A Comparison of Data-driven Partial Least Squares to Existing Partial Least Squares Models

---

A Thesis

Presented to

The Division of Mathematics and Natural Sciences

Reed College

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Arts

Harpeth Lee

May 2022

Approved for the Division
(Mathematics)

Hank Ibser

Acknowledgements

First I would like to thank Hank Ibser, my thesis advisor for his invaluable support and input throughout the process. Without his guidance this thesis would not exist. Secondly I would like to thank Jonathan Wells for his help writing this thesis as well as an advisor and professor throughout my time at Reed. I also want to thank Kara Cerveny, the work I did with her inspired my interest in high dimensional data that ultimately resulted in this thesis. Finally I would like to thank my friends for both distracting me and telling me to get back to work.

Table of Contents

List of Figures

Abstract

Data-driven sparse partial least squares is a recently proposed sparse partial least squares method seeking to improve on existing methodology. Data-driven sparse partial least squares is designed to perform well in situations with high numbers of uncorrelated predictor and response variables by incorporating an L_1 norm directly on the empirical covariance matrix which leads to sparsity among both predictors and responses. This approach is different than existing sparse partial least squares methods. In this thesis, data-driven sparse partial least squares is compared to two existing partial least squares models to determine if it performs comparably or better to existing models. We find that data-driven partial least squares outperforms a popular existing sparse partial least squares method in most cases while outperforming traditional partial least squares in cases in situations with high numbers of uncorrelated predictor and response variables. Furthermore, data-driven sparse partial least squares models tend to more accurately reflect the underlying structure of the data than similar models. We do observe that data-driven sparse partial least squares can struggle with higher variance in model performance than similar models.

Introduction

Advances in technology in the past half century or so have allowed for the collection and processing of data on a scale never seen before. While this change has caused the field of statistics to flourish, it has also demanded that statistical methods evolve to handle these large amounts of data. Most more traditional statistical methods assume that all of the data we are presented with is relevant to answering the questions at hand. When faced with huge amounts of data, this assumptions is often not true. A large number of modern statistical methods often find ways to simplify data, identifying only the relevant information that is present. One area of statistics where this problem is prominent is regression.

Regression, a variety of methods where a number of known variables are used to predict the values of unknown variables, is one of the cornerstones of statistics. Regression has wide variety of uses and is used in many other fields. Regression techniques are used for everything from tumor classification to predicting how much it will rain next week. The variables that we use to make our prediction are called predictors while the variables that we try to predict are called response variables.

Here is a good place to introduce the ideas of signal and noise. In statistics, signal is the relevant information that is included in the data while noise is information that is present but irrelevant to the question at hand. We can think of this as trying to have a conversation with someone in a noisy room. Here the signal is the conversation that we are having. As the amount of noise increases, it will be harder to understand the signal. In regression, the signal is the underlying relationship shared among the data we are interested in while noise is random variation that is present in our data. When performing regression, we want to build models that include the signal while ignoring the noise. As we collect more data, it is almost always the case that we will be adding more noise.

One of the central problems of regression is the bias-variance tradeoff. Bias refers to the amount of error made in our attempt to fit or model to the given data, we want models to have low bias as this mean that they are able to closely fit the given data.

If the bias is high, this means that our model is *underfitting* the data and ignoring the signal that is present. Variance refers to how much the model changes when built using different data. We want models to have low variance as this means that they are built on the signal present in the data and not the noise. If the variance is high, the model is *overfitting* the data and including too much noise. This model will perform well on the data used to build the model but not on other data.

It is extremely hard to build models that have low bias and low variance, instead we usually have to settle for a tradeoff between the two. Models with low bias tend to have high variance as they will fit noise included in the data alongside the signal. On the other hand, models with low variance tend to have high bias as they will miss some of the signal.

This thesis investigates the performance of data-driven sparse partial least squares, a recently proposed partial least squares model, in comparison to more established similar methods Lorenzo, Saracco, & Rodolphe (2010). Data-driven sparse partial least squares (ddsPLS) is a regression method designed to be used on large datasets containing many weakly or uncorrelated predictors and response variables. Models built on large datasets often have high variance due to the large amount of noise that often comes along with a large number of variables. ddsPLS attempts to solve this problem by either ignoring or lessening the effect that weakly or uncorrelated predictors and response variables have on the final model.

Given how recently it was proposed, the only existing research into the performance of ddsPLS was done by alongside the models proposal. In this thesis, I perform a more in depth exploration of the performance of ddsPLS in comparison to two similar models. I find that ddsPLS continues to give promising results and is deserving of further inquiry and applied use.

When evaluating model performance in this thesis we have two general goals for models that perform well: making accurate predictions and correctly identifying the underlying structure of the data. These two goals are often achieved at the same time, models that accurately identify the underlying structure of data often make accurate predictions. However, it is often possible to make models that perform well without identifying the underlying structure of the data. In this paper this often manifests in overly complex models that perform well without reflecting the method used to generate the data.

I find that ddsPLS tends to make more accurate predictions than existing models in situations with high numbers of weakly and uncorrelated predictors and response variables. In less complex cases, traditional partial least squares outperforms ddsPLS.

ddsPLS is almost always able to better identify the underlying structure of the data when compared to similar models. At times, ddsPLS overcompensates for bias by ignoring almost all of the noise present in the data along with some of the signal.

This thesis is broken into three main sections, Foundations and Context, Theory, and Results. The Foundations and Context section contains an overview of related statistical methods and problems alongside a review of the literature preceding and related to the proposal of ddsPLS. The Theory section gives more detail on key methods that are related to or used in ddsPLS and discussion of the algorithms used for the three partial least squares algorithms discussed in detail. The Results section contains the methods used to simulate data for testing the models alongside the results from simulations run with the three model types with discussion of these findings.

Chapter 1

Foundations and Context

1.1 Background

The following section discusses a number of topics related to data-driven sparse partial least squares. This section is not meant to exhaustively explore these topics, but rather to provide some important background information and serve as a refresher to some of the statistical methods and problems that will be addressed. More in depth discussion of all of the subjects mentioned here can be found in The Elements of Statistical Learning by Hastie, Tibshirani, & Friedman (2009).

1.2 Dimension Reduction

Dimension reduction is a common technique used for working with large data sets with the goal moving data from a high-dimensional feature space to a low-dimensional feature space while retaining key features of the original data. The feature space is the mathematical space which our data resides in. For example, if our data consists of observations of 10 variables that take real values, our feature space will be $\mathbb{R}^{10}$. By moving data to a lower dimensional feature space, we make the data more interpretable and easier to work with. Furthermore, we can also remove features of the data that aren't relevant as these features may interfere with relevant information. Approaches to dimension reduction can generally broken into one of two categories; feature selection and feature extraction. Feature selection techniques remove unneeded features from the data. For example, feature selection before building a regression model will often include remove predictors with low correlation to the response variable and predictors with high correlation. Feature extraction techniques will create a new set of features

that capture relevant information about the original data. For example, a feature extraction technique may create linear combinations of our original predictors. It is important to note that feature selection will return a subset of variables from the original data set while feature extraction will return a set of new variables. Commonly used dimension reduction tehcniques include best subset selection, LASSO, PCA, and adhoc variable selection. There is no single standard for deciding when and how to perform dimension reduction as it depends on a number of factors including but not limited to the size of the dataset, the methods being used, and the amount of precision needed in results. Dimension reduction is commonly used with large datasets to remove features that are mostly noise and to make computations less costly. In addition, dimesnion reduction can be used to make models more interpretable. For example, when performing linear regression with a data set originally containing 25 predictors one may select only 5 of the predictors to include in the model in order for the model to be simpler and easier to interpret.

1.3 Curse of Dimensionality

At first, dimension reduction may seem like a counter intuitive approach to take when working with data. It seems that the more features we have, the more we should be able to learn about our data. However, when working with large datasets, the curse of dimensionality comes into play. The curse of dimensionality is a term used to refer to issues that arise due to the size of the feature space. As the dimension of feature space increases, data tends to become increasingly sparse making trends in data more difficult to recognize. One way to think of how this problem manifests is to consider how the ratio of the volume of ball and the hypercube containing the ball changes as the dimension d increases.

The volume of a ball, B, of radius r in n dimensions is given by

$$\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^n$$

The volume of the hypercube,C, containing this ball (with sides of length $2r$) is given by

$$(2r)^n$$

Taking the ratio we have

$$\frac{\text{Vol}(B)}{\text{Vol}(C)} = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)\cdot 2^n} \text{ so } \lim_{n\to\infty}\frac{\text{Vol}(B)}{\text{Vol}(C)} = 0$$

Consider the case where the ball has radius 1,

n	Ratio
1	1
2	0.7854
5	0.1645
10	2.49*e-3
25	2.854e-11
100	1.868e-70

Table showing how the ratio of the volume of a ball to the volume of a hybercube decreases as the number of dimensions increases.

Thus, we can expect the number of points within distance 1 of a point will decrease to 0 as the dimension of data increases.

Another problem that arises from high dimensional data is computational time. The computation complexity of fitting a linear regression model to an $n \times p$ matrix of predictors is $O(np^2 + p^3)$ (many programs will not perform the full matrix inversion so the complexity is usually smaller in practice however the general principle of computational complexity increasing holds). This will quickly balloon for large p. More troubling in high dimensions is that for p predictors, there are 2^p possible combinations of predictors making techniques such as *Best Subset Selection* computationally prohibitive.

Working with high dimensional data presents a number of issues which make it more difficult to extract meaningful finding from data. By using dimension reduction techniques we can mitigate many of these issues however many feature selection techniques will be less successful as they require us to discard a large amount of the data and may have a hard time finding the most meaningful features. Instead, we may want to turn to extraction methods that project the data into a lower dimensional space while still maintaining as much of the structure as we can.

1.4 Linear Regression

Linear regression is perhaps the most commonly used regression method given its simplicity and that it often yields fairly well performing models. Many other regression techniques use linear regression as a foundation. Linear regression works to find the line in n space that minimizes the Mean Squared Error (the squared distance between the line and observed data). The MSE is used in order to ensure that there is a unique answer. In linear regression we are looking for terms $\beta_0, ..., \beta_n$ such that $\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$ where $\hat{y}$ is our predicted y value. The following plot shows the line of best fit for a randomly generated set of points.

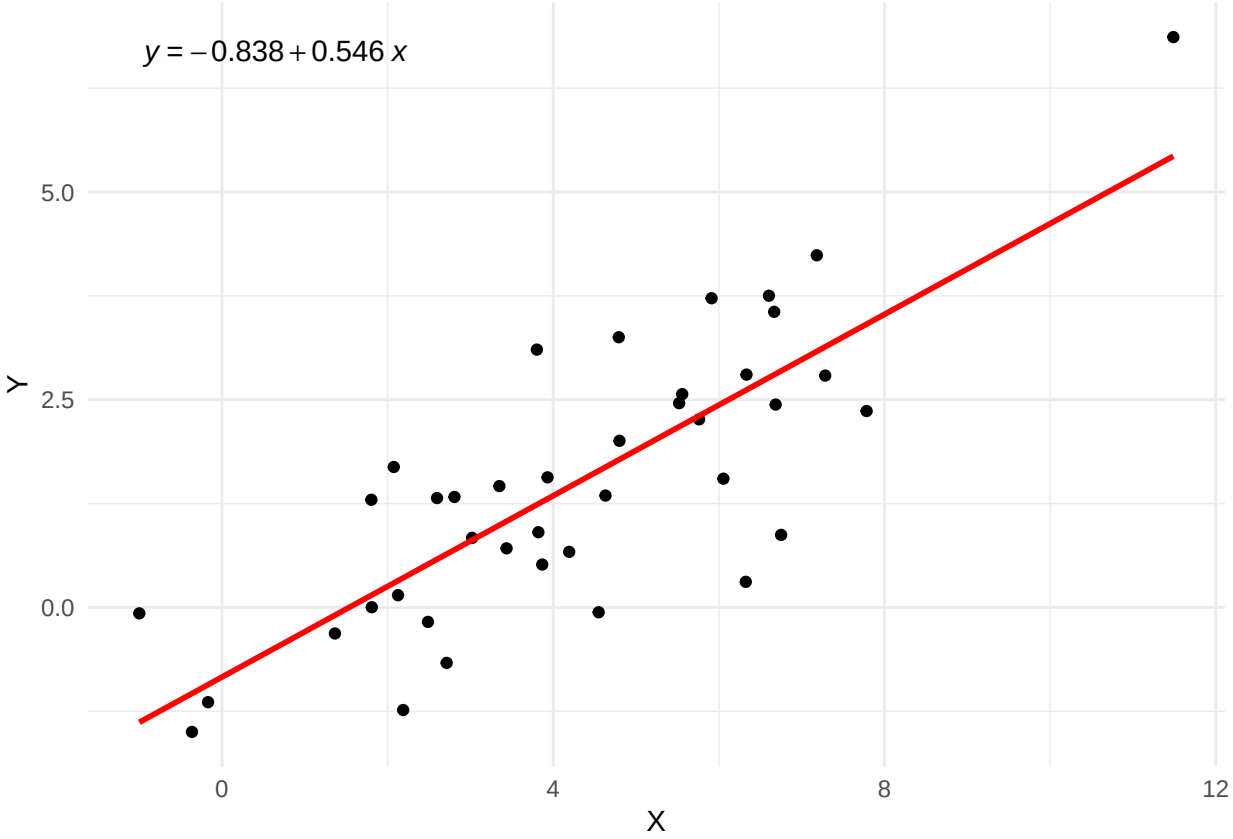

Figure 1.1: Demonstration of linear regression on randomly generated data.

Linear regression is one of the most commonly used regression methods due to its simplicity and wide applicability. Many other modeling methods use linear regression as a foundation or central part of building a model. Despite its wide applicability, simple linear regression is not a perfect model. For one it is highly susceptible to the curse of dimensionality as models become more variable. Furthermore, if $n > p$ simple

linear regression will not be able to produce a unique model as there will be an infinite number of hyperplanes in the feature space that minimize the sum of squares. In order to solve this problem, the data must be moved into a lower dimensional feature space.

1.5 Principal Component Analysis

Principal component analysis(PCA) is a commonly used unsupervised learning technique that can be used to reduce the dimension of data. PCA works by finding the principle components of a dataset, these can be thought of as the direction along which the data varies the most in the feature space. These principal components will preserve as much of the structure of the data as possible while projecting it into lower dimensions. For those of you with a background in linear algebra, the principal components will be the eigenvectors of the covariance matrix in order of the norm of the eigenvalues. PCA is one of the most commonly used dimension reduction techniques as it is often able to capture a large amount of the variation in a data set using only a few features. PCA returns a series of principal components, these can be thought of as vectors in the original feature space. Principal components will be orthogonal to each other so the variance explained by each component will be unique. Given an $n \times p$ data set, n principal components can be created where each principal component is a vector of length p. Note that if all n principal components are used, the principal components will be in the same feature space as the original data. Due to this, we will usually want to select the first few principal components to use.

1.6 Principal Component Regression

Principal Component Regression (PCR) is a regression technique that uses principal components as predictors. To perform PCR, one first finds the desired number of principal components and then performs linear regression using the selected principal components as predictors.

For high dimensional data, PCR can avoid problems of over fitting that occur with a large number of predictors. Furthermore, it can provide more interpretable models if the principal components can be linked to latent variables. PCR does have some drawbacks, since principal components only depend on variance within predictors, PCR can miss predictors that contribute little to variance among predictors but strongly explain variance among the response variable.

1.7 Partial Least Squares

Partial Least Squares (PLS) is a regression technique with similarities to PCR. Unlike PCR, PLS considers the covariance matrix instead of the variance matrix. By using the covariance matrix, we ensure that the latent variables of each order will have the most possible correlation with the response variable. For example, the first principle component may only be able to explain most of the variance in the predictors but very little in the response whereas the first component in PLS will explain as much of the variance in the response as possible. This allows it to make more accurate predictions than PCR when using models of the same complexity.

1.8 L_1 Sparsification

L_1 sparsification is a common shrinkage method used to avoid over fitting models. This method is referred to as adding an L_1 penalty, as a new term is introduced that penalizes the model for being overly complex. This penalty is generated using the L_1 norm, commonly known as the absolute value. The added penalty encourages the model to only give high weight (usually in the term of coefficients) to terms that have high predictive power in the model. The weighting of terms with lower predictive power is lowered ro shrunk thus this is called a shrinkage method.

Unlike many other common shrinkage methods such as the L_2 norm, using the L_1 norm will also perform variable selection. This is due to the fact that the L_1 norm will sometimes send the weighting of terms to 0, completely ignoring them in the model. Since some terms are eliminated from the model, model built using the L_1 norm will often be simpler as they depend on fewer terms. This is useful when working with high dimensional data sets as removing predictors will help with the curse of dimensionality.

One difficulty with using the L_1 norm is that it introduces another parameter, λ into the model that must be selected for. λ is often referred to as the tuning parameter and decides how much model complexity should be penalized. Higher values of λ will more heavily penalize the model causing fewer terms to be included and more shrinkage. The tuning parameter is usually selected using cross validation.

1.9 Why Partial Least Squares?

Regression models that use latent variables can help with the curse of dimensionality when building models for high dimensional data. When $n < p$ (the number of predictors is larger than the sample size) or $n \approx p$ many traditional regression methods will lose predictive power as they become increasingly susceptible to noise in the data. For example, using single variate least squares regression when $n < p$ is extremely likely to over fit the data. Furthermore, solutions that minimize the squared error will no longer be unique.

Models that use latent variables will be able to avoid this problem by projecting the data into a lower dimensional feature space so that $n > p$. Since we project from the full feature space less information about the data will be lost than by simply performing variable selection.

Sparse partial least squares further mitigates problems of over fitting the data as it will perform variable shrinkage and selection. Features of the data that do not have much predictive power won't be included in the final model or their impact will be greatly reduced.

1.10 Literature Review

Partial Least Squares was first proposed by Herman Wold in 1975 under the name Non-linear Iterative Partial Least Squares (NIPALS) in order to model complex datasets (Wold, Sjöström, & Eriksson (2001)). Later models built using the framework of NIPALS would more simply be referred to by the name Partial Least Squares(PLS). Wold would later suggest that the model be referred to as Projection onto Latent Structures as he found this name to more accurately describe the model. Parameters for the model are found as the result of bivariate least squares linear regression using a latent variable believed to describe the underlying relation between predictor and response variables. These latent variables are then iterated over to estimate model parameters.

PLS models always contain one parameter, K, the number of latent variables included in the model. Some variants allow for manual selection of K while others use their own criteria to select the number of latent variables included in the model. Latent variables corresponding to higher values of K will explain less of the relationship between the predictors and response. Ideally K is low as this means that the relation between predictors and the response can be described using a small number of latent

variables.

PLS and variants have shown to be some of the best performing models for working with high dimensional data. This has lead to its widespread use in fields such as chemometrics, genomics, spectrometry, as well as others that work with data where the number of predictors is close to or greater than the number of observations.

PLS models generally follow a five step process as outlined by Lorenzo, Saracco, & Rodolphe (2010).

a. Estimate the covariance matrix between the predictor ($\mathbf{X}$) and response ($\mathbf{Y}$) variables.

b. Estimate the singular-space associated with the largest singular-value of the previous matrix.

c. Project the covariate and response parts on the previously defined subspace.

d. Estimate the linear regression matrix of the response part on the covariate part in the subspace.

e. Remove the information carried by the current subspace from the covariate and response parts

Although not a problem unique to PLS, model consistency is shown to decrease as the ratio of predictors to observations increases. This is due to models tendency to overfit the data as they are unable to distinguish between noise and the underlying relationships of variables. In order to improve on these problems of overfitting, L_1 penalization of regression coefficients was introduced by Tibshirani (1995) with the Lasso method. The Least Absolute Shrinkage and Selection Operator(Lasso) model is based on Ordinary Least Squares Regression(OLS) and introduces a penalty term to enforce sparsity among predictors.

The basic idea of penalization is to discourage models from including larger coefficients for terms that only explain a small amount of variance. This prevents models from overfitting data as the model will give less weight to terms that only explain noise in the data when trained on a sufficiently large sample. Unlike similar models with a penalty term such as Ridge Regression, Lasso also performs variable selection.

The Lasso model contains one parameter, λ, which takes positive real numbers as values and decide how heavily the model will be penalized. When 0 is chosen, the model is identical to partial least squares. As λ increases, the model will become increasingly

sparse. When $\lambda \to \infty$, the model will become $\hat{y} = \beta_0 = \bar{X}$ as all coefficients other than β_0 are penalized. Unlike other shrinkage methods, Lasso also performs variable selection as the L_1 norm will send coefficients to 0 for sufficiently large λ. This means that Lasso tends to select simpler more interpretable models than similar techniques. Parameter selection is usually performed through cross validation, a method originally proposed by Tibshirani.

Sparse partial least squares(SPLS) is a PLS method that uses the L_1-norm to penalize PLS models and thus avoid overfitting the data. SPLS was first formulated by Chun & Keles (2007) and later formally published in 2010 Chun & Keles (2010). Their formulation was heavily inspired by the recently proposed technique of sparse principal component analysis. Their model works by finding a surrogate vector close to the original direction vector and imposing sparsity through the L_1-norm on this vector. The SPLS model has four parameters, $(K, \kappa, \lambda_1, \lambda_2)$. K is the number of components, $\kappa \in [0, 1]$ and decides how far the starting surrogate vector can be from the original direction vector, λ_1 determines the severity of the L_1 penalty, and λ_2 determines the severity of the L_2 penalty.

Functionally, the SPLS only has two parameters that need to be tuned, K and λ_1. Values of $\kappa < \frac{1}{2}$ avoid local solution problems while offering little variation in the final model. Chun and Keles suggested that several values of $\kappa < \frac{1}{2}$ be tried but did not offer a formal proposal for selecting an ideal value. λ_2 is always tuned to ∞ in order to preserve the soft-thresholding solution. λ_1 and K are tuned using cross validation.

Referring to the 5 step process of PLS algorithms outline by Lorenzo, we can write the algorithm for SPLS such that only the second step(b) is different depending on our formulation.

Another sparse partial least squares(this time denoted as sPLS) algorithm was suggested by Lê Cao, Rossouw, Robert-Granié, & Besse (2008). The sPLS algorithm depends on initializing the the singular vector $\mathbf{v}$ and iterating until the relation between the two singular vectors, $\mathbf{u}$ and $\mathbf{v}$, converges. Each of these singular vectors has a corresponding parameter, $\lambda_{\mathbf{u}}$ and $\lambda_{\mathbf{v}}$. This leads to a total of $2K + 1$ parameters(note that K is one of the parameters) as there are two additional parameters that must be tuned for each component. We should note that sPLS only performs variable selection among predictors.

Data-driven sparse partial least squares(ddsPLS) is a new method introduced in 2021 by Lorenzo, Saracco, & Rodolphe (2010). Unlike previous sparse PLS methods, ddsPLS directly penalizes the empirical covariance matrix of the data. This is where the "data-driven" part of the name comes from the fact that the covariance matrix

is more closely related to the original data than the eigenvectors of the covariance matrix which are normally penalized.

Unlike other sparse PLS methods, ddsPLS only has $K + 1$ parameters, K and $\Lambda_K = \lambda_k, k = 1, ..., K$. Since the covariance matrix is directly penalized, we only need one tuning parameter per component with λ_k being the tuning parameter for the kth component. Each λ_k is tuned through bootstrapping, we can either select using $R^2 - Q^2$ or Q^2 metric with the former being recommended.

The use of bootstrapping for parameter selection in the ddsPLS model differs from practice for similar models. Lorenzo justifies the choice by saying that "use of bootstrap is interesting to enrich the underlying information of the available data." Bootstrapping was developed by Efron (1979) in 1979 and has become one of the most important techniques in statistics with the increased availability of high speed computers. Typically K-fold cross validation is used for parameter selection with K varying based on the sample size and other features of the data. In his study of model selection techniques, Kohavi (1995) recommended 10-fold stratified cross validation for model selection.

ddsPLS and other PLS models often use the Q^2 metric to assess model performance. Q^2 is closely related to the more common R^2 metric as both are calculated using the formula $1 - \frac{\text{RSS}}{\text{TSS}}$ where RSS is the residual sum of squares and TSS is the total sum of squares. R^2 is calculated using the data the model is built using while Q^2 is calculated using data that the model isn't built on. Typically this is done as part of cross validation with Q^2 calculated using the excluded data. The Q^2 metric was introduced by Stone (1974) while Tenenhaus (1998) first suggested it uses to assess PLS models.

Chapter 2

Theory

2.1 Principal Component Analysis

Due to the similarity between partial least squares and principal component regression, we will briefly review principal component analysis.

Principal component analysis(PCA) is a commonly used unsupervised learning technique. PCA works by finding the principle components of a dataset, these can be thought of as the direction along which the data varies the most in the feature space. For those of you with a background in linear algebra, the principal components will be the eigenvectors of the covariance matrix in order of the norm of the eigenvalues.

