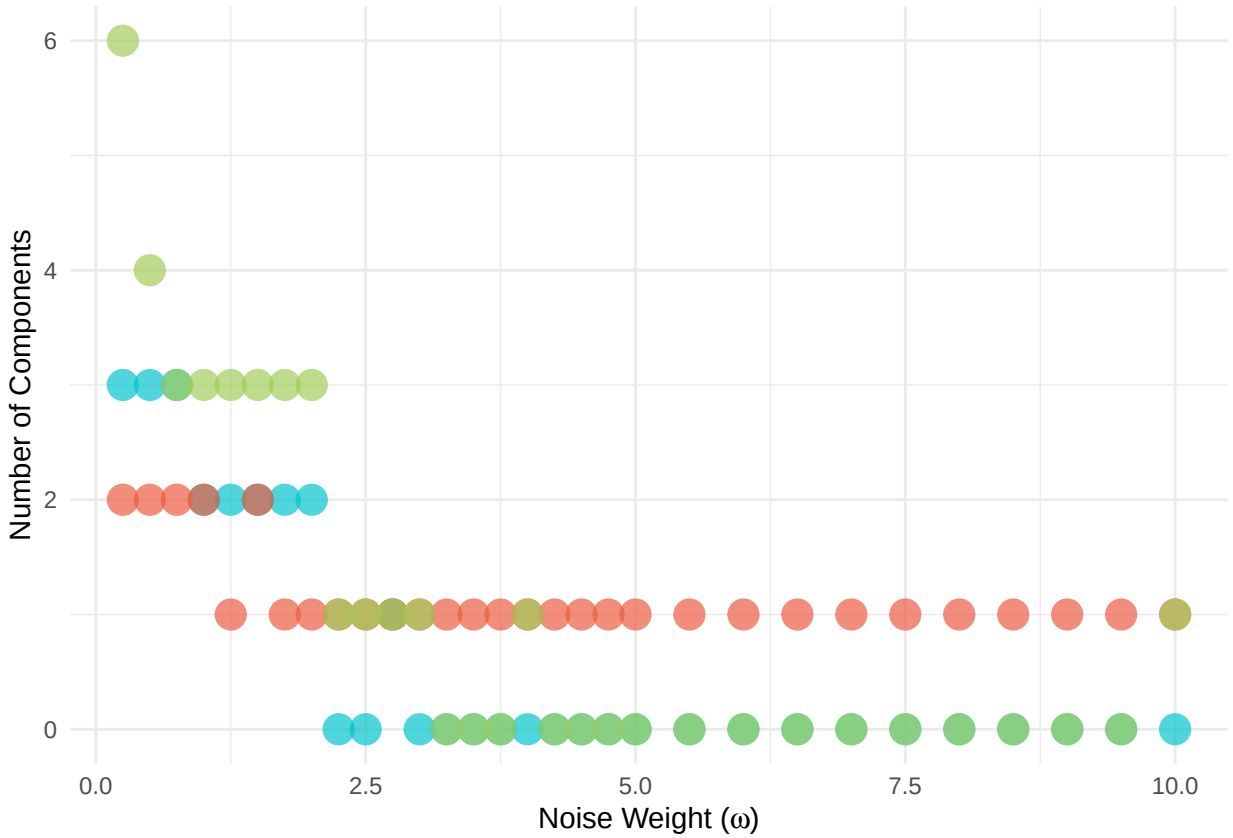


Here we can see the issues that arise with the method sPLS uses to choose the number of components to build. When ddsPLS and PLS start performing mean estimation, sPLS continues to build a model. This leads to a higher R^2 but a lower Q^2 meaning models perform worse on test data than mean estimation.






```
ggplot(samp_test_ddspls, aes(x = noise_weight, y = ncomp)) +
  geom_point(size = 5, alpha = 0.7, color = "turquoise3") +
  geom_point(data = samp_test_spls,
             aes(x = noise_weight, y = ncomp),
             size = 5,
             alpha = 0.7,
             color = "tomato2") +
  geom_point(data = samp_test_pls,
             aes(x = noise_weight, y = ncomp),
             size = 5,
             alpha = 0.7,
             color = "darkolivegreen3") +
  labs(x = unname(TeX("Noise Weight ($\\omega$)")),
       y = "Number of Components") +
  ylim(c(0,6)) +
  theme_minimal()
```