PCA is one of the most commonly used dimension reduction techniques as it is often able to capture a large amount of the variation in a data set using only a few features. PCA returns a series of principal components, these can be thought of as vectors in the original feature space. Principal components will be orthogonal to each other so the variance explained by each component will be unique. Given an $n \times p$ data set, n principal components can be created where each principal component is a vector of length p.

The algorithm for finding the first principal component of a standardized $n \times p$ data matrix $\mathbf{X}$ is as follows:

$$\mathbf{w}_1 = \text{argmax}_{||\mathbf{w}||=1}\left\{\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}\right\}$$

Note that $\mathbf{X}^T\mathbf{X}$ is the covariance matrix for $\mathbf{X}$. For all following components, we will find the matrix $\hat{\mathbf{X}}_k$ such that all variance explained by the first $k-1$ principal

components is removed. This is done by finding

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{j=1}^{k-1} \mathbf{X}\mathbf{w}_j\mathbf{w}_j^{\mathrm{T}}$$

We then use the same equation used to find $\mathbf{w}_1$ to find the kth principle component, replacing $\mathbf{X}$ with $\hat{\mathbf{X}}_k$.

The following plot shows the first two(and only) principals components for a set of two dimensional data. The red line is the first principal component while the green line is the second. Note that they are perpendicular. The red line describes as much variation in the data using a singular vector in the original space. Although it may look similar, the first principal component is not the same as the regression line which minimizes the residual distance.

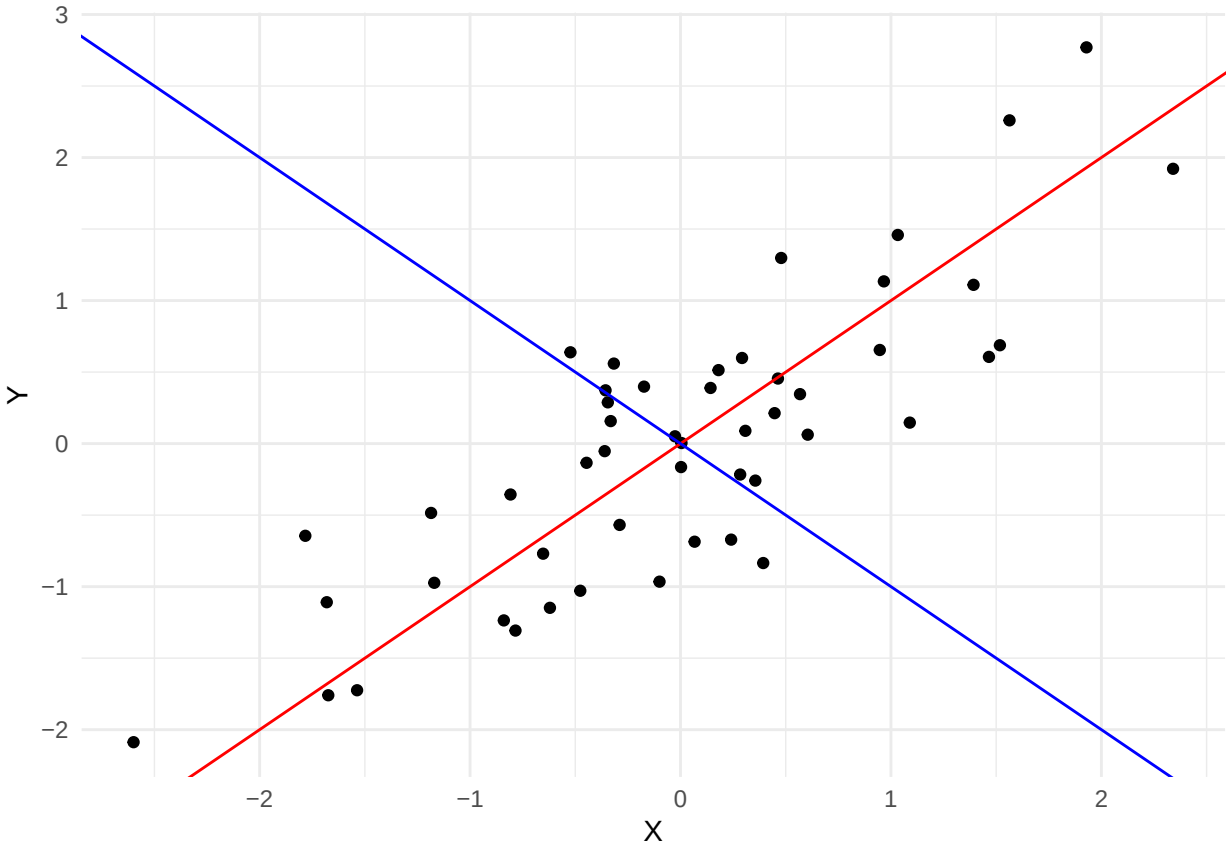

Figure 2.1: The red line shows the first principal component of the data while the blue line shows the second principal component.

Often principal components can be used as latent variables. Sometimes they may be meaningful and correspond to a property of the data that isn't directly measured. For example, the first principal component of data containing the nutritional values

for food may correspond to how rich in vitamins a food is.

Principal Component Regression (PCR) is a regression technique that uses principal components as predictors. To perform PCR, one first finds the desired number of principal components and then performs linear regression using the selected principal components as predictors. Letting $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_k$ be the first k principal components of our predictors $\mathbf{X}$. This will take the form

$$\mathbf{y} = \theta_0 + \sum_{i=1}^{k} \theta_i \mathbf{w}_i$$

2.2 L1 Sparsification

The L_1 penalty can be added to a model by adding a term of $\lambda \sum_{i=1}^{p} |\beta_i|$ where λ is a parameter for the model. For linear regression, coefficients for the model will then be found by finding $\{\beta_0, ..., \beta_n\}$ such that the following is minimized

$$\text{argmin} \left\{ \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

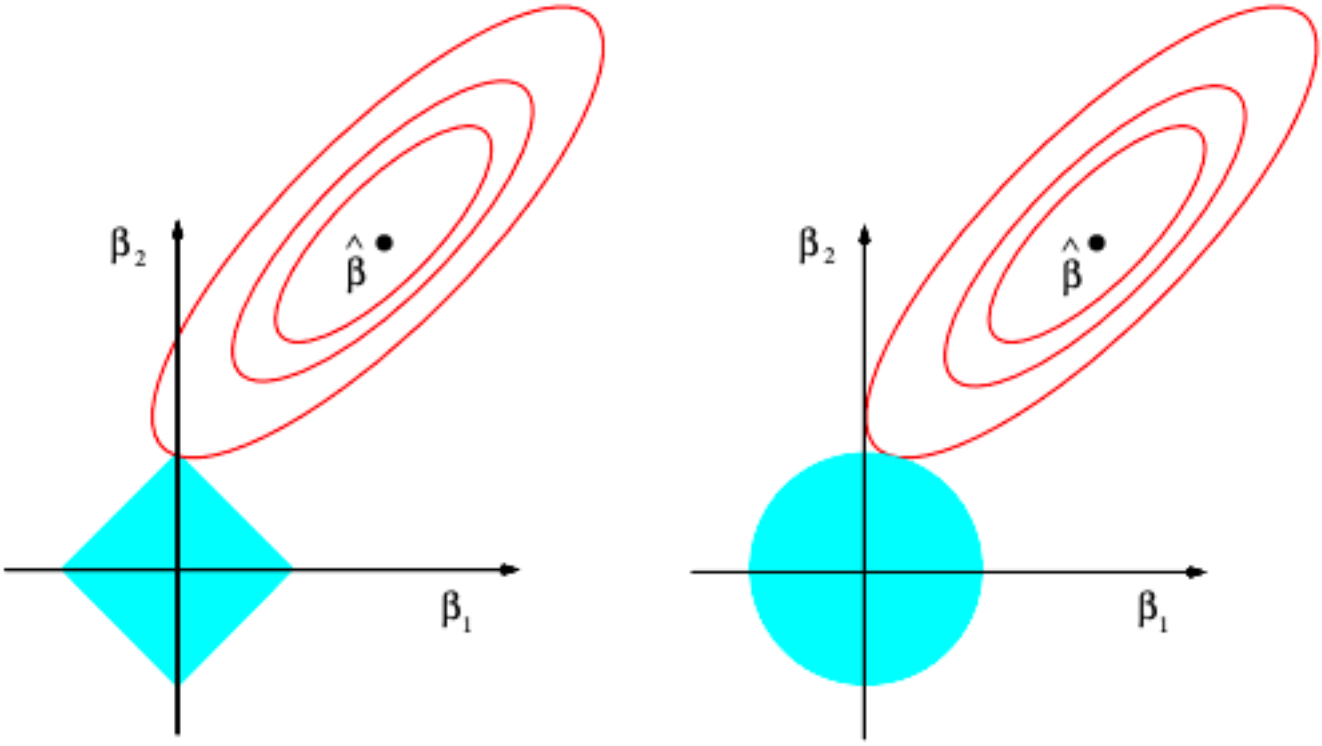

Figure 2.2: Comparison of L_1(left) and L_2(right) norms. As we can see restraints on the L_2 norms prevent coefficients from being sent to 0. Image taken from ESLR Hastie, Tibshirani, and Friedman (2009).

Note that as $\lambda \to 0$ the model will become unchanged by the penalty and as $\lambda \to \infty$, $\beta_i = 0$ for all i. If λ is sufficiently large, we will begin to see some $\beta_i = 0$ which makes the L_1 norm special among commonly used norms.

2.3 Latent Variable Models

Latent variable models such as PCR and PLS are built on the assumption that there exists a structure linking the predictors and response variables that isn't directly captured in the data. There are two ways we can try and conceptualize these latent variables, either as being derived from the observed data or being used to generate the observed data. In this case, it is most helpful to think of latent variables as generating the observed data.

We will assume that $\mathbf{X} = \phi\mathbf{A} + \epsilon_X$ and $\mathbf{Y} = \phi\mathbf{D} + \epsilon_Y$. Here ϕ is a matrix of latent variables linking $\mathbf{X}$ and $\mathbf{Y}$. $\mathbf{A}$ and $\mathbf{D}$ are the transformation applied to ϕ to generate $\mathbf{X}$ and $\mathbf{Y}$. ϵ_X and ϵ_Y are random noise.

$\mathbf{X} \in \mathbb{R}_{n\times p}$ and $\mathbf{Y} \in \mathbb{R}_{n\times q}$. Thus, $\phi \in \mathbb{R}_{n\times \mathcal{K}}$, $\mathbf{A} \in \mathbb{R}_{\mathcal{K}\times p}$,and $\mathbf{D} \in \mathbb{R}_{\mathcal{K}\times q}$. Note that

$\mathcal{K} \neq K$ where K is the number of components that will be used in a PLS model.

Since all possible variance is explained by ϕ, $\mathbf{A}$, and $\mathbf{D}$ we will find that $\text{Cov}(\phi, \epsilon_X) = \text{Cov}(\phi, \epsilon_Y) = \text{Cov}(\epsilon_X, \epsilon_Y) = 0$. Furthermore, $\text{Var}(\mathbf{X}) = \mathbf{A}\mathbf{A}^{\text{T}} + \text{Var}(\epsilon_X)$ and $\text{Var}(\mathbf{Y}) = \mathbf{D}\mathbf{D}^{\text{T}} + \text{Var}(\epsilon_Y)$. Finally, $\text{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbf{D}^{\text{T}}\mathbf{A}$.

It may seem intuitive to build a model of the form $\mathbf{Y} = \mathbf{B}\mathbf{X} + \epsilon$ where $\mathbf{B}$ satisfies $\mathbf{A}\mathbf{B} = \mathbf{D}$. This only provides a unique answer under certain circumstances. Instead we want to formulate a model that is more similar to how the data is structured Thus, our model will be of the form $\mathbf{X} = \mathbf{t}\mathbf{P} + \epsilon_X$ and $\mathbf{Y} = \mathbf{t}\mathbf{C} + \epsilon_Y$.

2.4 Evaluating Model Performance

While building models is a difficult task selecting the correct model to use may be even more difficult. We want to find a model that fits our original data well without overfitting it. Unlike when building a model, there is usually more than one criteria that we want our model to perform well with. Ultimately, model selection often ends up being a subjective process dependent on a variety of factors. While there are standard measures designed to evaluate the performance of a model, I advise against putting too much faith in any singular statistic.

One of the most widely used statistics to evaluate model performance is the root mean squared error(RMSE). The RMSE is calculated using the formula

$$RMSE = \sqrt{\frac{\sum_{j=1}^{q}\sum_{i=1}^{n}(\hat{y}_{i,j} - y_{i,j})^2}{n}}$$

where $\hat{y}_{i,j}$ is the predicted value of a response variable. Here we want a low RMSE as this means that the model tends to estimate values close to the true value.

Another popular statistic used is R^2. Simply, $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$ where RSS is the real sum of squares and TSS is the total sum of squares. The RSS is calculated using the equation $\text{RSS} = \sum_{j=1}^{q}\sum_{i=1}^{n}(\hat{y}_{i,j} - y_{i,j})^2$ while the TSS is calculated using $\text{RSS} = \sum_{j=1}^{q}\sum_{i=1}^{n}(\bar{y}_{i,j} - y_{i,j})^2$ where $\bar{y}_{i,j}$ is the empirical mean of the observed data. The idea of R^2 is to evaluate how a model performs compared to using mean estimation. Mean estimation is perhaps the most crude form of regression available, if our model does not perform well when using the R^2 metric, it is almost always a sign that we should not use the model. Values of R^2 usually range between 0 and 1. Values less than 0 mean that the model performs worse than mean estimation. Values equal to 0 mean the two perform the same (usually this mean the model is just performing mean estimation). Values between 0 and 1 mean that the model performs better than

mean estimation with larger values signifying better performance compared to mean estimation. If $R^2 = 1$ this means that the model is able to perfectly estimate every response value in the data.

Before moving onto the next metric now is a good time to discuss test-training splits, cross validation(sometimes notated as CV), and bootstrapping.. All of these methods are used to try and answer questions about how the model will perform on data that is not used to train the model. Common to all of these methods is the idea of in-bag(IB) and out-of-bag(OOB) observations. IB observations are the data that are used to build our model while OOB observations are the data that the model is not built with that can be used to test it. All of the following methods are based on some permutation of this idea of creating IB and OOB groupings of the data.

Test-training splits are probably the most basic of the three methods relying on splitting our data into two groups, the training group and the test group. Here the IB group will be the training data and the OOB group will be the test data, these groups will not change during the process. To create the test and training sets we randomly split our data into two groups. Usually a smaller percentage of the data is relegated to the training set (e.g. 30% is assigned to the training set while the remaining 70% will be the test set). The model will then be built using the training data and assessed using the test data. This will help tend to help with overfitting as it will make sure the model can perform well on data that isn't used to build it. This still makes the assumption that our larger sample mirrors the general population that we hope to use the model with.

Cross validation is one of the most common tools for model assessment and parameter tuning. Cross validation involves splitting the data into test and training sets multiple times and then assessing model performance in each instant. It can be helpful to think of cross validation as a repeated use of test-training splits. There are two commonly used types of cross validation, K-fold CV and Leave-one-out CV(LOOCV). K-fold CV randomly divides the data into K equal sized groups and then uses $K - 1$ of the groups as the training set and the remaining one as the test set. This is repeated K times until all groups have been used as the test set. LOOCV is technically a form of K-fold where K is equal to the total number of observations. Thus, the test set will only consist of 1 observation.

CV is often used to select parameters, as we will build models with a number of parameters and then assess its performance across parameters using cross validation. The choice of statistic used to assess parameter performance often varies depending on the model we are building although R^2 is a common choice. One of the problems with

this approach is that it is computationally expensive as it requires the model to be built across a range of parameters. Due to this, when working with a large number of observations, it is generally advisable to use K-fold CV with a smaller number of folds (10 is often recommended) as this will require many fewer computations the LOOCV.

Bootstrapping relies on randomly sampling with replacement from our data to build the test and training sets. When bootstrapping we will expect to see repeated values in the training set (no matter the size of our dataset we expect to see 63.2% in our bootstrap sample). The observations not included in the bootstrap sample will then be the test set.

Following this discussion of IB and OOB observations and methods of generating them we will discuss the Q^2 statistic. Q^2 is very similar to the R^2 statistic with the only difference being that it is exclusively calculated using OOB observations. The Q^2 statistic is only commonly used when discussing PLS models. There is often some ambiguity as to what data is used to calculate the R^2 statistic. It can sometimes be calculated on the training set to help built the model or on the test set to evaluate model performance. Differentiating R^2 and Q^2 solves this ambiguity. R^2 for a test-training split b is found with the equation

$$R_b^2 = 1 - \frac{\sum_{j=1}^{q} \sum_{i \in IB(b)} (\hat{y}_{i,j}^b - y_{i,j})^2}{\sum_{j=1}^{q} \sum_{i \in IB(b)} (\bar{y}_{i,j}^b - y_{i,j})^2}$$

Q^2 for the split b is calculated as follows

$$Q_b^2 = 1 - \frac{\sum_{j=1}^{q} \sum_{i \in OOB(b)} (\hat{y}_{i,j}^b - y_{i,j})^2}{\sum_{j=1}^{q} \sum_{i \in OOB(b)} (\bar{y}_{i,j}^b - y_{i,j})^2}$$

where $\hat{y}_{i,j}^b$ are predictions from the model built using the training group b and $\bar{y}_{i,j}^b$ are the mean estimates for rows of the training group b. We will then take the means across a B different splits so that $\bar{R}_B^2 = \frac{1}{B} \sum_{b=1}^{B} R_b^2$ and $\bar{Q}_B^2 = \frac{1}{B} \sum_{b=1}^{B} Q_b^2$.

Despite the similarity of the R^2 and Q^2 metrics, the two metrics will improve under different conditions for the model. Maximizing R^2 will favor more complex models that closely fit the training data while maximizing Q^2 will favor less complex models that address the underlying structure of the data. Initially, increasing model complexity will improve both metrics. However, past a certain threshold, increasing model complexity will cause Q^2 to decrease while R^2 to continues to increase. Due to this, minimizing $R^2 - Q^2$ is recommended for selecting values of λ_k for the ddsPLS model.

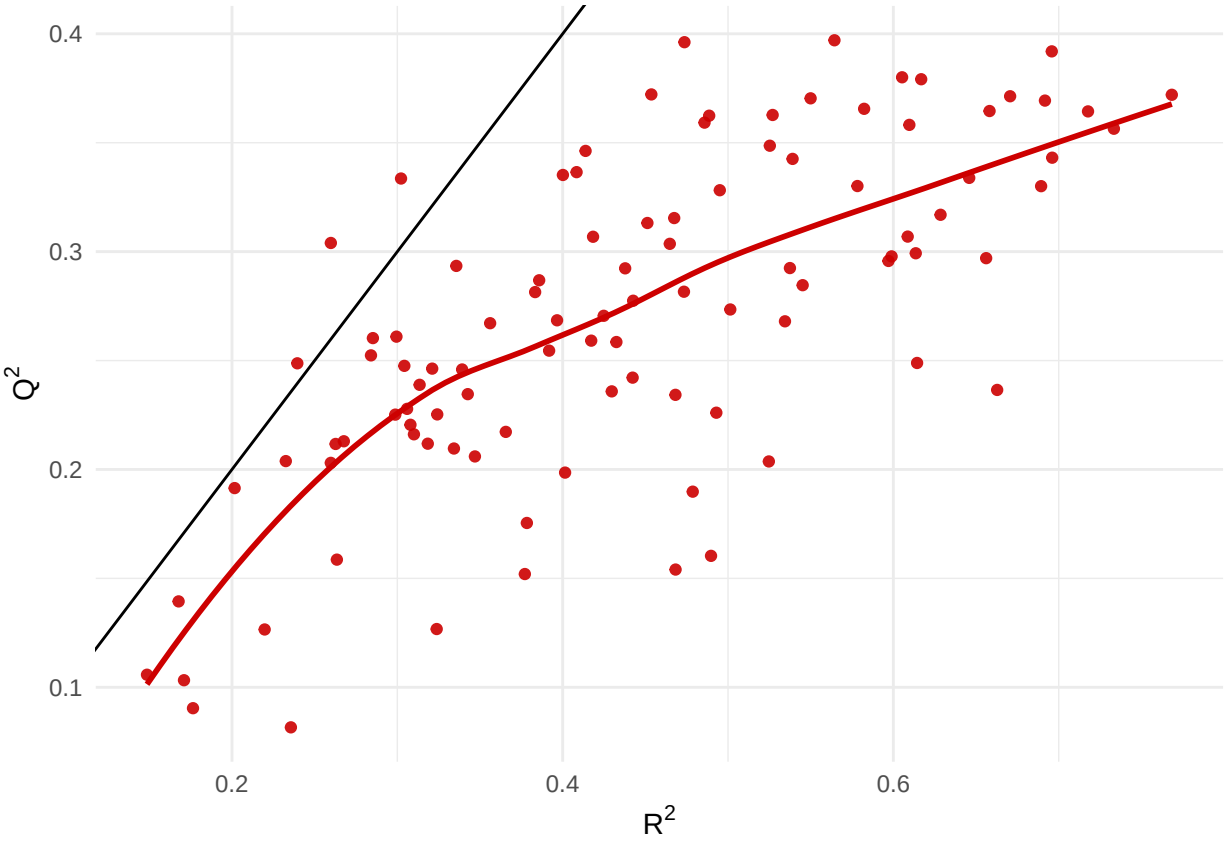

Figure 2.3: Relation between R^2 and Q^2 for the performance of a model.

Ideally we want a high R^2 and Q^2 meaning the model performs well with both the test and training data. In practice R^2 is almost always an inflated metric of performance and Q^2 will be more accurate for accessing model performance. For this reason minimizing $R^2 - Q^2$ may be used for parameter selection as low values indicate models that perform well on both IB and OOB observations. Although it may seem likely to pick parameters where both R^2 and Q^2 perform poorly, R^2 tends to increase more quickly and fall of more slowly than Q^2 leaving a difference at these extremes.

2.5 Partial Least Squares

Partial Least Squares(PLS) is a regression technique similar to PCR. Both techniques work by finding latent variables in the original data with which to perform linear regression. PLS is considered to be one of the best tools for modeling the underlying structure between $\mathbf{X}$ and $\mathbf{Y}$ in feature space. As suggested its creator, the name "Projection onto Latent Structures" is a more descriptive acronym as the original data is projected onto latent structures which are then used for linear regression.

Unlike PCR, PLS creates latent variables using the covariance matrix instead of the variance matrix. Implicit in PCR is the assumption that direction of high variance among the predictors explains much of the variance among the response variables. While this assumption is usually reasonable, PLS circumvents this assumption by directly building the latent variables along the direction in feature space along where the responses vary the most along the predictors.

PLS is designed to model situations where there are a high number of predictors in relation to the number of observations($n \approx p/n < p$). In these situation, most models will struggle to differentiate noise from signal and will be highly variable depending on the data used to train them. PLS circumvents this problem by simplifying the data into a smaller number of latent variables that still strongly reflect the original data set. Unlike models that depend solely on variable selection, PLS will not discard relevant information from all observed variables in order to fit a model.

PLS is also designed to perform well in situations where predictors are highly correlated and noisy. The data is projected onto underlying structures in order to handle correlation between predictors. Correlated predictors can often be better explained along a singular vector in feature space that captures all correlated predictors. This correlation will be captured in the latent structures that the data is project onto. These underlying structures captured in the latent variables will also be less sensitive to noise among the predictors.

As previously mentioned, PLS models often follow a five step process outline by Lorenzo. To briefly review, these steps are a) estimate the covariance matrix, b) find the space associated with the first eigenvector, c) project the covariate and response matrices into the space from the previous step, d) perform linear regression between the projection of the covariate and response matrices, and e) remove the predicted information.

Like PCA, PLS also require all variables to be standardized with $\mu = 0$ and $\sigma^2 = 1$. We then find the mth latent variable $\mathbf{z}_m$ using the following formula

$$\mathbf{z}_m = \sum_{i=1}^{p} \hat{\varphi}_{mi} \mathbf{x}_j^{(m-1)}$$

where $\hat{\varphi}_{mi} = \langle \mathbf{x}_j^{(m-1)}, \mathbf{y} \rangle$.

We then find the regression coefficient $\hat{\theta}_m$ for $\mathbf{z}_m$ as follows

$$\hat{\theta}_m = \frac{\langle \mathbf{z}_m, \mathbf{y} \rangle}{\langle \mathbf{z}_m, \mathbf{z}_m \rangle}$$

Note that this is identical to the formula used in classic linear regression and PCR at this step.

2.5.1 NIPALS-PLS

The NIPALS-PLS algorithm (sometimes referred to as the PLS2 algorithm) is the first PLS model suggested by Wold and still one of the more commonly used PLS models. Similar basic PLS models almost exclusively differ in the method used to calculate the covariance matrix, especially when dealing with missing data.

Before going into detail on the model it is important to note that all PLS algorithms require the data to be centered and standardized in order to prevent variables with different scales from exhibiting an outsize impact on the final outcome.

We will assume that $\mathbf{M} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}$ will be used to estimate the covariance matrix unless otherwise noted. This estimate, $\mathbf{M}$ is the empirical covariance matrix calculated using the observed data. Estimating $\mathbf{M}$ is the most sensitive step as this estimation is sensitive to the curse of dimensionality.

The NIPALS-PLS algorithm is as follows:

For $k \in [1 : K]$,

a) $\mathbf{u}_k = \overrightarrow{\mathrm{RSV}}(\mathbf{M}_k)$

b) $\mathbf{t}_k = \mathbf{X}_k\mathbf{u}_k$

c) $\mathbf{p}_k = \frac{\mathbf{X}_k^{\mathrm{T}}\mathbf{t}_k}{\mathbf{t}_k^{\mathrm{T}}\mathbf{t}_k}$

d) $\mathbf{c}_k = \frac{\mathbf{Y}_k^{\mathrm{T}}\mathbf{t}_k}{\mathbf{t}_k^{\mathrm{T}}\mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k\mathbf{p}_k^{\mathrm{T}}$, $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k\mathbf{c}_k^{\mathrm{T}}$

Walking through these steps; a) finds the vector that explains the most variance in the covariance matrix for the remaining data, b) estimates the score, c) and d) estimate the linear regression matrices for $\mathbf{X}$ and $\mathbf{Y}$, and e) deflates the data.

The NIPALS-PLS algorithm maximizes $\mathbf{v}^{\mathrm{T}}\mathbf{M}_k\mathbf{u}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. The only parameter that needs tuning is K, the number of components used in the model.

2.5.2 sPLS

Here we will discuss a sparse PlS algorithm outlined by Le Cao et.al. It is important to note that there exist other sparse PLS algorithms most notably the one created by

Chun and Keles.

First we must define the soft-thresholding function $S_\lambda(\mathbf{M}) = \text{argmin}\left(||\mathbf{M} - \mathbf{\Sigma}||^2 + 2\lambda|\mathbf{\Sigma}|\right)$ with $\mathbf{\Sigma} \in \mathbb{R}^{q\times p}$ and where $\mathbf{M}$ is again our estimate of the covariance matrix.

The sPLS algorithm is as follows:

For $k \in [1:K]$,

a†) Start with initial $\mathbf{v}_k$ and iterate until the two steps converge.

i†) $\mathbf{u}_k = S_{\lambda_u^k}(\mathbf{M}_k^{\mathrm{T}}\mathbf{v}_k)$, $\mathbf{u}_k = \frac{\mathbf{u}_k}{||\mathbf{u}_k||}$

ii†) $\mathbf{v}_k = S_{\lambda_v^k}(\mathbf{M}_k^{\mathrm{T}}\mathbf{u}_k)$, $\mathbf{v}_k = \frac{\mathbf{v}_k}{||\mathbf{v}_k||}$

b) $\mathbf{t}_k = \mathbf{X}_k\mathbf{u}_k$

c) $\mathbf{p}_k = \frac{\mathbf{X}_k^{\mathrm{T}}\mathbf{t}_k}{\mathbf{t}_k^{\mathrm{T}}\mathbf{t}_k}$

d) $\mathbf{c}_k = \frac{\mathbf{Y}_k^{\mathrm{T}}\mathbf{t}_k}{\mathbf{t}_k^{\mathrm{T}}\mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k\mathbf{p}_k^{\mathrm{T}}$, $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k\mathbf{c}_k^{\mathrm{T}}$

Notice that step a) is the only step that differs from the NIPALS-PLS algorithm. This is where sparsity is imposed upon $\mathbf{u}$ and $\mathbf{v}$ which are used to calculate the score and loadings. Note that the sparsity is only directly imposed on $\mathbf{X}$.

The algorithm minimizes $||(n-1)\mathbf{M}_k - \mathbf{v}\mathbf{u}^{\mathrm{T}}||^2 + \lambda_u^k|\mathbf{u}| + \lambda_v^k|\mathbf{v}|$ for $\mathbf{u}$ and $\mathbf{v}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. sPLS has $2K+1$ parameters with two tuning parameters for each component and then the number of components.

2.5.3 Data-driven Sparse Partial Least Squares

Data-driven sparse partial least squares(ddsPLS) is a new PLS method recently proposed by Lorenzo. Unlike previous sparse partial least squares algorithms, ddsPLS directly imposes sparsity on the empirical covariance matrix instead of its eigenvector decomposition.

For $k \in [1:K]$,

a) $\mathbf{u}_k = \overrightarrow{RSV}(S_{\lambda_k}(M_k))$, $\mathbf{v}_r = \overrightarrow{RSV}(S_{\lambda^{(r)}}(M^{(r)^T}))$

b) $\mathbf{t}_k = \mathbf{X}_k\mathbf{u}_k$

c) $\mathbf{p}_k = \frac{{\mathbf{X}_k}^T\mathbf{t}_k}{{\mathbf{t}_k}^T\mathbf{t}_k}$

d) $\Pi_k = \text{diag}\left(\{\delta_{\neq 0}(\mathbf{v}_k)_j\}_{j\in[1:q]}\right)$ $\mathbf{c}_k = \text{argmin}||\mathbf{Y}_k\Pi_r - \mathbf{t}_k\mathbf{V}^T||^2 = \frac{(\mathbf{Y}_k\Pi_k)^T\mathbf{t}_k}{{\mathbf{t}_k}^T\mathbf{t}_k}$

e) $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k\mathbf{p}_k^T$ $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \mathbf{t}_k\mathbf{c}_k^T$

Note that if $\lambda = 0$ the outcome is identical to the NIPALS-PLS algorithm.

ddsPLS maximizes $\mathbf{v}^{\mathrm{T}} S_{\lambda_k}(\mathbf{M}_k^{\mathrm{T}})\mathbf{u}$ for $\mathbf{u}$ and $\mathbf{v}$ with the constraint $\mathbf{v}^{\mathrm{T}}\mathbf{v} = \mathbf{u}\mathbf{u}^{\mathrm{T}} = 1$. The ddsPLS algorithm has $K + 1$ parameters as there is only one parameter that needs to be maximized for each component. This gives it a better computation time when compared to other sparse PLS algorithms as cross validation or bootstrapping only needs to be performed for one parameter per component.

Along with the ddsPLS algorithm, Lorenzo also proposes a standard method for selecting the number of components to include in the model. If the following 3 conditions are met, the kth component is built. If the conditions are not built, the model will include only $k - 1$ components. The conditions are as follows,

1. The kth component performs better than mean estimation, $\bar{Q}^2_{B,k} > 0$.

2. The model containing K components performs than the model containing $K - 1$ components, $\bar{Q}^2_{B,K} > \bar{Q}^2_{B,K-1}$.

3. The bootstrapped explained variance is close to the boot-strapped cross-validated explained variance, $\bar{R}^2_B - \bar{Q}^2_B$ is minimum on the set of hyperparameters.

Chapter 3

Results

This chapter discusses the results of a variety of simulations run comparing the three model types discussed in the Theory chapter. If you are not already familiar with the models being discussed, I recommend that you read the the theory chapter first to get a better idea of the differences between the three model types. Throughout this chapter, the NIPALS-PLS algorithm is referred to as PLS for the sake of simplicity.

This chapter is broken into five sections: Data Simulation, Noise, Predictors, Responses and Parameter Selection. [Data Simulation][datasim] discusses the method used to simulate data for trials in the rest of the section and why these methods were chosen. Noise discusses how model performance changes as the amount of noise included in simulated data changes. Predictors discusses how model performance changes as the number and structure of predictors changes. Responses discusses how model performance changes as the number and structure of predictors changes. [Parameter Selection][parameters] discusses different metrics used to select parameters for ddsPLS and the differences between metrics. The goal of this chapter is to compare the performance of ddsPLS to existing PLS models and to determine under what conditions it performs well.

3.1 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). We want to make sure columns of $\boldsymbol{\phi}$ are uncorrelated in order to ensure that $\mathbf{A}$ and $\mathbf{D}$ have the structure that we intend. If columns of $\boldsymbol{\phi}$ exhibited varying amounts of correlation, we would not be able to know the number of components that the model should build. $\boldsymbol{\epsilon}_X$ 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 noise that appears correlated.

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 & Almøy (1994).

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,

$$\mathbf{A} = \begin{pmatrix} \mathbf{1}_{3\times5} & \mathbf{0}_{3\times10} & \mathbf{0}_{3\times5} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times2} & \mathbf{0}_{3\times(p-23)} \\ \mathbf{0}_{2\times5} & \mathbf{1}_{2\times10} & \mathbf{0}_{2\times5} & \mathbf{0}_{2\times1} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times(p-23)} \\ \mathbf{0}_{3\times5} & \mathbf{0}_{3\times10} & \mathbf{1}_{3\times5} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times2} & \mathbf{0}_{3\times(p-23)} \\ \mathbf{0}_{2\times5} & \mathbf{0}_{2\times10} & \mathbf{0}_{2\times5} & \mathbf{1}_{2\times1} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times(p-23)} \\ \mathbf{0}_{3\times5} & \mathbf{0}_{3\times10} & \mathbf{0}_{3\times5} & \mathbf{0}_{3\times1} & \mathbf{1}_{3\times2} & \mathbf{0}_{3\times(p-23)} \end{pmatrix}$$

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

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{R\times1} & \mathbf{0}_{R\times(q-1)} \end{pmatrix}$$

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

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times(q-4)} \\ \mathbf{0}_{3\times1} & \mathbf{1}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times(q-4)} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{1}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times(q-4)} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{1}_{3\times1} & \mathbf{0}_{3\times(q-4)} \\ \mathbf{0}_{1\times1} & \mathbf{0}_{1\times1} & \mathbf{0}_{1\times1} & \mathbf{0}_{1\times1} & \mathbf{0}_{1\times(q-4)} \end{pmatrix}$$

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.

3.2 Noise

When examining model performance, it is important to consider how it performs under conditions where the amount of noise varies. This section examines the performance of ddsPLS, sPLS, and PLS as ω, the term that weights the amount of random noise applied to the predictors and response variables, varies. For higher values of ω, there is more noise and we expect worse performance by the models.

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.

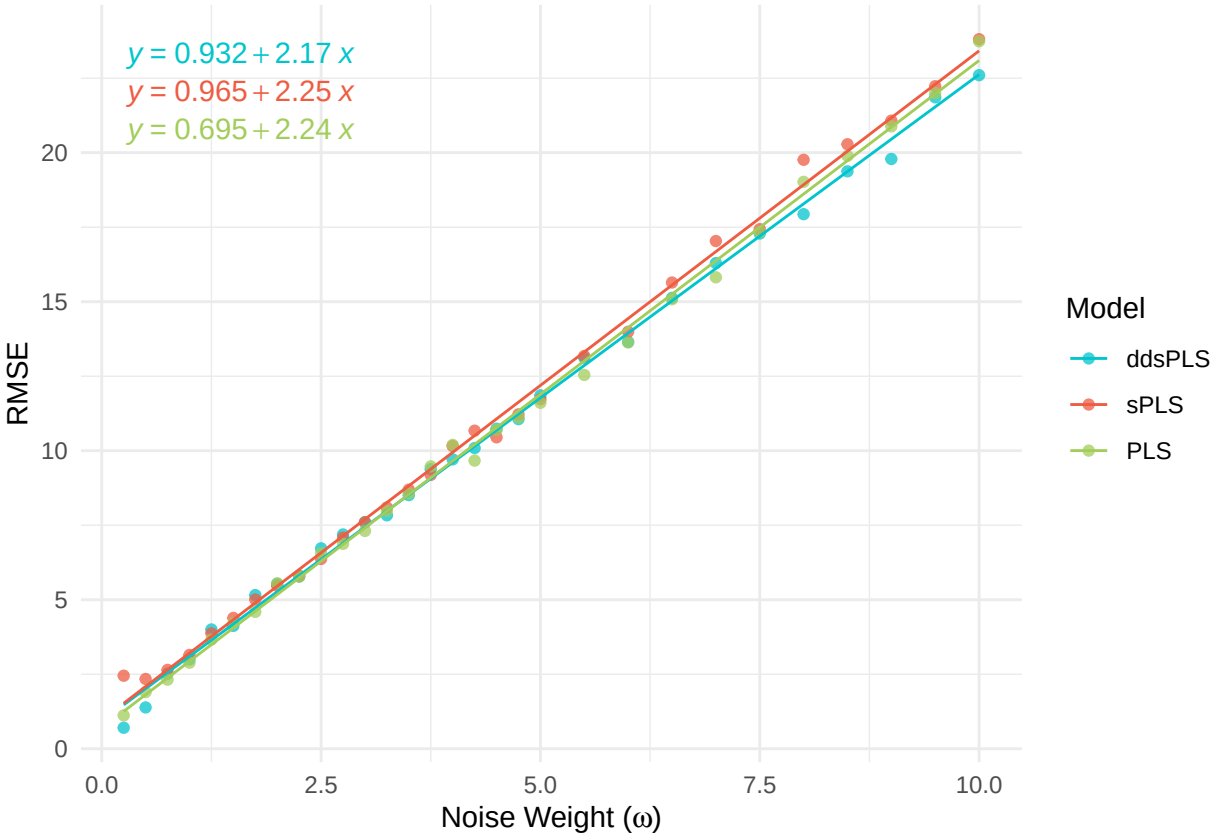

Figure 3.1: Relation between ω and the test set RMSE as ω increases for all model types. $n = 100, p = 100, q = 5$

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.

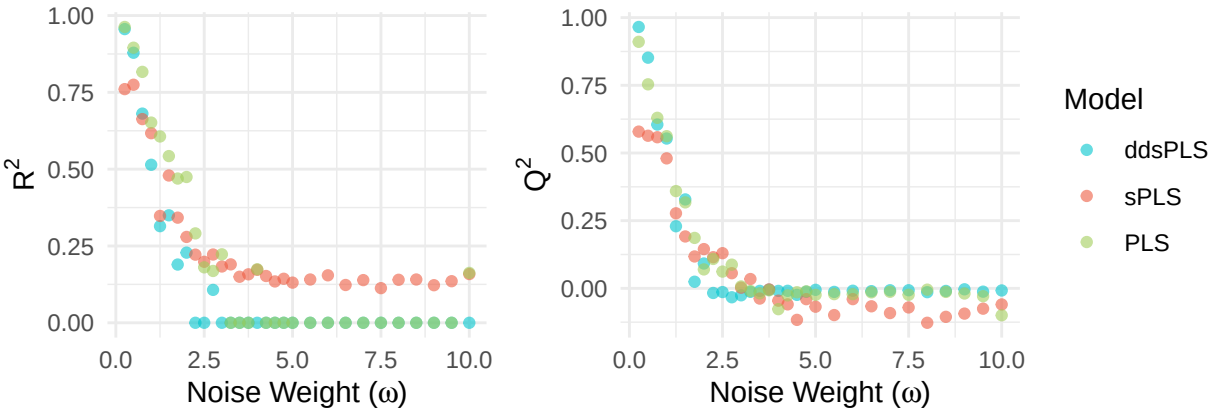

Figure 3.2: As ω increases, both R^2 and Q^2 decrease at a similar rates. Note that ddsPLS and PLS begin performing mean estimation at around $\omega = 2.5$.

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. For $\omega > 3$, the noise drowns out the signal in the model and all methods begin to perform as well as mean estimation at best. ddsPLS begins to perform mean estimation slightly earlier around $\omega = 2.5$.

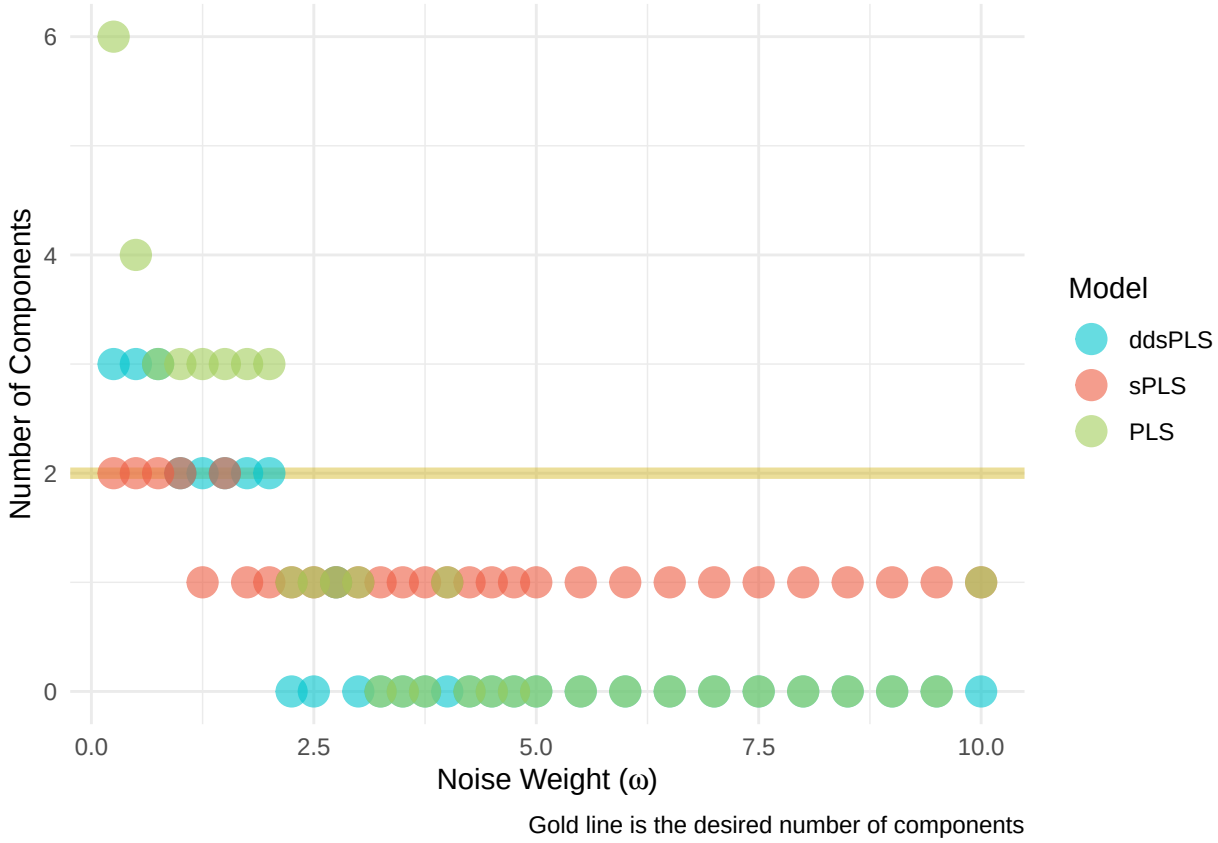

Figure 3.3: Relation between ω and the number of components built. As ω increases, all model types tend to build fewer components.

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 predictive models to be built.

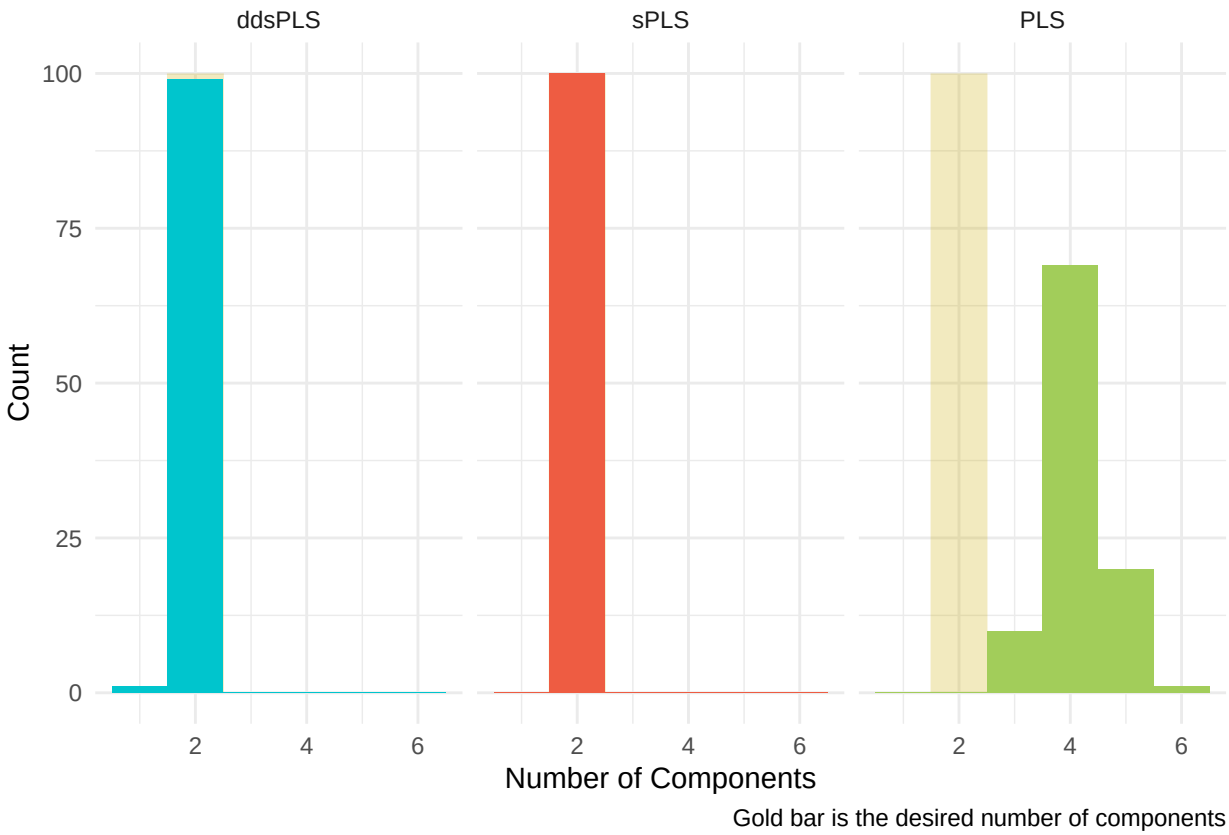

Figure 3.4: Distribution of components built across 100 simulations by model type when $\omega = 0.5$.

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. In this case, PLS always builds too many components.

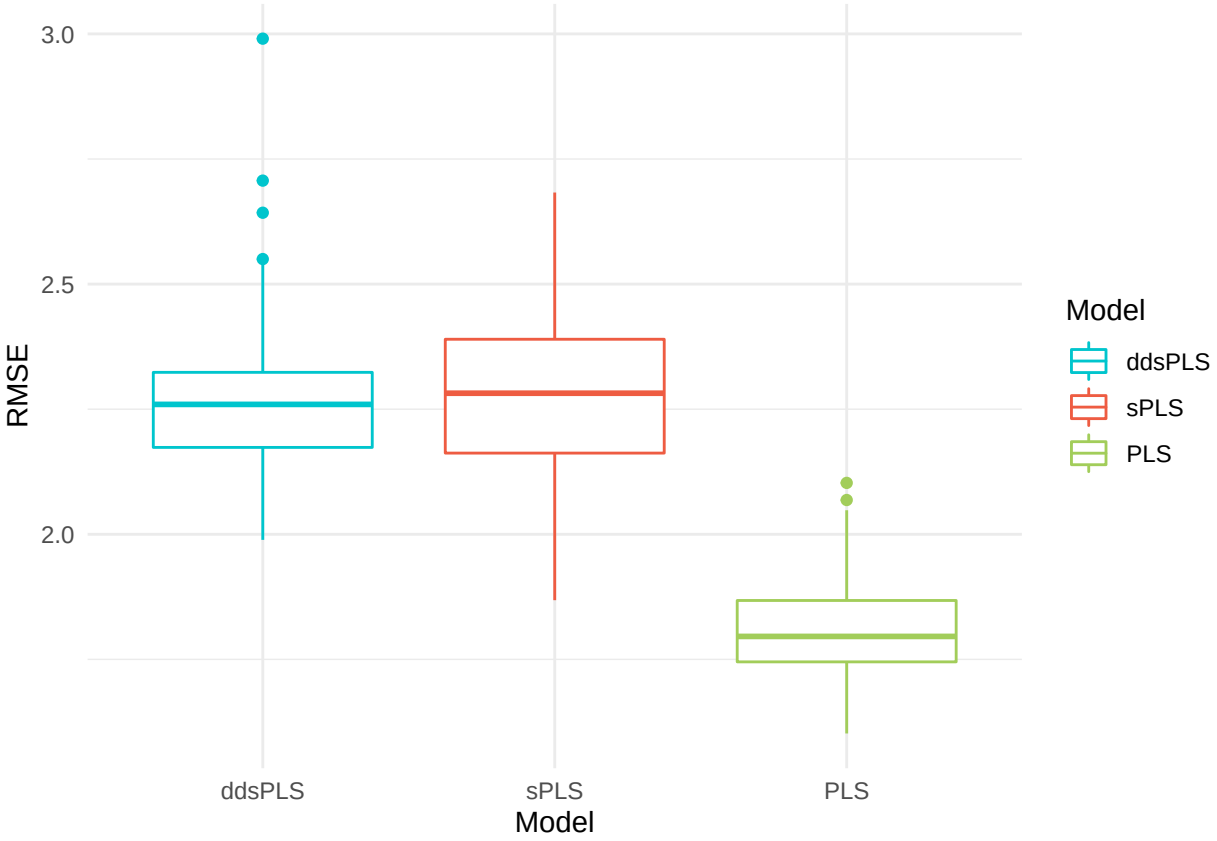

Figure 3.5: At $\omega = 0.5$, we can see that ddsPLS and sPLS perform similarly in regards to to RMSE while PLS performs significantly better.