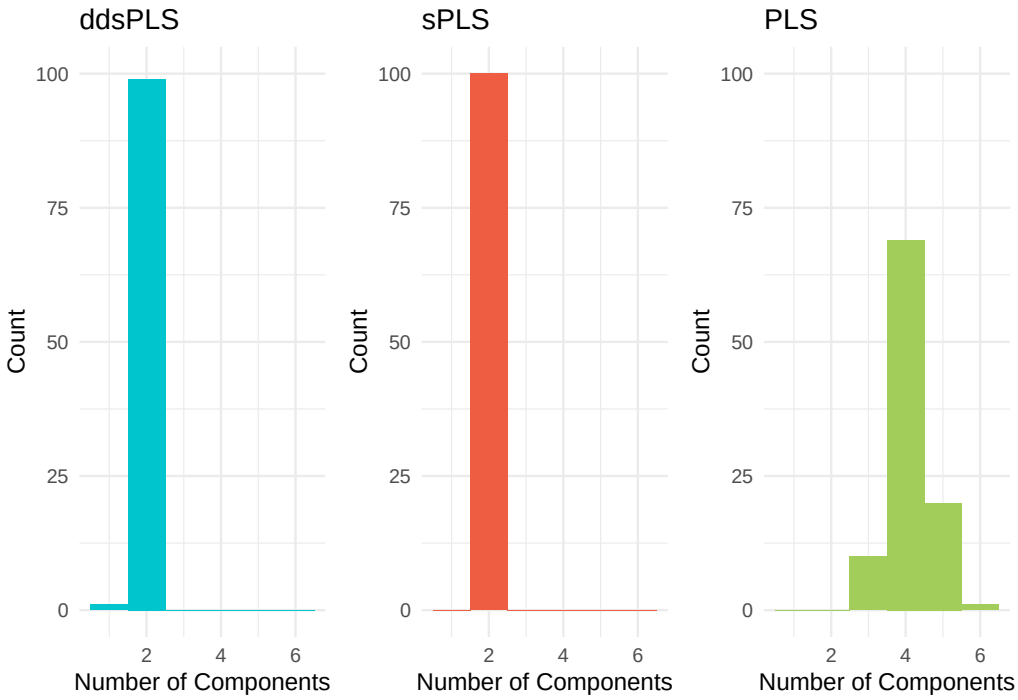


Here is the number of components built as the noise increases across model types. It is important to note that the sPLS tuning algorithm used always selects at least one component. When 0 components are included in the model this means that mean estimation performs better than the model built. Not building any components at high noise levels is not necessarily a bad sign as PLS methods are not designed to deal with problems due to high noise to signal ratios, it simply means that the noise included has drowned out the signal.

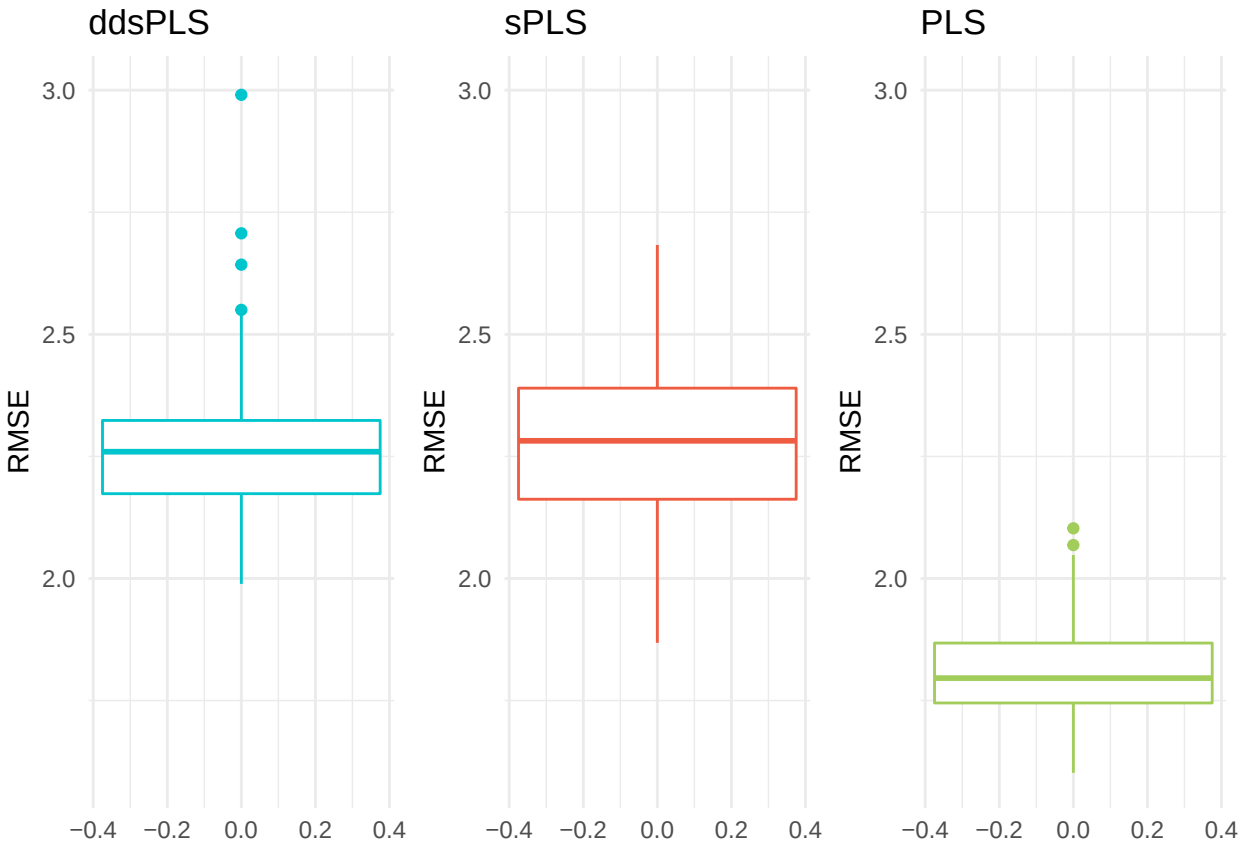
Here we can see that PLS tends to build more components than other models as we would expect since it makes no attempt to build sparser models.