Now let $\omega = 1$.

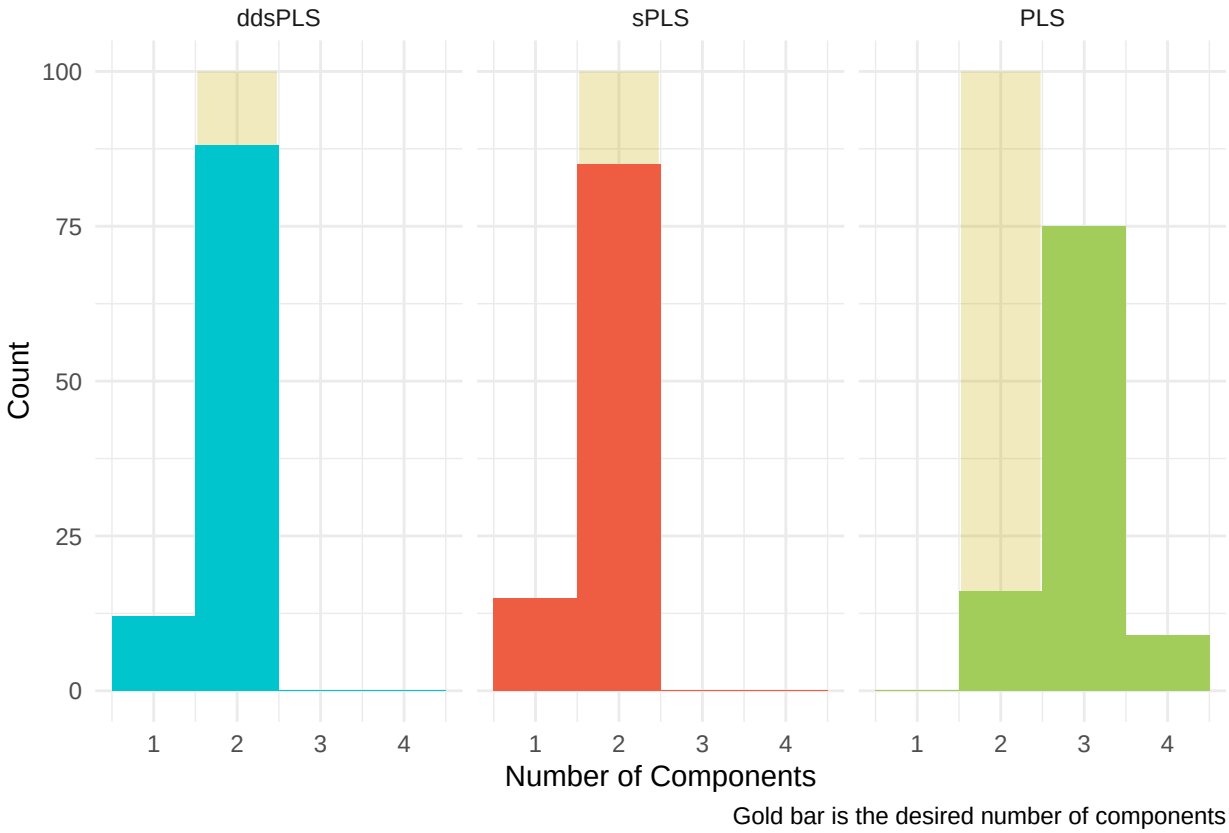

Figure 3.6: Distribution of components built across 100 simulations by model type when $\omega = 1$.

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.

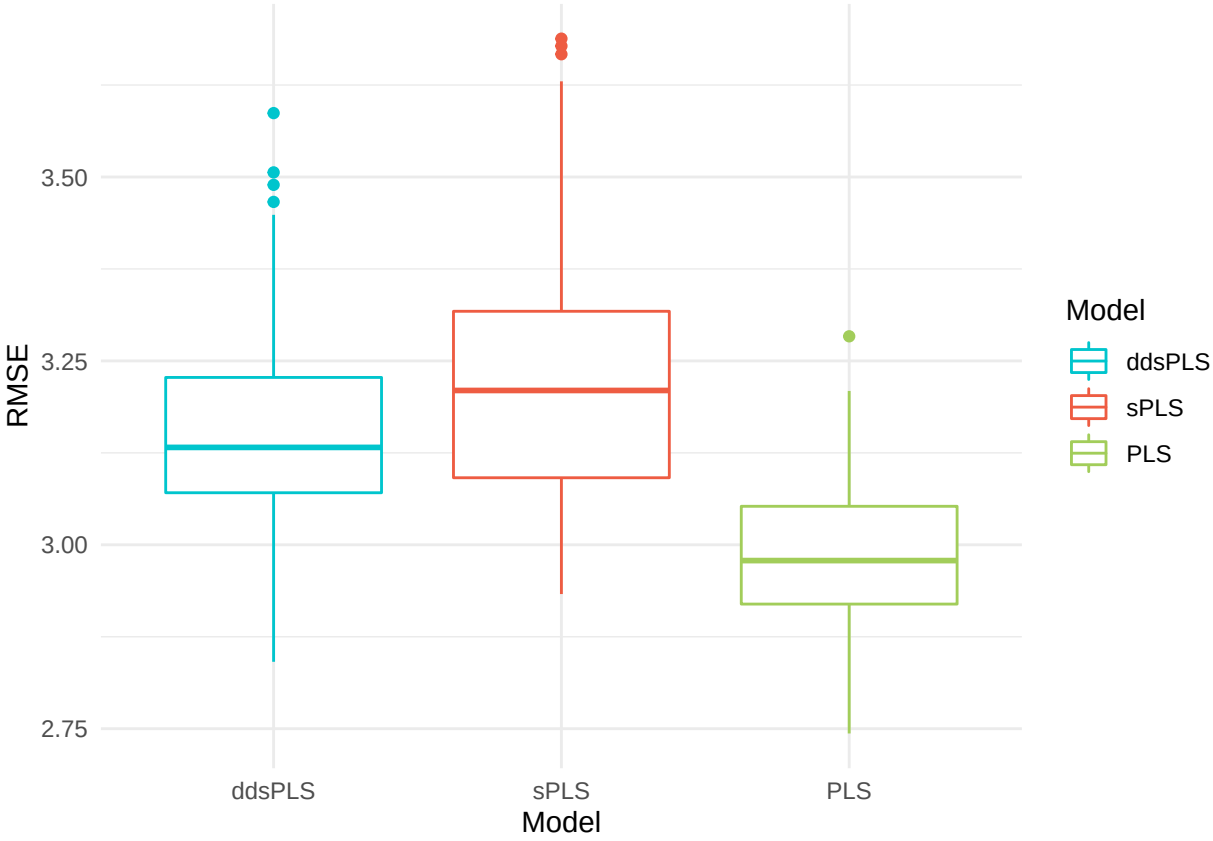

Figure 3.7: Distribution of RMSE for each model type when $\omega = 1$.

Again PLS tends to have the lowest RMSE however it's performance is now more similar to the other models than with less noise. ddsPLS and sPLS still perform very similarly however we can see that ddsPLS now performs slightly better.

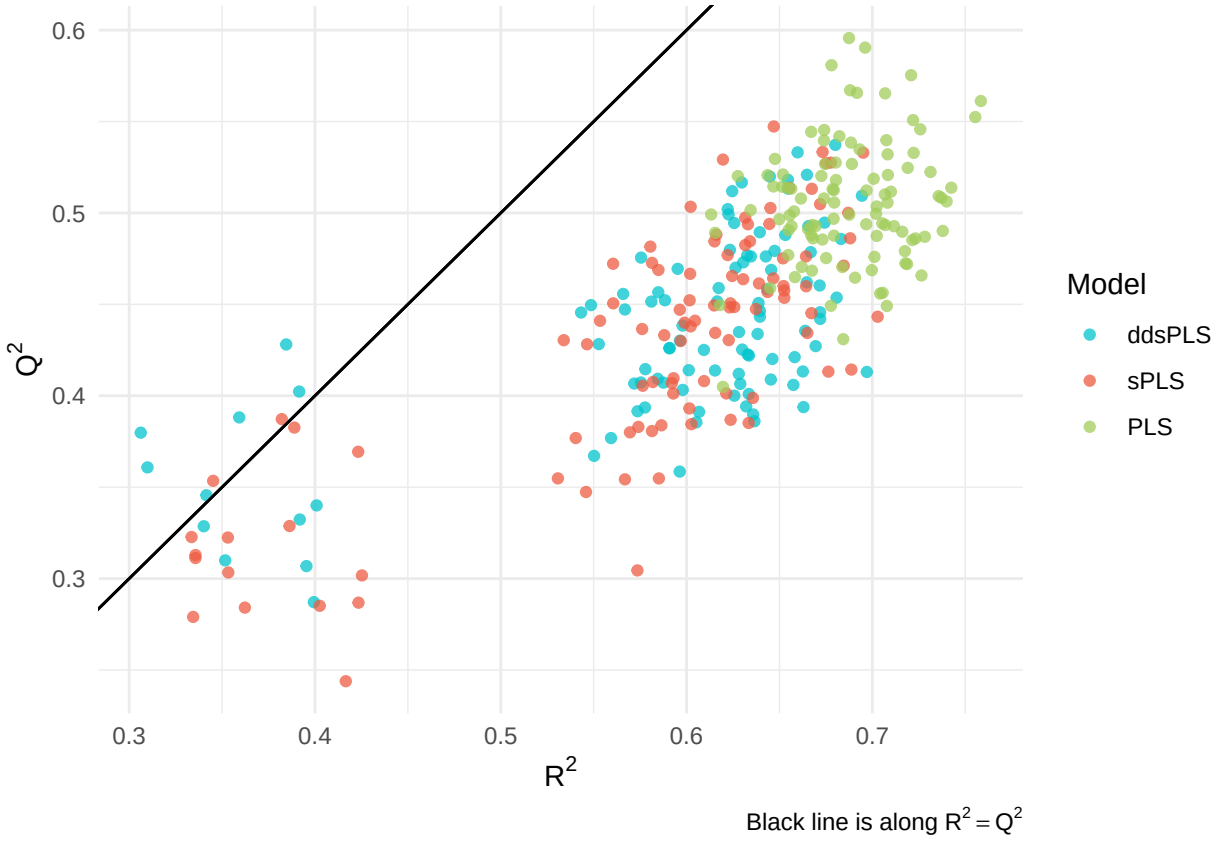

Figure 3.8: Relation between R^2 and Q^2. For the most part models perform smilarily except for a small group of sparse models.

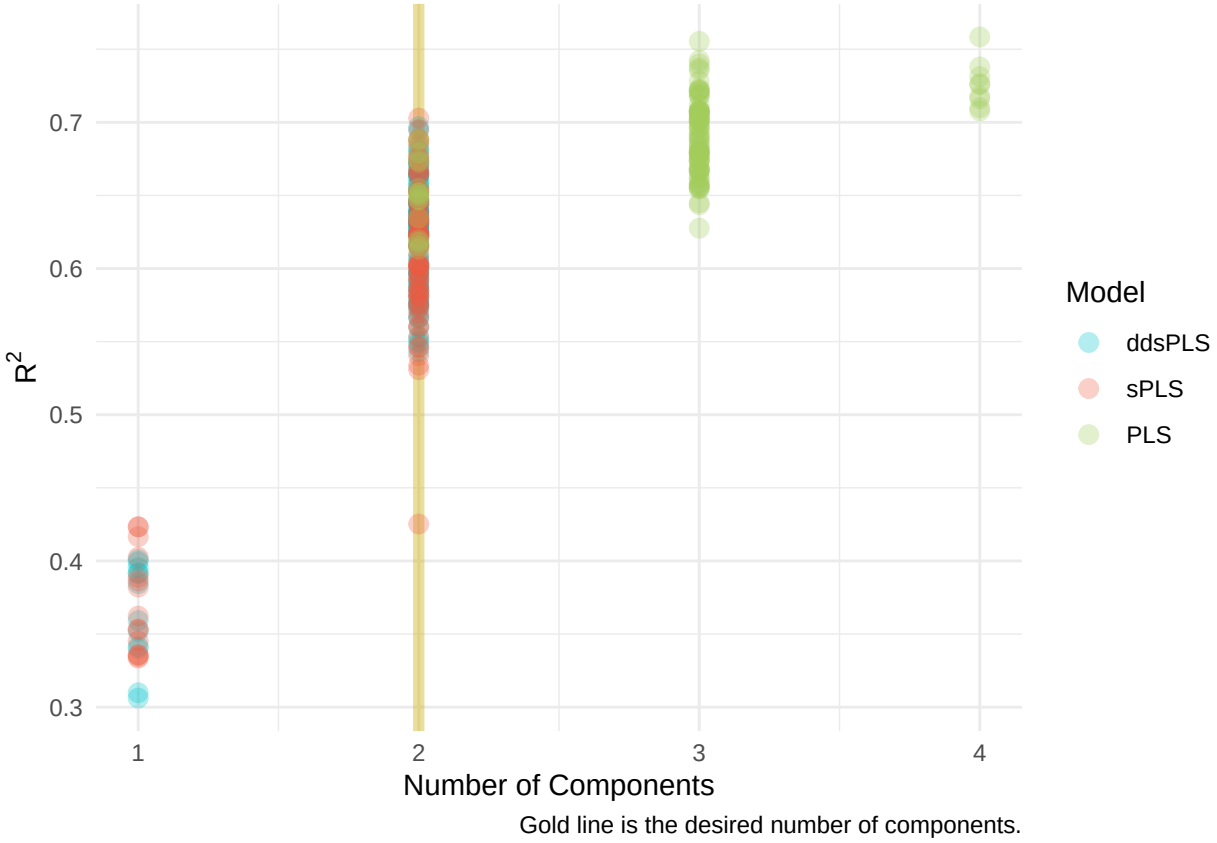

Figure 3.9: Relation between number of components built and R^2. Models with more components are better able to explain variance in the training data.

As the above plot illustrates, there is a clear relationship between the number of components and the R^2 value. Almost all of the models that poorly explain the original data build only 1 component. Keep in mind that a higher R^2 does not necessarily mean that the model has more predictive power or more accurately depicts the structure of the data.

Here we will also introduce a problem that sparse models will suffer from. Looking at the results from PLS models, we see what we would hope to observe, models consistently have fairly high R^2 values while also having lower but still well performing Q^2 values. For the most part, ddsPLS and sPLS exhibit the same behavior however, we can notice an issue with the cluster of points in the lower left corner. These points exhibit cases where the original model failed to provide a good fit of the training data as evidenced by the noticeably lower R^2 value. While we expect sparse models to have a lower R^2 than PLS, the problems is that these models do not offer consistent performance on similar data.

Due to the nature of these models, I would guess that this is due to the models

overly penalizing model complexity. Due to the frequency of these models that underfit the data (12 of ddsPLS models and 16 of sPLS models) it is extremely unlikely that these results are due to random variation in the data that the different models fit. Since this problem is prevalent in both ddsPLS and sPLS, at least part of it is likely due to the models either heavily penalizing or selecting to remove relevant predictors.

In this section we found that all models handle an increases in ω relatively similarly. In these simple cases, we saw that PLS had more predictive power in these cases with relatively simple data. However, both sparse models seem to better describe the structure of the data. Finally, we introduced problems that sparse models can have relating to model consistency.

3.3 Predictors

In this section we will discuss how models perform as p, the number of predictors, varies along with the structure of the predictors. For the most part, we will be adding more predictors that are uncorrelated with the response variables. Given the goals of ddsPLS and sPLS, we hope to see them perform relatively better as p increases.

The following are for data generated with $n = 50$, $q = 5$, $\omega = 1$, $\mathbf{A}$ generated with a complex structure, and $\mathbf{D}$ generated with the 2-block method.

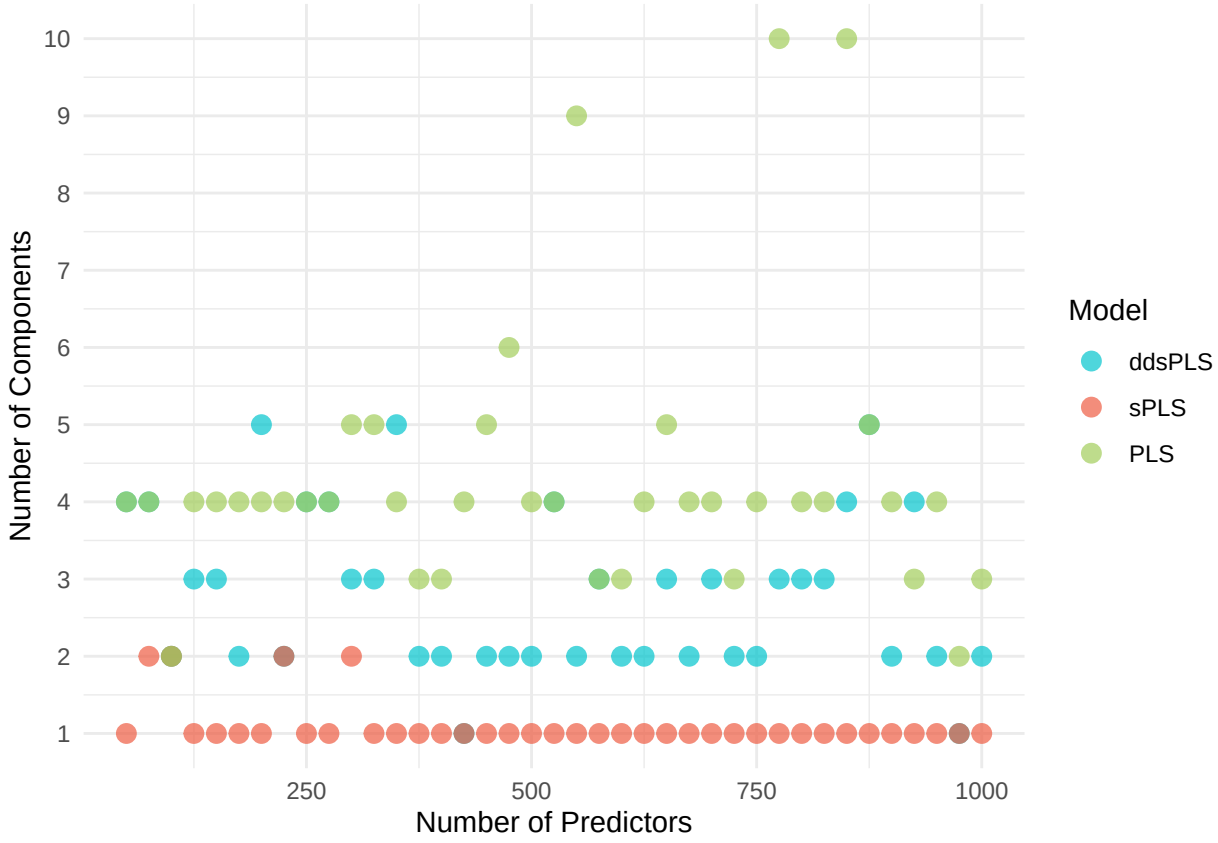

Figure 3.10: There is no clear relationship between the size of p and the number of components built. We can see that PLS tends to build the most components, ddsPLS the second most, and sPLS the fewest.

Here we can see the differing tendencies in the number of components that the models build. Ideally, models will build two components in this situation. ddsPLS performs by far the best in this regard as PLS tends to overfit the data while sPLS underfits it.

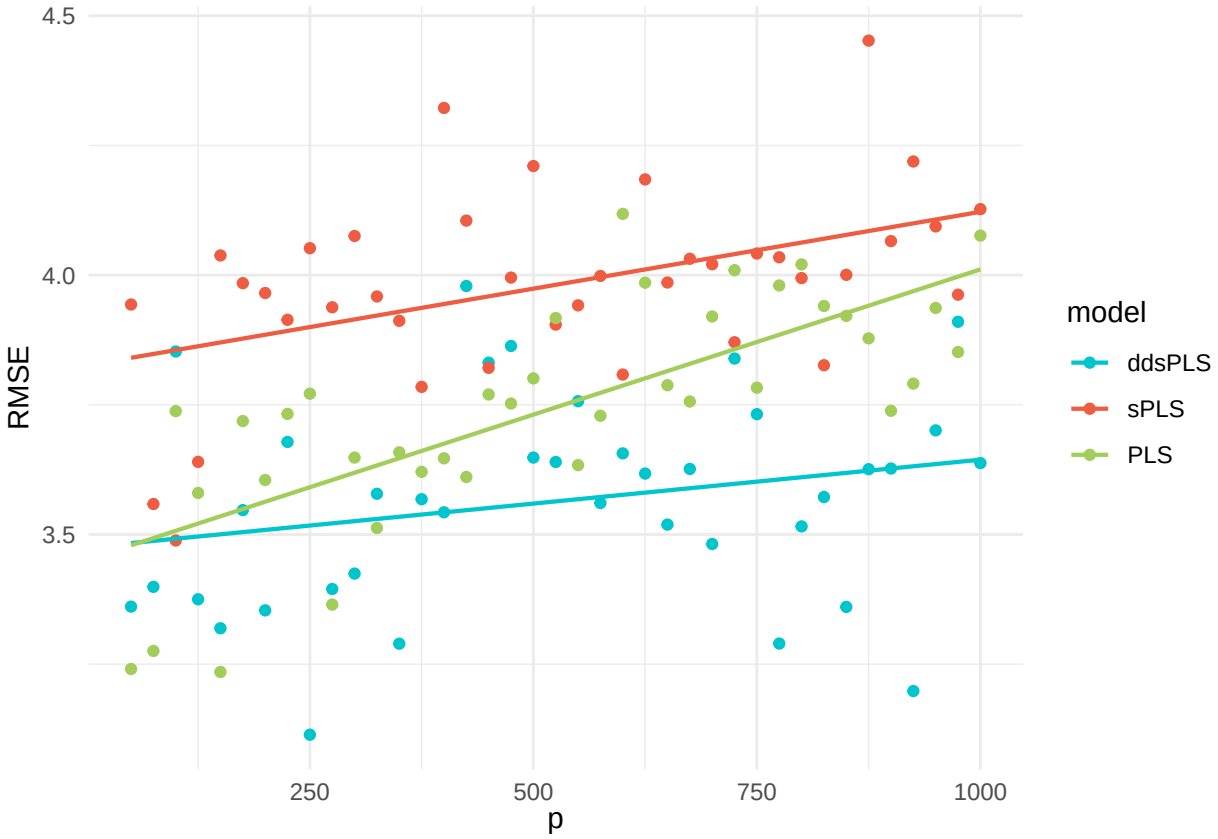

Figure 3.11: As the number of predictors increases, the performance of PLS declines at a faster rate than sparse models.

As p increases all PLS models show some degree of decline in performance as we would expect. Both of the sparse models exhibit a noticeable but fairly moderate increase in the RMSE. Meanwhile PLS exhibits a more drastic increase in its RMSE. This is expected as the new predictors are mainly noise which the sparse models will be able to better seperate from the signal.

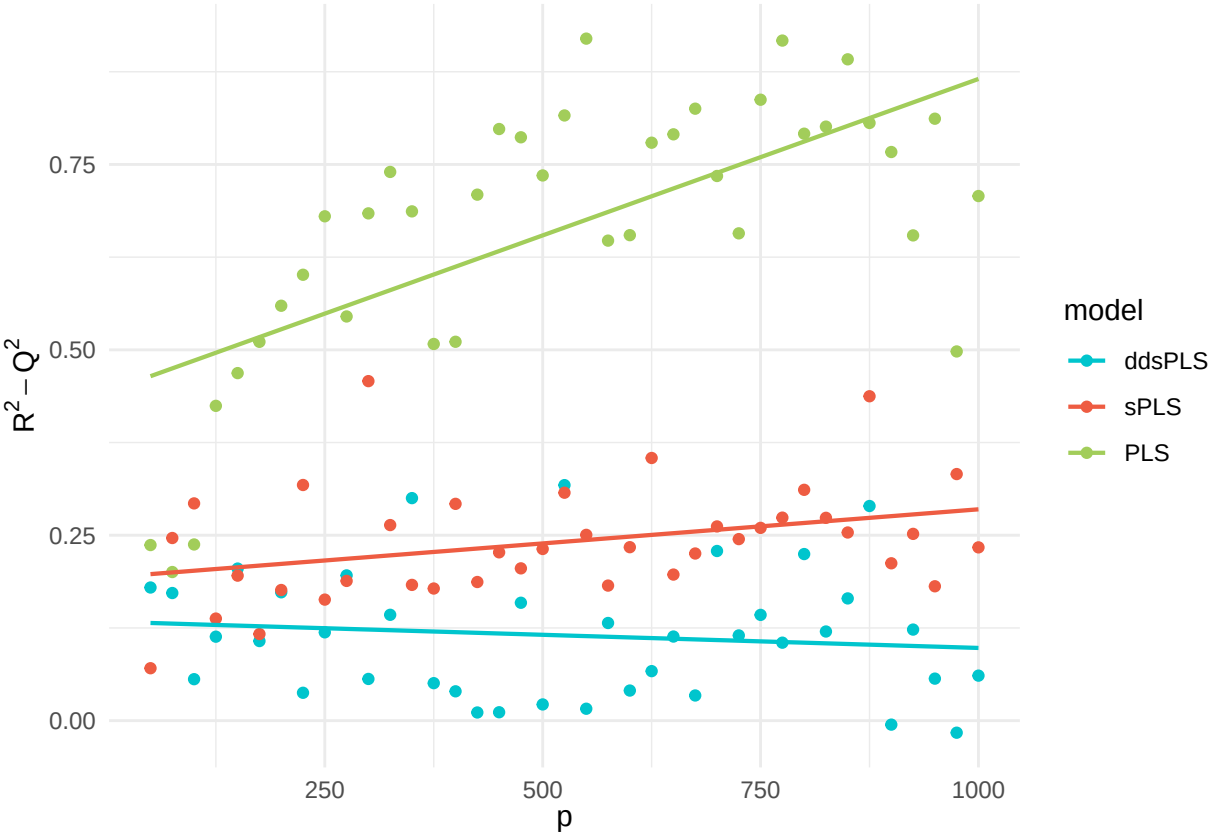

Figure 3.12: Change in $R^2 - Q^2$ as p increases. PLS exhibits much different behavior due to not having a sparsity constraint.

The increase in RMSE for the PLS model is due to it overfitting the data. As p increases, so does the difference in R^2 and Q^2. This is due to PLS fitting random correlation that appears in the noise that the sparse models are able to mostly ignore. Since ddsPLS is the only model to use $R^2 - Q^2$ as the metric for parameter selection it is unsurprising that it is the only one of the three models for which the metric decreases.

The following table shows the results of 100 replications with $p = 1000$ for each model type for the RMSE.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.569	3.454	3.565	3.674	0.0297
sPLS	4.052	3.954	4.041	4.142	0.0149
PLS	3.945	3.843	3.943	4.045	0.0185

From this chart we can see that with a large number of predictors uncorrelated to the responses, ddsPLS performs significantly better than the other methods in

regards to the RMSE. While the mean RMSE of PLS is significantly lower than that of sPLS the two often offer models that perform similarly. Furthermore, note the higher variance of the RMSE for ddsPLS models. This is representative of the performance of ddsPLS in general as models tend to be more variable in their performance on the test data.

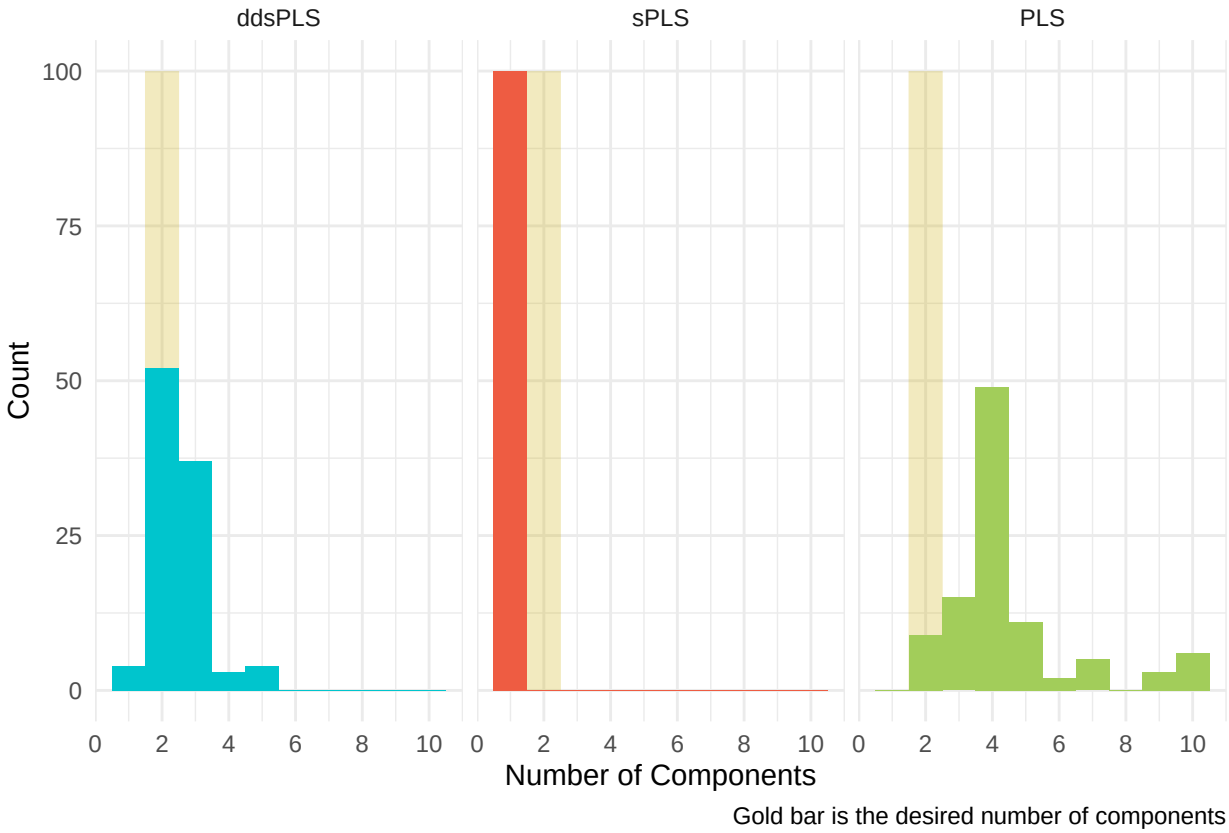

Figure 3.13: Distribution of components built across 100 simulations by model type when $p = 1000$.

When looking at the number of components built during the replications, we can see that ddsPLS is the only model that tends to build the correct number of components. As in previous trials, sPLS uses too few components while PLS builds too many.

In our assessment of the models as p varies, we found that ddsPLS and sPLS suffer from some decline in performance as uncorrelated predictors are added. This decline in performance is far less drastic than the one shown by PLS in the same situation. Furthermore, we have seen ddsPLS deliver consistently better performance that sPLS both in regards to making accurate prediction and capturing the structure of the data.

3.4 Responses

In this section we will discuss how models perform as q, the number of response variables, varies along with their structure. In addition to changing the structure of the responses, we will also change the structure of the responses at times. The goal of this is to assess model performance when faced with significantly more complex data structures than in previous parts. We expect ddsPLS to handle increases in the number of uncorrelated or weakly correlated response variables the best as it it the only method that directly penalizes $\mathbf{Y}$, the matrix of responses.

The following are for data generated with $n = 50$, $p = 100$, $\omega = 0.5$, $\mathbf{A}$ generated with a simple structure, and $\mathbf{D}$ generated with the simple method.

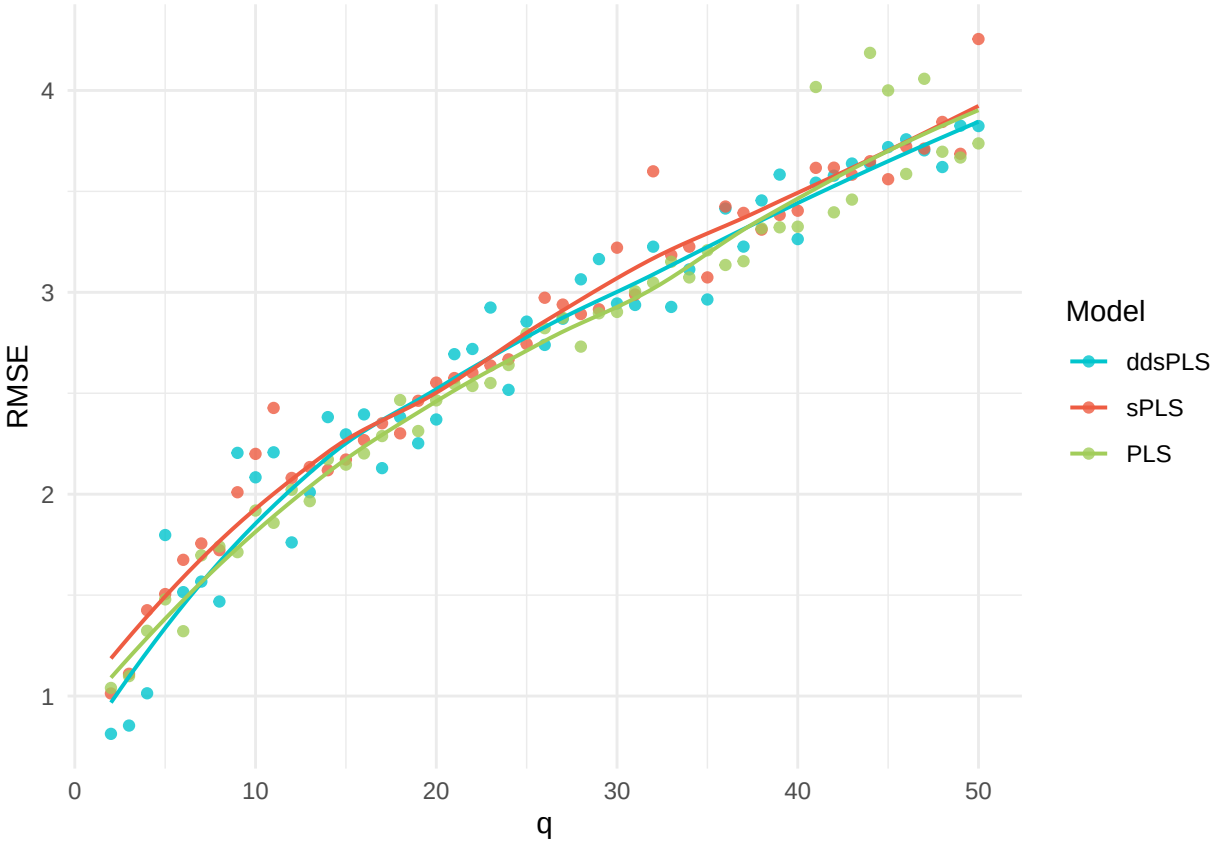

Figure 3.14: Relation between q and the RMSE.

As expected, the RMSE increases as the number of response variables increase. More importantly, we can see that all models continue to perform similarly as the number of response variables increases.

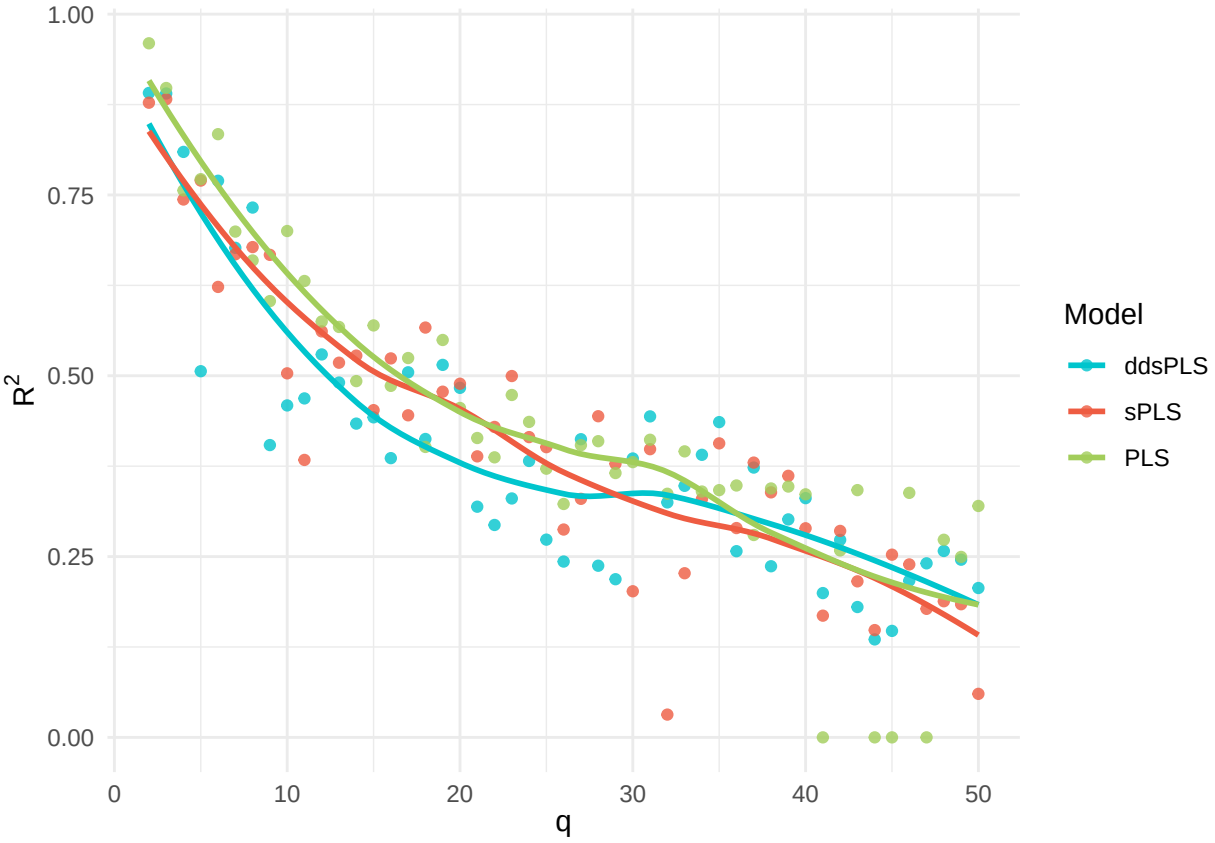

Figure 3.15: As q increases, we see all models perform similarly in their ability to fit the data.

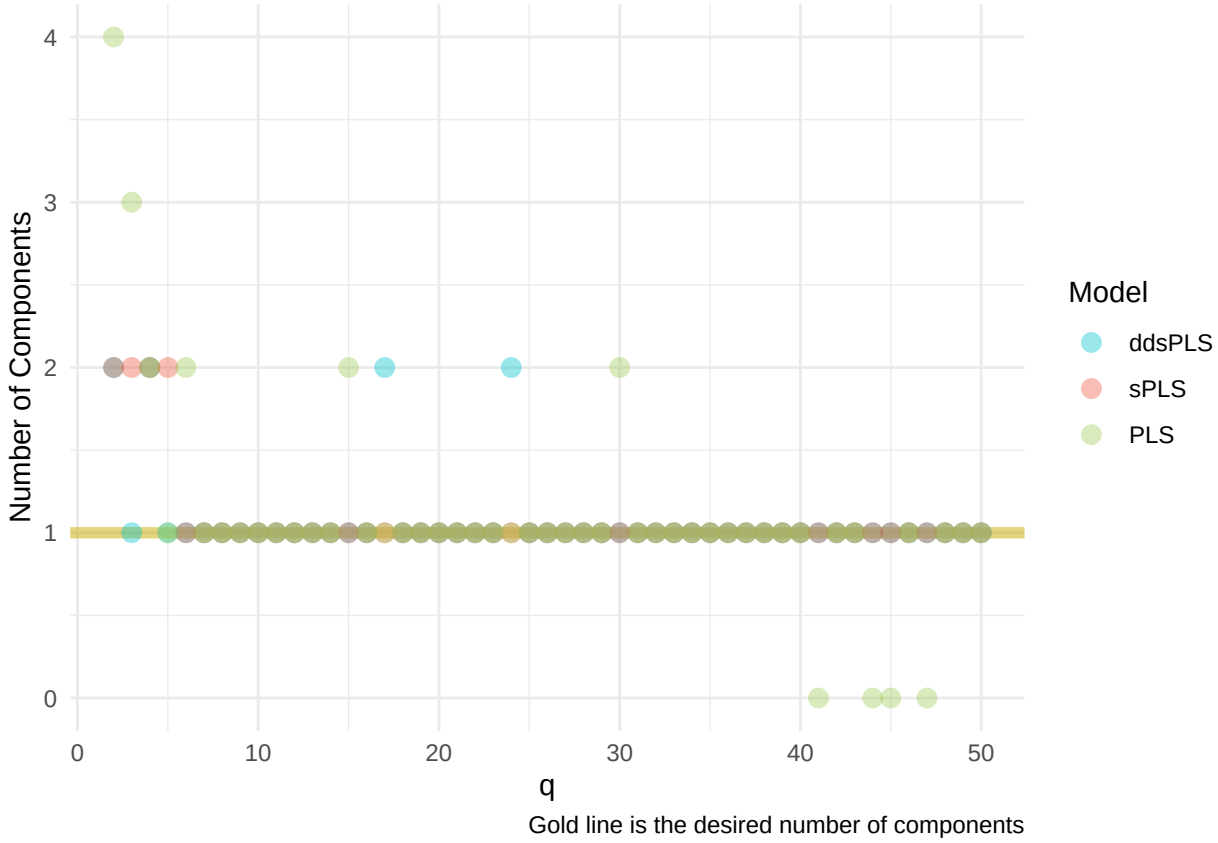

Figure 3.16: Number of components built as q increases.

Models tend to build the correct number of components in this simple case when there are a small number of responses, we can see that all models have a slight tendency to build too many components however, this quickly stops at around $q = 5$.

The following results are for $q = 25$, $n = 50$, $p = 100$, $\omega = 0.5$, $\mathbf{A}$ generated with a simple structure, and $\mathbf{D}$ generated with the basic method.

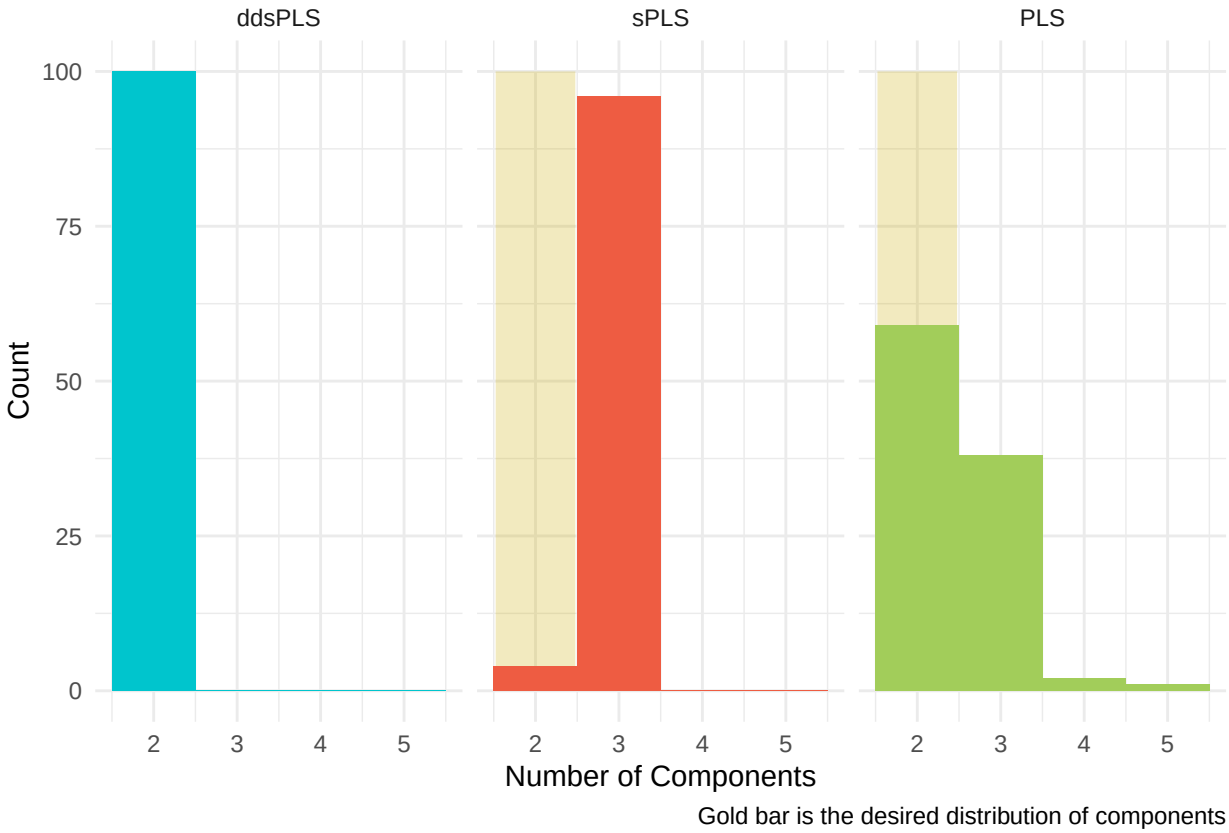

Figure 3.17: Distribution of components built across 100 simulations by model type when $q = 25$ and $n = 50$.

Interestingly, sPLS is most likely to build too many components in this scenario where there are a large number of predictors that are all strongly correlated. Here ddsPLS is able to build the correct number of components every time. This may be due to the fact that sPLS doesn't directly impose sparsity on the response variables.

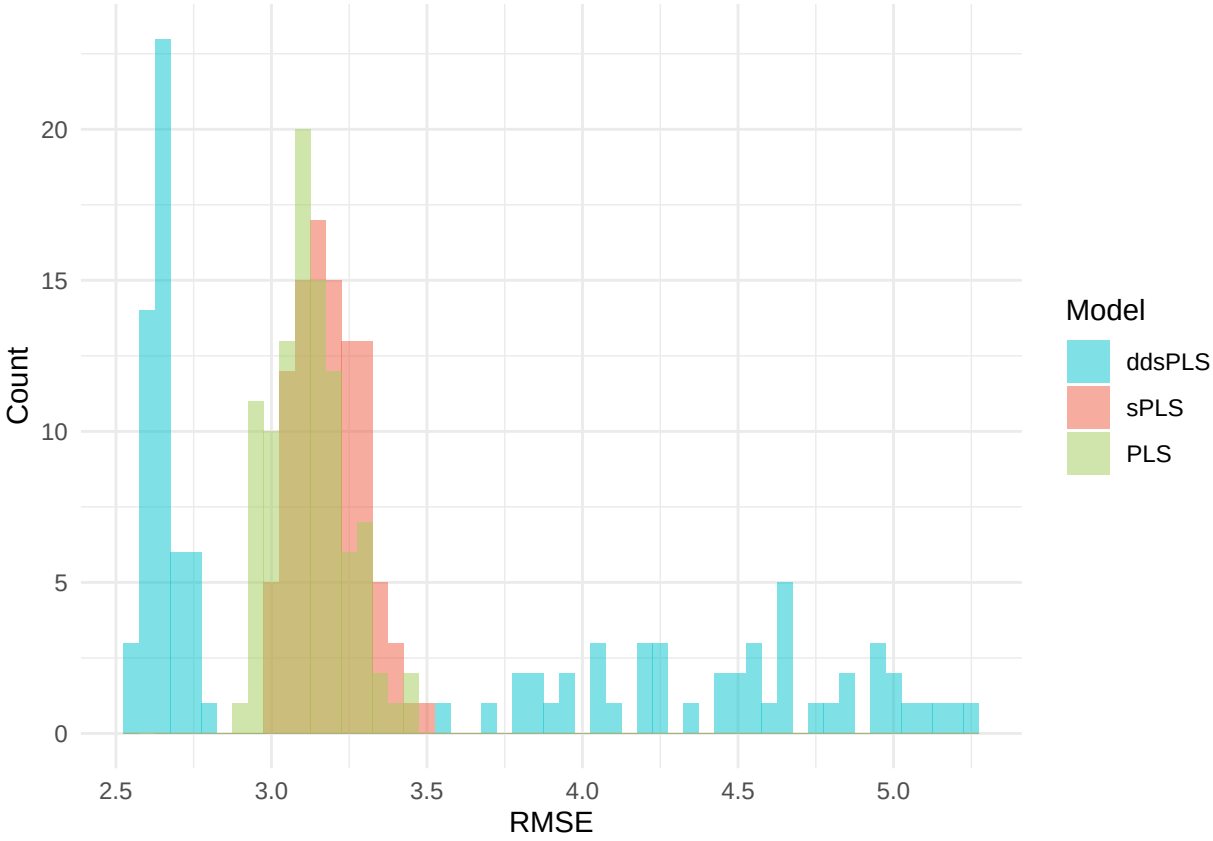

Figure 3.18: Distribution of RMSE by model type. Note the inconsitency in performance for ddsPLS.

sPLS and PLS have a similar distribution of the RMSE with PLS performing slightly better. ddsPLS exhibits a much different behavior with models either performing notably better than the other two methods or far worse.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.502	2.644	2.756	4.448	0.9230
sPLS	3.187	3.107	3.178	3.257	0.0118
PLS	3.124	3.042	3.112	3.200	0.0145

Based on summary statistics of the distribution of RMSE we can see that the variance of ddsPLS performance is much higher than the other two models which perform fairly similarity. A slight majority of ddsPLS models perform better with the rest performing much worse. This seems to signal ddsPLS as a high-risk, high-reward model to use in these cases.