Lets now compare model performance at fixed values of ω when $\mathbf{A}$ is generated using the complex method and $\mathbf{D}$ is generated using the 2-block method. For these trials we will fix $n = 100$, $p = 100$, and $q = 5$.

First let $\omega = 0.5$. At this noise level we can expect fairly robust and preictive models to be built.



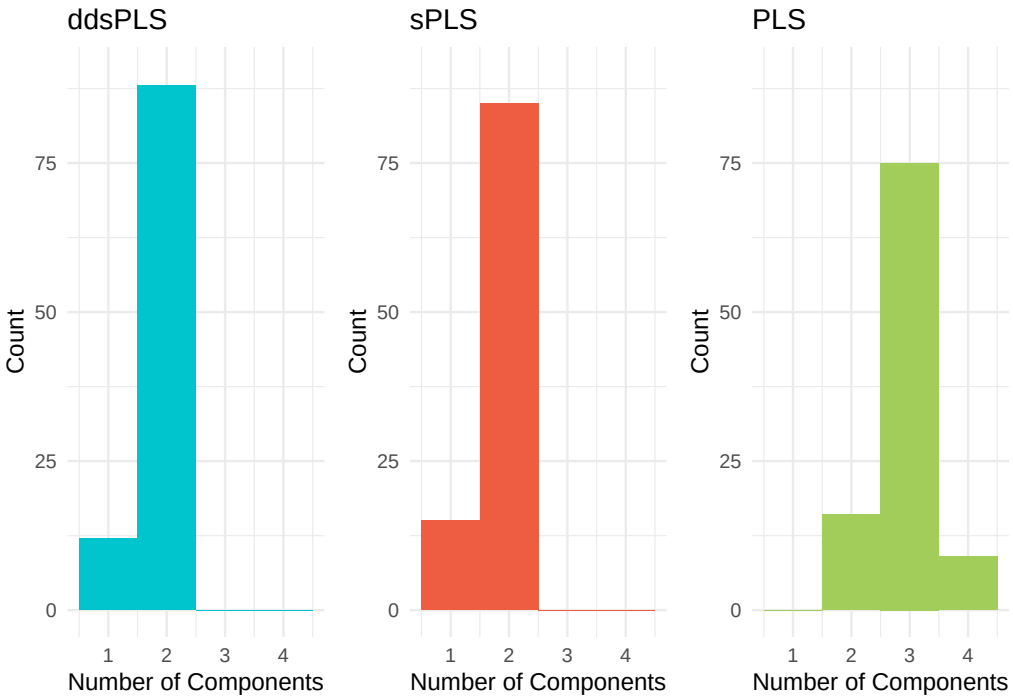
Due to the shape of the data, models should include 2 components. ddsPLS and sPLS both tend to build the correct number of components with sPLS building 2 components every time. PLS always builds too many components and never builds the correct number.



At $\omega = 0.5$, we can see that ddsPLS and sPLS perform similarly in regards to to RMSE while PLS performs

significantly better.

Now let $\omega = 1$.



ddsPLS and sPLS still tend to build the correct number of components when ω is increased. Both show an increased tendency to build too few components. PLS now builds fewer components but still tends to build too many components.

Predictors