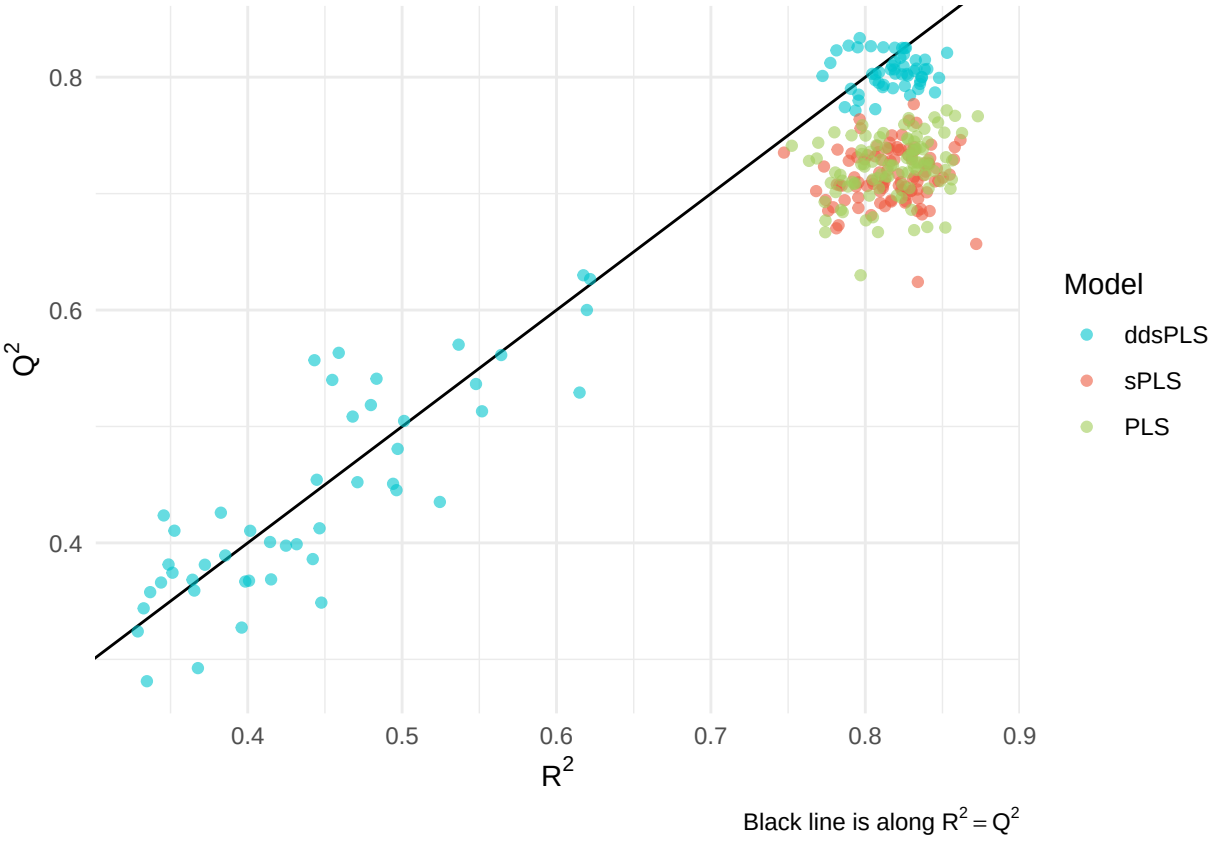

Figure 3.19: Relationship between R^2 and Q^2 when $q = 25$ and $n = 50$. Note that ddsPLS exhbits noticeably different and more variable behavior than other models.

Here we can see that low R^2 values seem to be causing the high RMSE. Notice that for ddsPLS, $Q^2 \geq R^2$ in many cases, behavior that we do not expect to see. This suggests that the method is doing a good job of ignoring noise in the data and only reflecting parts of the underlying structure. The problem seems to be that ddsPLS is also ignoring signal at times and only capturing part of the underlying structure. In these instances, we see a lower R^2 value and thus a similar but also lower Q^2 value. Note for that sPLS and PLS, we always have $Q^2 < R^2$ showing that the models are to some degree fitting noise.

The following models are generated in a similar way as before, except this time $\mathbf{D}$ is generated with the 2-block method.

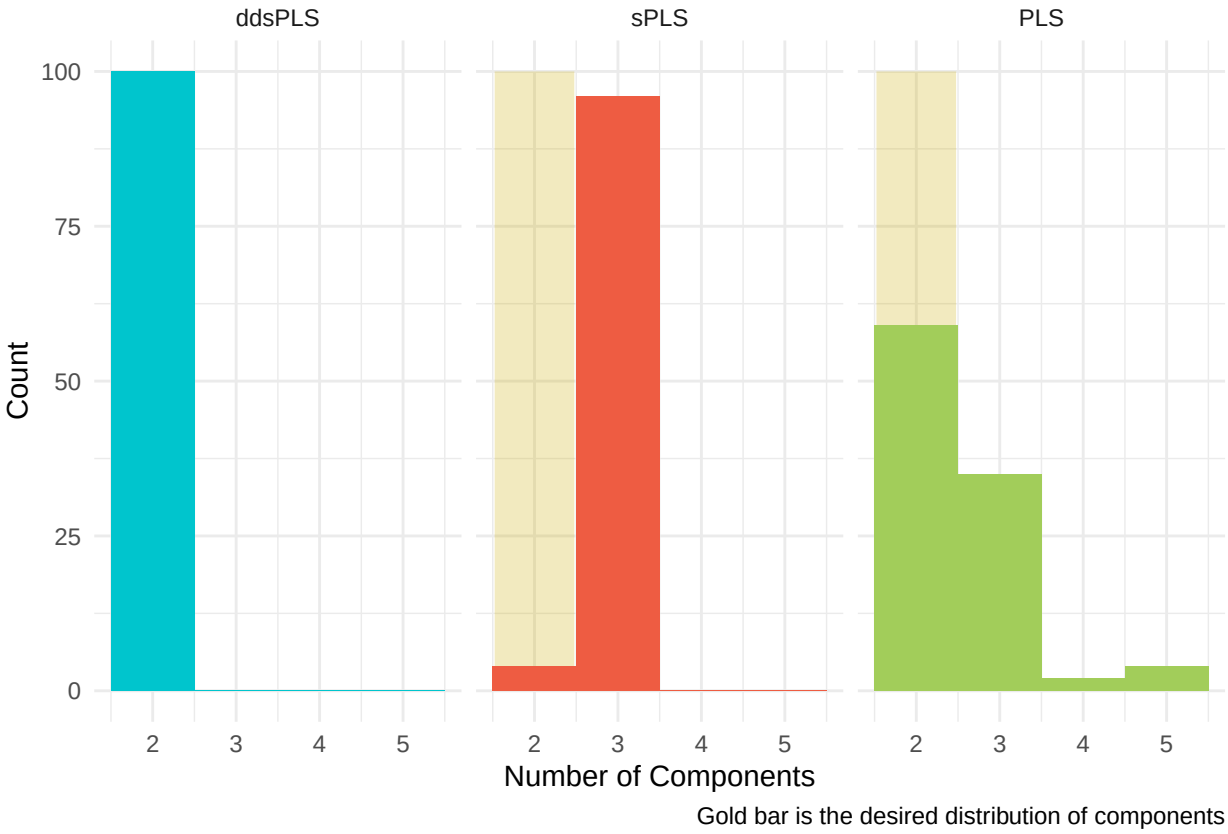

Figure 3.20: Distribution of components built across 100 simulations by model type when **D** is generated with the 2-block method.

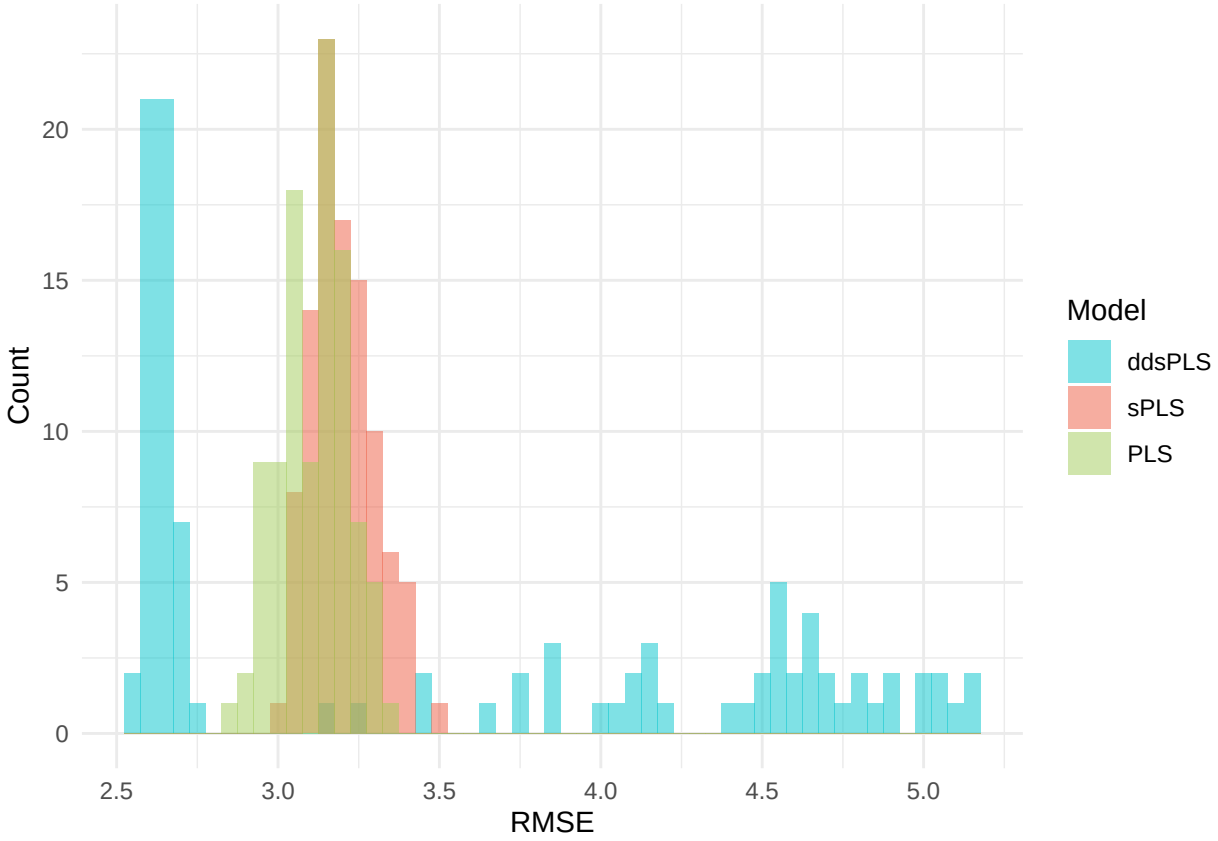

Figure 3.21: Distribution of RMSE by model type when **D** is generated with the 2-block method. Again, note the inconsitency in performance for ddsPLS.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.480	2.627	2.718	4.531	0.9141
sPLS	3.197	3.134	3.181	3.263	0.0097
PLS	3.112	3.033	3.127	3.185	0.0110

Comparing these to the results of using the diagonal method to generate **D** we see little change. Nearly identical results appear both times.

The following are for $q = 10$ with **A** generated with a complex structure and **D** generated with the 4-block method.

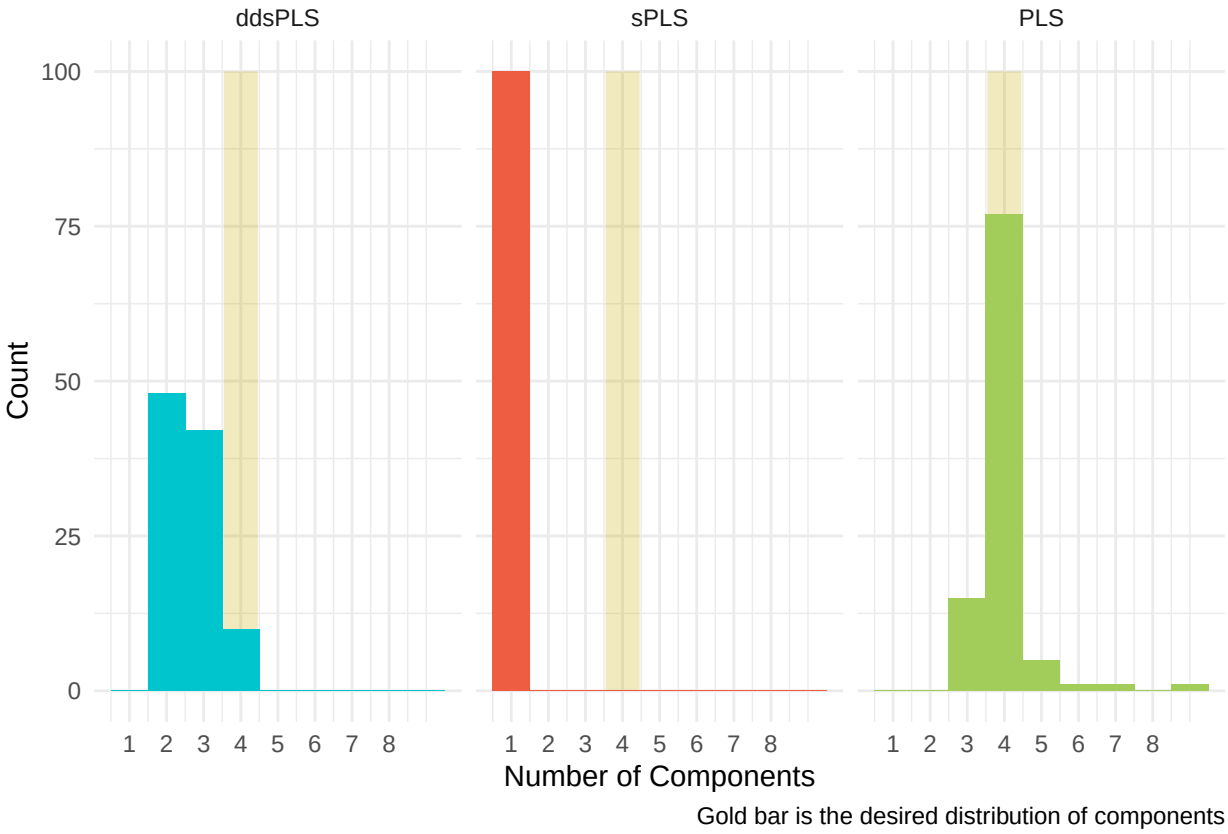

Figure 3.22: Distribution of components built across 100 simulations by model type when data has a more complex structure and $n = 50$. This is one of the rare times PLS performs the best by this metric.

With a more complex structure to both the predictors and response variables, ddsPLS and sPLS tend to build too few elements. Surprisingly, PLS is best at building the correct number of components in this situation.

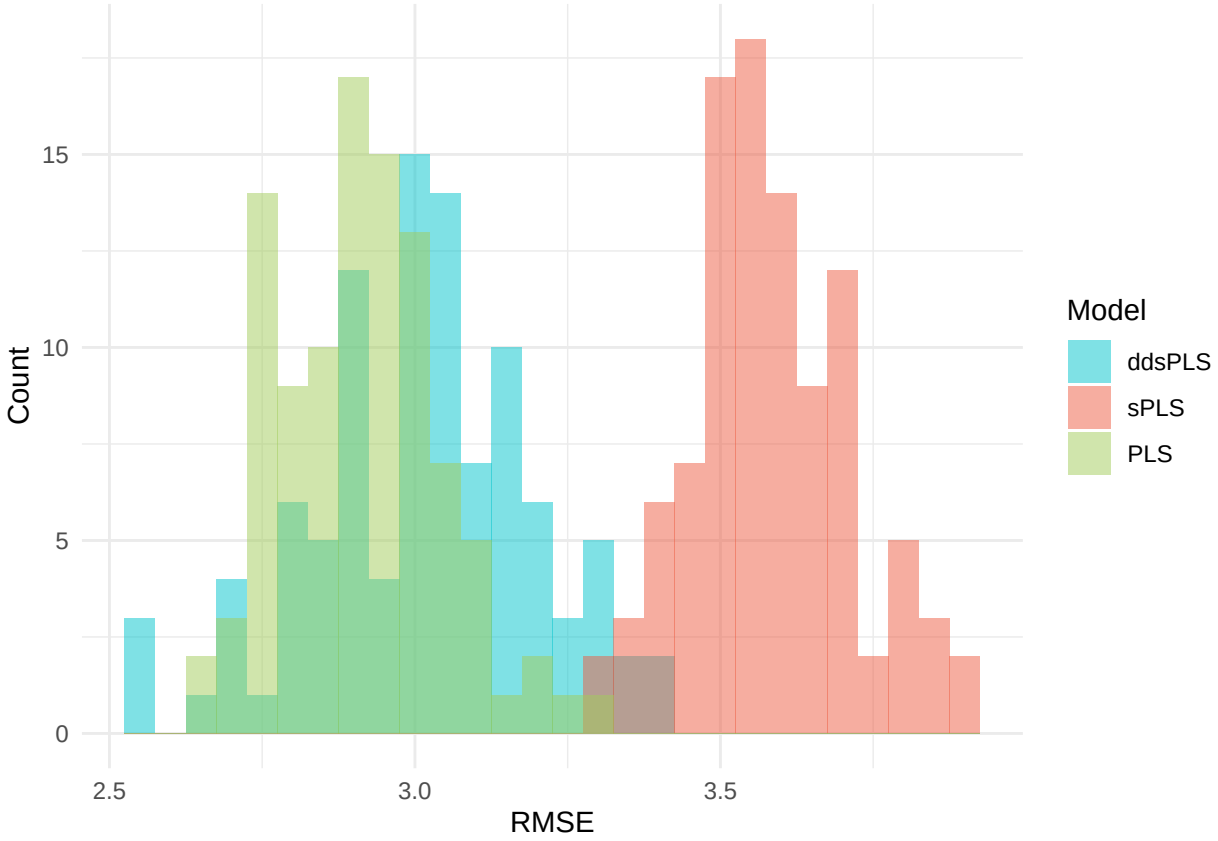

Figure 3.23: Distribution of RMSE by model type when data has a more complex structure and $n = 50$. Note the more consistent performance by ddsPLS here.

ddsPLS and PLS perform similarly in regards to the RMSE, with PLS having a slight edge in performance. sPLS performs significantly worse.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.016	2.914	3.024	3.132	0.0343
sPLS	3.582	3.496	3.572	3.671	0.0176
PLS	2.910	2.808	2.893	2.992	0.0169

While ddsPLS doesn't suffer as drastic a problem with underfitting as in previous trials, it still has the most variable model performance.

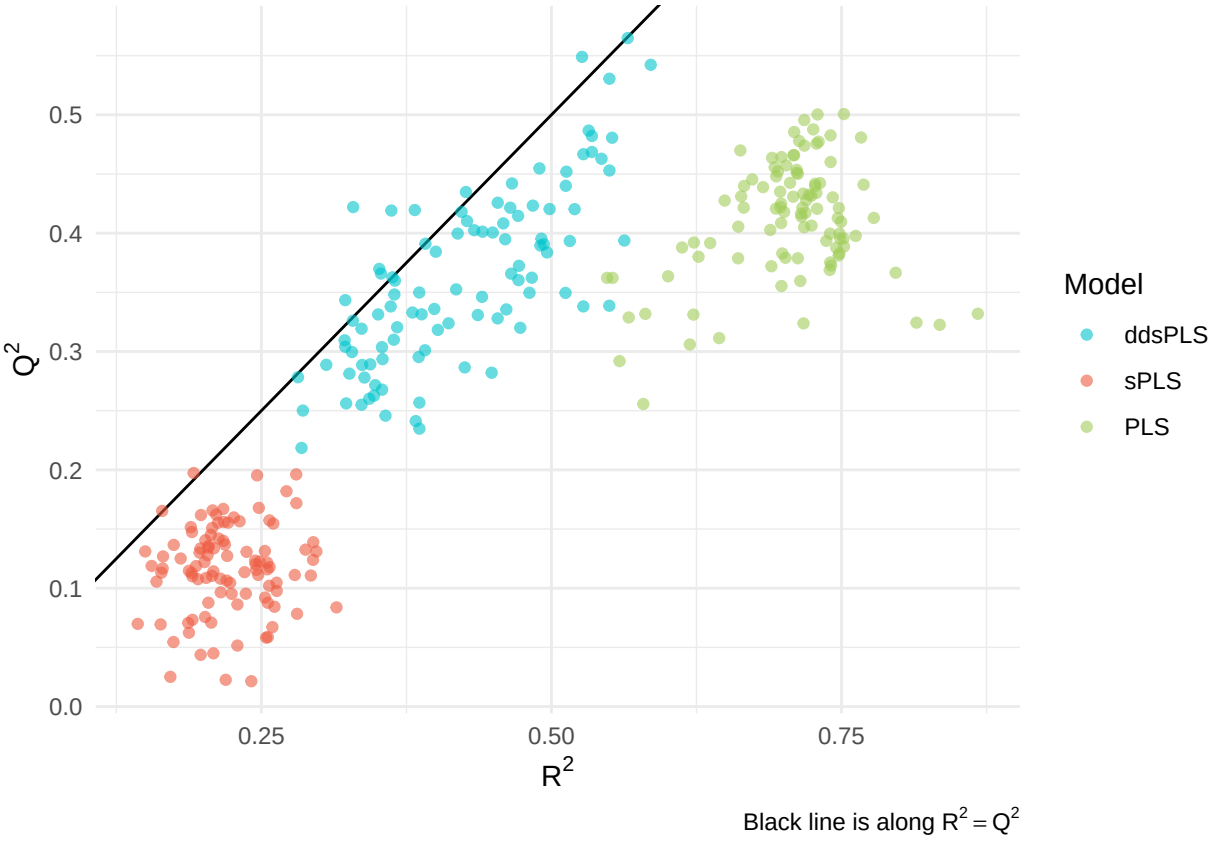

Figure 3.24: Relationship between R^2 and Q^2 when data has a more complex structure and $n = 50$. Note the distinct performance of each model type in this situation.

Both ddsPLS and sPLS exhibit an ability to avoid including noise in their models. Especially in the case of sPLS, this looks like it also causes them to miss some of the signal. While PLS models are more prone to include noise in the models as we can see in the larger difference between R^2 and Q^2. This also suggests that while PLS is building the correct number of components, these components may not be the most accurate.

The following results are for a simulation of models built with the same structure as before this time with a sample size of $n = 100$.

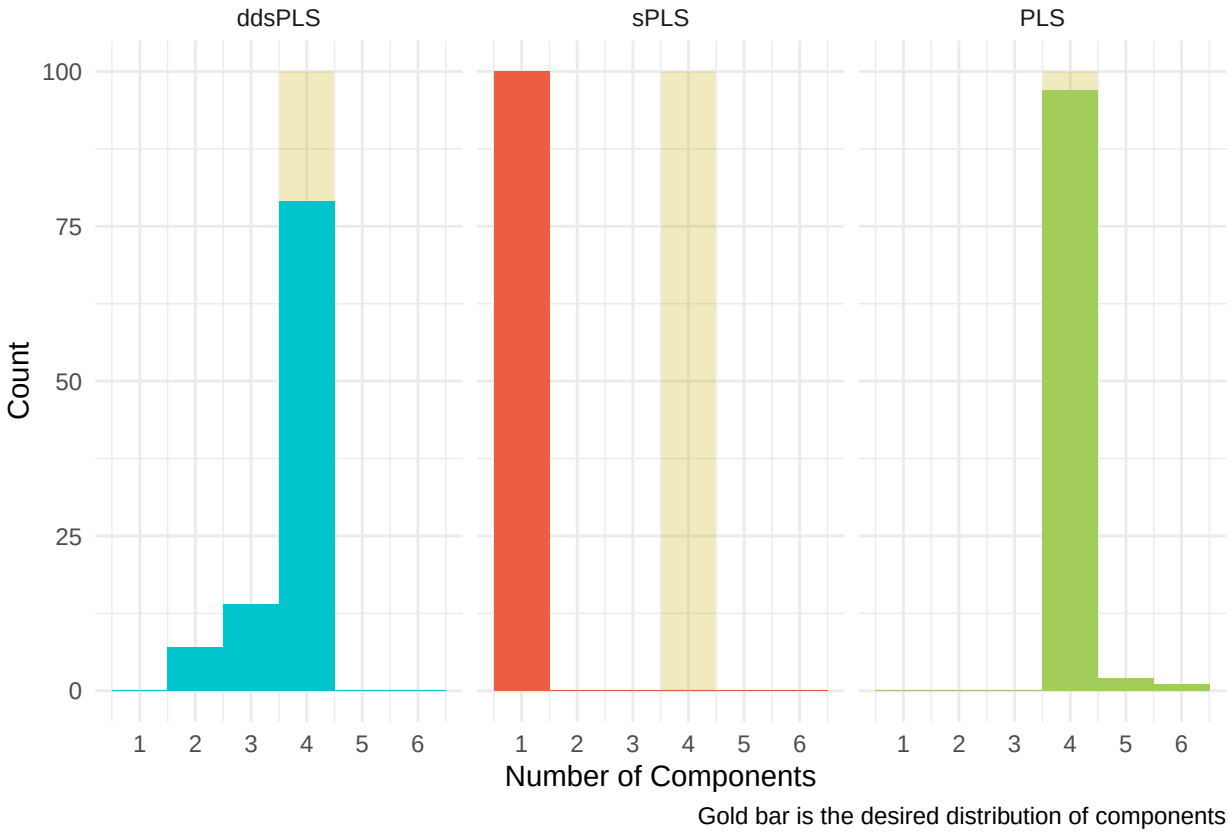

Figure 3.25: Distribution of components built across 100 simulations by model type when data has a more complex structure and $n = 100$. Note that while performance of PLS is still the best, ddsPLS is much improved by the larger sample.

With a larger sample size, we now see that ddsPLS tends to build the correct number of components. This suggests that the number of components built when $n = 50$ was largely due to the sample size and less indicative of a problem with the performance of ddsPLS on more complex data structures. sPLS again builds only 1 components every time suggesting that the model does have issues on data with a more complex structure.

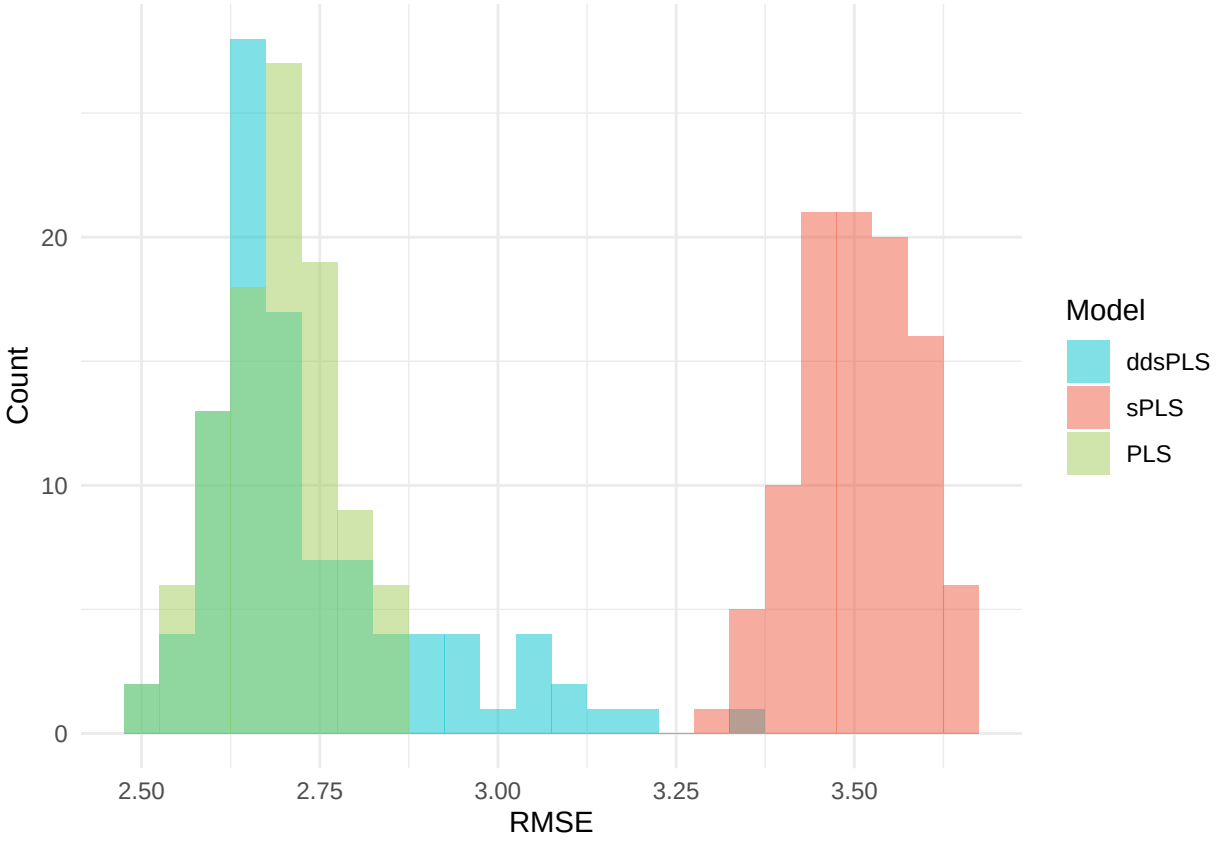

Figure 3.26: Distribution of RMSE by model type when data has a more complex structure and $n = 100$. Note that consistency issues with ddsPLS are more apparent here.

With the larger sample size the performance of ddsPLS and PLS becomes much more similar. sPLS doesn't exhibit the same decrease in RMSE that the other two models see. ddsPLS still exhibits some of the problem with variability in model performance that we have observed as is evidenced in the longer tail.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	2.737	2.640	2.684	2.809	0.0259
sPLS	3.507	3.449	3.511	3.563	0.0064
PLS	2.691	2.634	2.694	2.743	0.0061

PLS still has a slightly lower RMSE than ddsPLS on average. Again, ddsPLS has a significantly higher variance than the other two models. Furthermore, the variance of the RMSE for ddsPLS has declined at a lower rate than other models.

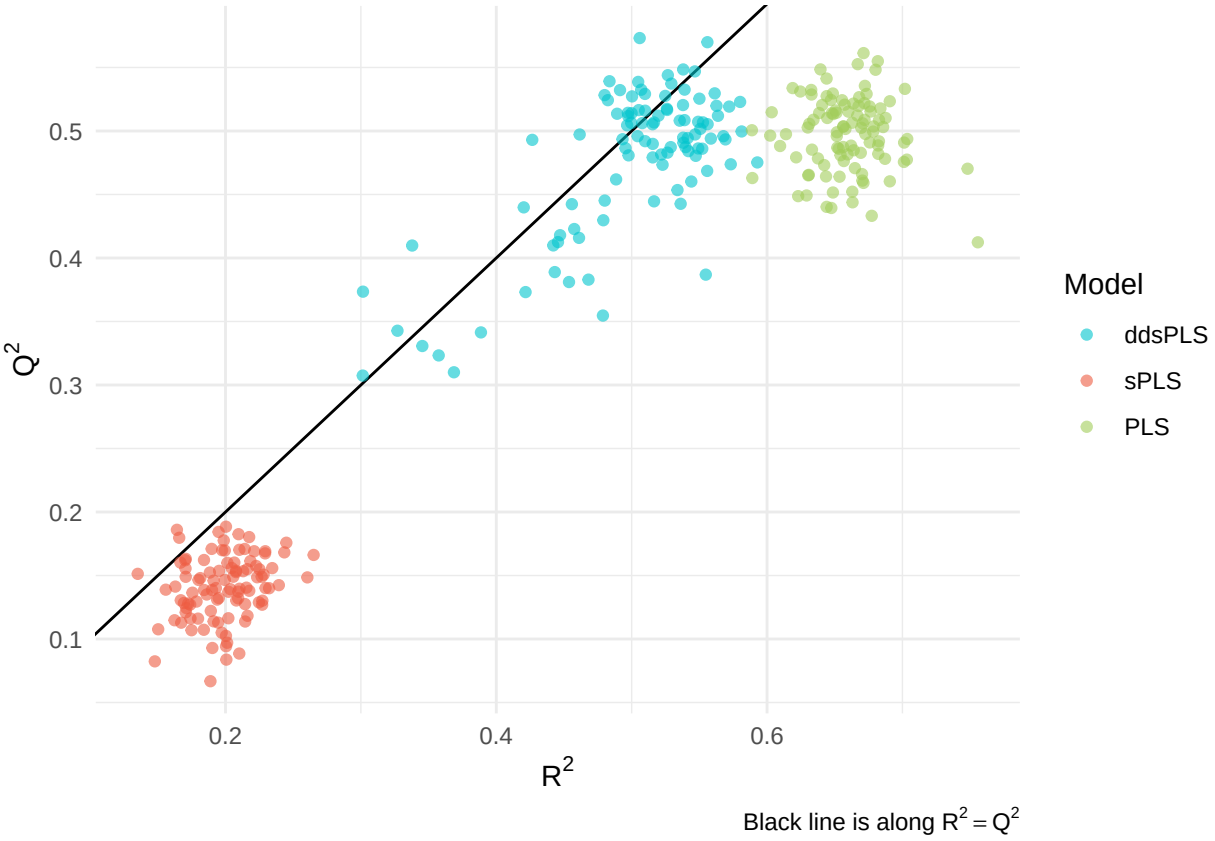

Figure 3.27: Relationship between R^2 and Q^2 when data has a more complex structure and $n = 100$. This plot clearly shows that ddsPLS has more variable performance both on the training and test data.

This plot clearly illustrates the problems of points that underfit the data. A number of ddsPLS models have noticeably lower R^2 values and thus lower Q^2 values. Both sPLS both have a more defined cluster of points, exhibiting close R^2 and Q^2 values across all models.

In our assessment of the models as q varies, we found that ddsPLS only partially delivers on its promise. While it can outperform PLS, it also suffers from much higher variance than other models due to missing part of the data's structure at times. Furthermore, for complex data with a small sample size, PLS seems to best capture the structure of the data.

3.5 Parameter Selection

This section compares using $R^2 - Q^2$ and Q^2 as metrics to tune the parameters $(\lambda_1, ...\lambda_k)$ of ddsPLS. $R^2 - Q^2$ is the recommended for parameter tuning with ddsPLS, however, Q^2 is commonly used to select parameters for similar PLS based models.

Based on the results of previous sections, we have seen that ddsPLS has unique problems with models that have low values of $R^2 - Q^2$ while performing relatively poorly in terms of Q^2. In this section we hope to investigate if Q^2 serves as a better metric to use for parameter selection.

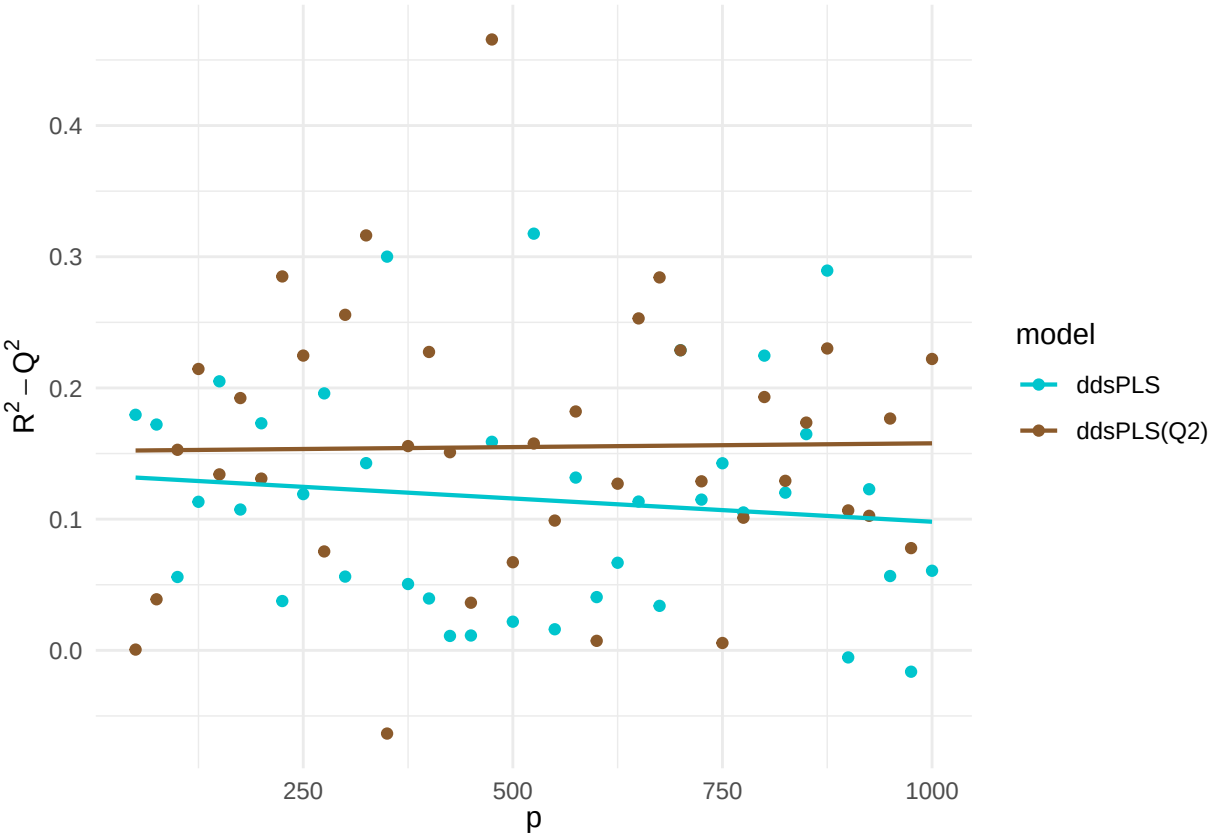

Figure 3.28: Relation between p and $R^2 - Q^2$ for ddsPLS models using different metrics for parameter selection.

As p increases, $R^2 - Q^2$ slightly decreases for the ddsPLS model for which $R^2 - Q^2$ is used as the metric for parameter selection, this is a positive sign as for more traditional techniques we would expect $R^2 - Q^2$ to increase as the amount of noise present in the predictors increases. When using Q^2 as the metric, we don't see a noticeable change in $R^2 - Q^2$. This suggests that there may not be that much of a need to use $R^2 - Q^2$ in order to prevent overfitting.

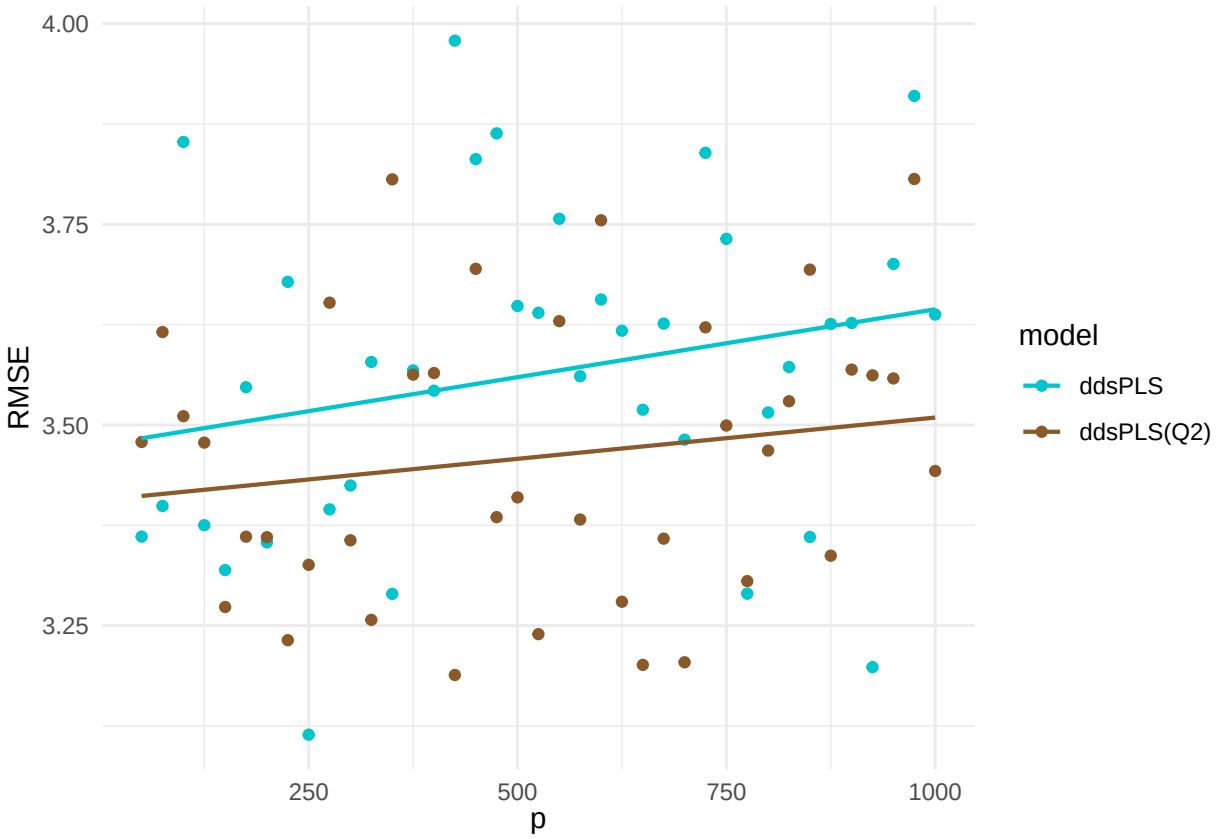

Figure 3.29: Relation between RMSE and Q^2 for ddsPLS models using different metrics for parameter selection.

Both models exhibit similar performance in terms of the RMSE with the model built using $R^2 - Q^2$ as a metric performing slightly better. We would expect the model using Q^2 as a metric to perform better since this is what it attempts to minimize. However, this suggests that there may be reason to prefer $R^2 - Q^2$ as a metric if it is more discriminating and offer comparable or better performance.

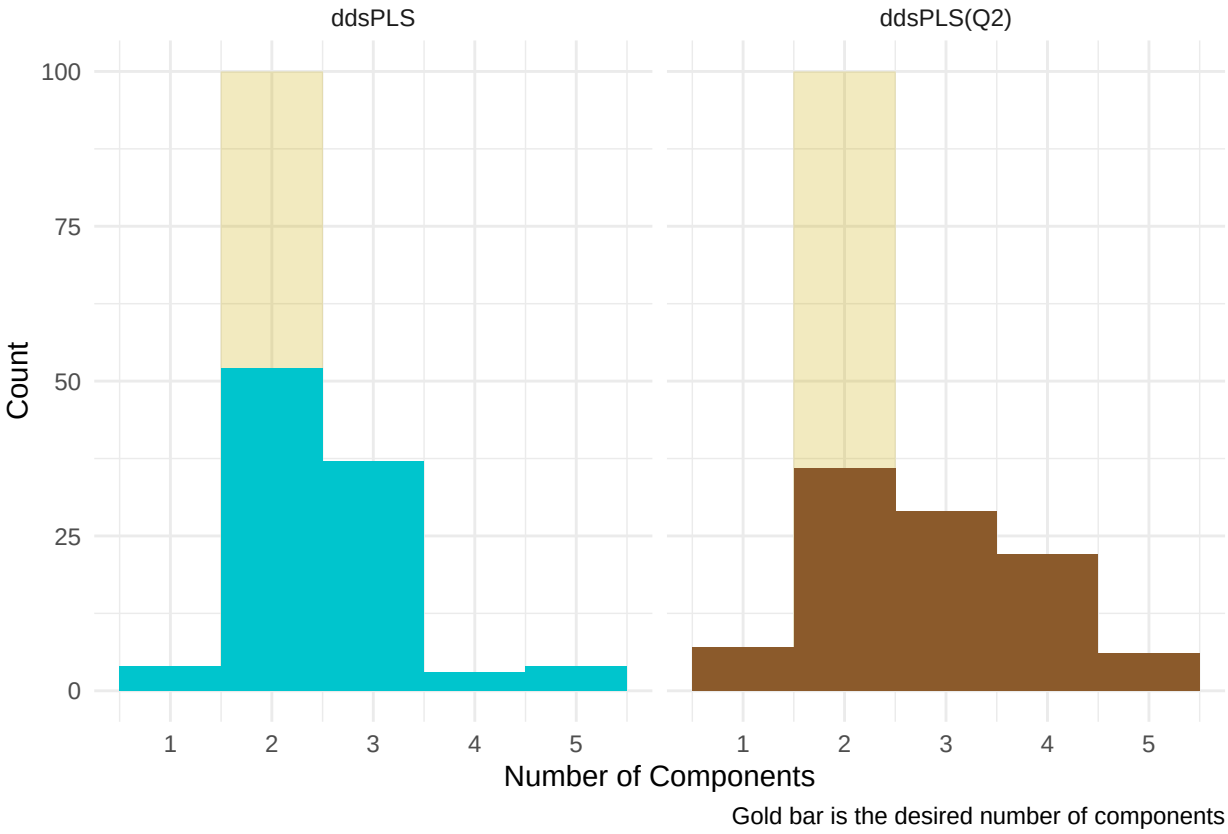

Figure 3.30: This plot illustrates a drawback of using Q^2 in order to select parameters. Models using Q^2 to select parameters are more likely to build too many components

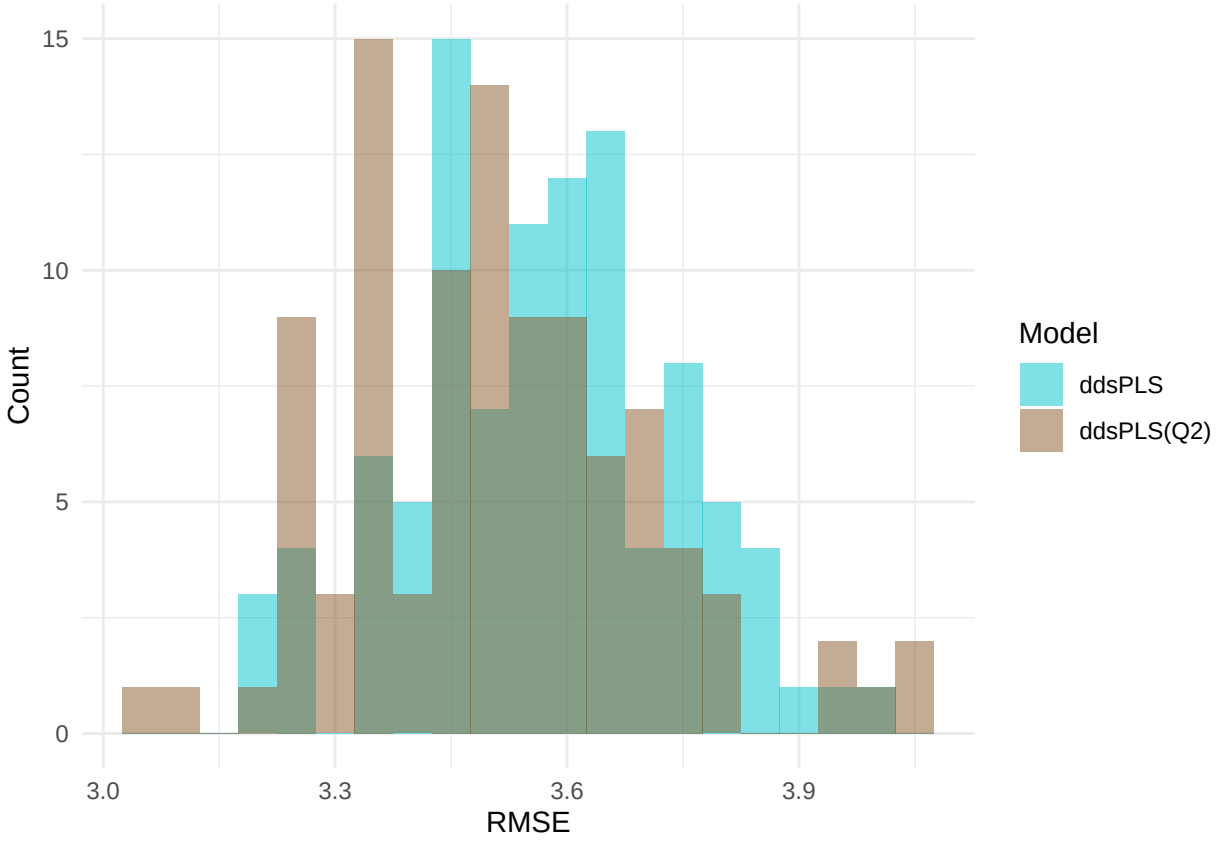

Figure 3.31: Distribution of RMSE by metric used for parameter selection.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.569	3.454	3.565	3.674	0.0297
ddsPLS(Q2)	3.508	3.356	3.492	3.618	0.0385

In this case, using Q^2 as a metric for parameter selection gives a slight advantage in predictive performance. This does seem to come at a slight decrease in the ability of the model to accurately capture the structure of the data.

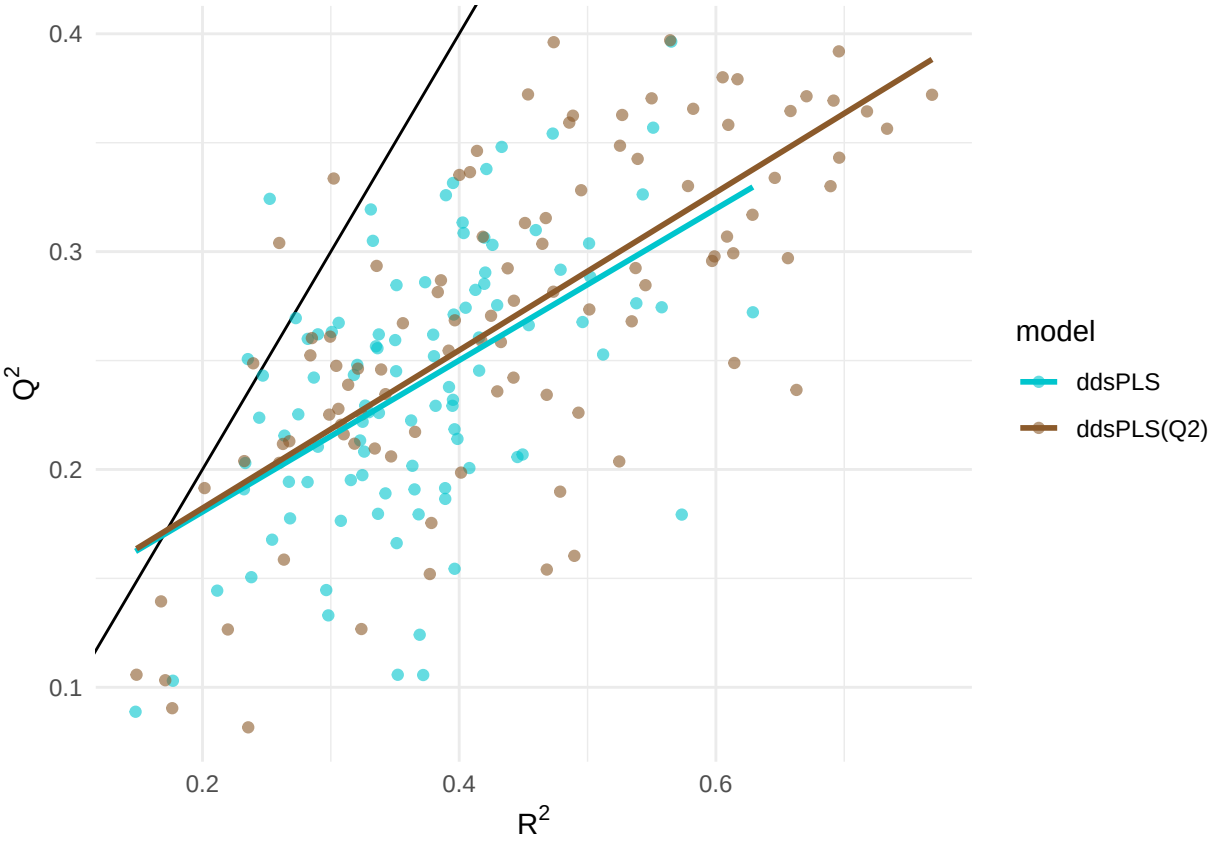

Figure 3.32: Relation between R^2 and Q^2 with $p = 1000$ by metric used for parameter selection.

In this plot we see that the relation between R^2 and Q^2 is very similar for the two methods of parameter selection. When comparing ddsPLS to other models, we often saw that ddsPLS would have a closer association between R^2 and Q^2, i.e. if a PLS and ddsPLS model had the same R^2 value, then the ddsPLS model would tend to have a higher Q^2 value. However, this appears more to be a feature of how ddsPLS models rather than due to the fact that $R^2 - Q^2$ is used for parameter selection.

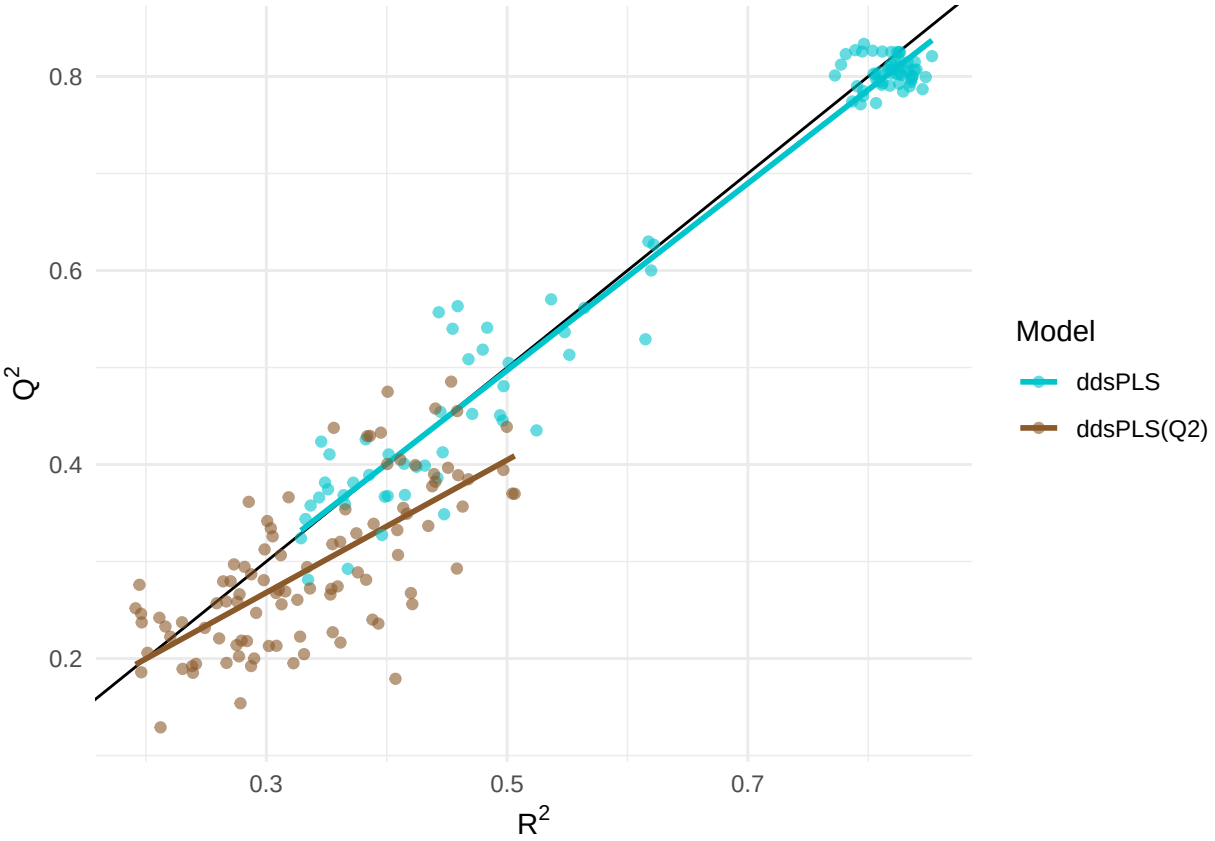

Figure 3.33: Relation between R^2 and Q^2 when $q = 25$ by metric used for parameter selection.

Here, $q = 25$, using $R^2 - Q^2$ for parameter selection far outperforms using Q^2. With a large number of responses it seems that using Q^2 struggles to accurately capture the structure of the data in all cases, as opposed to the small grouping of poorly performing models that are created when $R^2 - Q^2$ is used.

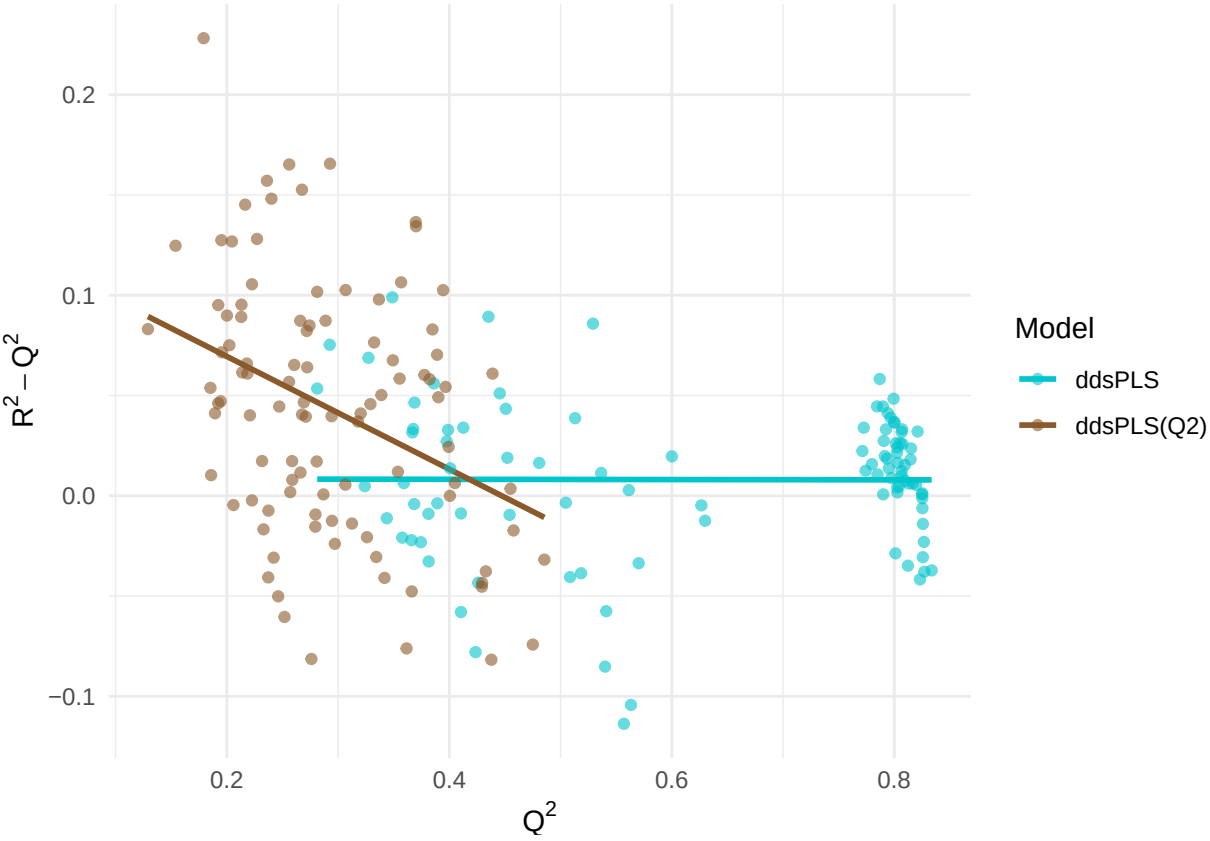

Figure 3.34: Relation between Q^2 and $R^2 - Q^2$ when $q = 25$ by metric used for parameter selection.

This is a plot of Q^2 and $R^2 - Q^2$ from simulations discussed in the responses section. For ddsPLS, there appears to be little relation between Q^2 and $R^2 - Q^2$ when $R^2 - Q^2$ is used for parameter selection. However, when Q^2 is used, better performing models have a low $R^2 - Q^2$ value.

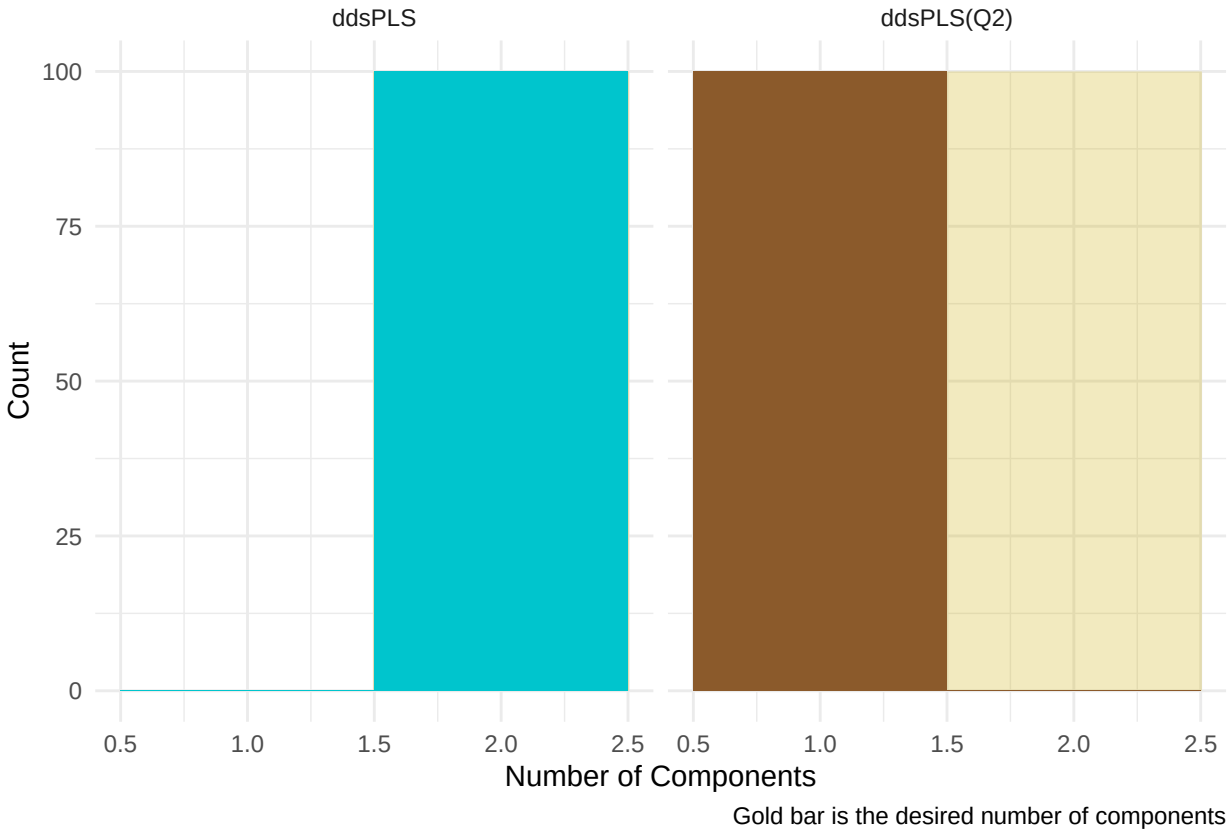

Figure 3.35: Number of components built by parameter selection metric with $q = 25$.

This plot confirms our suspicion that model built using Q^2 have problems accurately capturing the structure of the data as it always build too few components. This behavior is somewhat surprising but my guess would be that with a large number of uncorrelated responses, using Q^2 is less likely to penalize these responses as this could lead to a lower Q^2 value in trials to select parameters. This leads to the model trying to fit these uncorrelated responses in order to minimize the Q^2 on the training set. When applied to the test set, the relation between these responses and the predictors will likely be different as it is determined only by random noise.

The findings in this section seem to suggest that $R^2 - Q^2$ is actually a better metric for parameter selection as it is more likely to build a model that reflects the structure of data and at worst makes only slightly less accurate predictions than model using Q^2. In cases were there are a large number of uncorrelated responses, using Q^2 leads to significantly worse model performance than even poorly performing models built using $R^2 - Q^2$.

Conclusion

Ultimately, ddsPLS delivers on many of its promises, offering an improvement over existing sparse PLS models. As expected, in simpler cases with fewer variables and responses NIPALS-PLS still performs better in regards to common statistics used to evaluate model performance. However, as model complexity increases especially in regards to the number of predictors or responses that are just noise, ddsPLS begins to outperform NIPALS-PLS. In almost all cases, ddsPLS outperforms sPLS.

ddsPLS also does an good job at building models that reflect the structure of the underlying data. NIPALS-PLS tends to build too many components while sPLS tends to build too few. As promised, this shows that ddsPLS is often better at identifying the underlying structure of data than comparable methods.

In terms of parameter selection, using $R^2 - Q^2$ as a metric leads to models that perform better than those built using Q^2. These models offer at worst comparable predictive power and slightly better capture underlying structure. In addition, models built using Q^2 have much more variable performance and in some cases fail to make good predictions or describe the original data.

One persistent problem that ddsPLS exhibits not as prevalent in sPLS is higher variance in the models built. In many cases, ddsPLS exhibits a significant probability of building models that don't include part of the data's structure. In order to remedy issues regarding inconsistent performance, it may be helpful to develop a procedure using other more consistent methods of PLS to find cases where this error occurs.

There are still a number of questions surrounding ddsPLS that should be pursued. As discussed, during repeated trials of the ddsPLS model, we would often see a number of models that exhibited noticeably lower R^2 and Q^2 values while $R^2 - Q^2$ was close to 0. Since the data on which the model was built was not saved, it would be helpful to find what features of the data cause this behavior to take place. It may be that their was simply more noise present in these trials or that there was more random correlation in the noise that caused the model to struggle. In these cases, it would be helpful to find out what levels of noise or random correlation cause these problems to

occur. We may also want to run more test on a variety of sample sizes to see if this problem is less persistent at larger sample sizes.

Given that $R^2 - Q^2$ served as a better metric for parameter selection, I would also recommend seeing how it performs as a metric for other sparse partial least squares models and similar model types. It may be that using this metric only works well due to unique qualities of the ddsPLS algorithm however, this is worth further investigation.

Furthermore, we may want to assess the performance of ddsPLS on real world data. To my knowledge, there is no published work in which the model is used on real world data. While offering improved performance on simulated data is a positive sign, ultimately we want a model that can be applied to real world problems and deliver compelling results in these cases.

Appendix: R Code

This appendix contains the main R code used to run simulations included in this thesis. This is not all of the code used to run simulations included, only the general functions. A more complete archive of R code used in this thesis can be found at `https://github.com/HarpethLee/thesis-lee`.

Library Calls

```
library(ddsPLS2)
library(MASS)
library(pls)
library(mixOmics)
```

Data Simulation

```
sim_data <- function(n = 25,
                     p = 50,
                     q = 5,
                     R = 5,
                     noise_weight = 1,
                     D_method = "newest",
                     noise_type = "rnorm",
                     struc = "simple") {

  # Creates A and D matrices

  if(D_method == "complex") {
    R = 13
```

```
}

Row1 <- c(rep(1, 5), rep(0, p - 5))
Row2 <- c(rep(0, 5), rep(1, 10), rep(0, p - 15))

reps1 <- round(3*R/5)
reps2 <- R - reps1

A1 <- do.call("rbind", replicate(reps1, Row1, simplify = FALSE))
A2 <- do.call("rbind", replicate(reps2, Row2, simplify = FALSE))

A <- rbind(A1, A2)


if(struc == "complex") {
  R = 13

  Row1 <- c(rep(1, 5), rep(0, p - 5))
  Row2 <- c(rep(0, 5), rep(1, 10), rep(0, p - 15))
  Row3 <- c(rep(0, 15), rep(1, 5), rep(0, p - 20))
  Row4 <- c(rep(0, 20), 1, rep(0, p - 21))
  Row5 <- c(rep(0, 21), rep(1, 2), rep(0, p - 23))

  A1 <- do.call("rbind", replicate(3, Row1, simplify = FALSE))
  A2 <- do.call("rbind", replicate(2, Row2, simplify = FALSE))
  A3 <- do.call("rbind", replicate(3, Row3, simplify = FALSE))
  A4 <- do.call("rbind", replicate(2, Row4, simplify = FALSE))
  A5 <- do.call("rbind", replicate(3, Row5, simplify = FALSE))

  A <- rbind(A1, A2, A3, A4, A5)

}


if(D_method == "new") {
```

```
  D <- matrix(rep(1, R*q), nrow = R)

} else if(D_method == "diag") {

  D <- diag(max(q, R))[1:R, 1:q]

} else if(D_method == "simple") {

  D <- cbind(rep(1, R), matrix(rep(0, R*(q-1)), nrow = R))

} else if(D_method == "complex") {


  D1 <- c(rep(1, 3), rep(0, 10))
  D2 <- c(rep(0, 3), rep(1, 3), rep(0, 7))
  D3 <- c(rep(0, 6), rep(1, 3), rep(0, 4))
  D4 <- c(rep(0, 9), rep(1, 3), rep(0, 1))
  D5 <- matrix(rep(0, R*(q-4)), nrow = R)

  D <- cbind(D1, D2, D3, D4, D5)

} else {
   q_s <- round(q/4)

   if(q_s == 0) {
     q_s = 1
   }

   Row1D <- c(rep(1, q_s), rep(0, q - q_s))
   Row2D <- c(rep(0, q_s), rep(1, q_s), rep(0, q - 2*q_s))

   reps1D <- round(3*R/5)
   reps2D <- R - reps1D

   D1 <- do.call("rbind",
                 replicate(reps1D, Row1D, simplify = FALSE))
```

```
    D2 <- do.call("rbind",
                  replicate(reps2D, Row2D, simplify = FALSE))

    D <- rbind(D1, D2)
}

d <- ncol(A)+nrow(A)+ncol(D)
psi <- MASS::mvrnorm(n = n,mu = rep(0,d),Sigma = diag(d))
phi <- psi[,1:nrow(A)]

phi <- matrix(rnorm(n*R), nrow = n)

# This section creates the random noise added to the data.
# If `rnorm` is used to generate noise a lower noise weight
# should be used as the function is more sensitive since
# we directly weight results and not the covariance matrix.

if(noise_type == "mvrnorm") {
  epsilon_X <- mvrnorm(n = dim(phi)[1],
                       rep(0, dim(A)[2]),
                       Sigma = noise_weight*diag(dim(A)[2]))

  epsilon_Y <- mvrnorm(n = dim(phi)[1],
                       rep(0, dim(D)[2]),
                       Sigma = noise_weight*diag(dim(D)[2]))
} else {
 epsilon_X <- matrix(noise_weight*rnorm(n = n*p),
                       nrow = n)
 epsilon_Y <- matrix(noise_weight*rnorm(n = n*q),
                       nrow = n)
}

# Creates and returns the X and Y matrices in a list

X <- phi %*% A + epsilon_X
Y <- phi %*% D + epsilon_Y
```

```
  list(X=X, Y=Y)
}
```

Main Simulation Function

```
pls_test <- function(n,
                     sim,
                     func,
                     passed_arg,
                     criterion = "diffR2Q2") {

  # Splits into training and test
  in_train <- round(n/3)
  in_test <- round(2*n/3)

  split <- sample(c(rep(0, in_train),
                    rep(1, in_test)))

  sim_train_X <- sim$X[split == 0, ]
  sim_train_Y <- sim$Y[split == 0, ]

  sim_test_X <- sim$X[split == 1, ]
  sim_test_Y <- sim$Y[split == 1, ]

  # Generates model using the training set and predicts RMSE along
  # with other statistics.

  if(func == "ddsPLS") {
    mod <- ddsPLS(sim_train_X,
                  sim_train_Y,
                  criterion = criterion)

    preds <- predict(mod, sim_test_X)
    preds_ib <- predict(mod, sim_train_X)
```

```
mse <- sum((preds$y_est - sim_test_Y)^2)/nrow(sim_test_Y)
rmse <- sqrt(mse)

pred_mean_tr <- t(replicate(nrow(sim_train_Y),
                            colMeans(sim_train_Y)))

RSS <- sum((preds_ib$y_est - sim_train_Y)^2)
TSS <- sum((sim_train_Y - pred_mean_tr)^2)
R2 <- 1 - (RSS/TSS)

pred_mean_ts <- t(replicate(nrow(sim_test_Y),
                            colMeans(sim_test_Y)))

QSS <- sum((preds$y_est - sim_test_Y)^2)
TSS_Q <- sum((sim_test_Y - pred_mean_ts)^2)
Q2 <- 1 - (QSS/TSS_Q)

ncomp <- mod$R
} else if(func == "pls") {

df <- data.frame(X = I(sim_train_X), Y = I(sim_train_Y))
mod <- plsr(Y~X,
            data = df,
            ncomp = 10,
            method = "oscorespls",
            validation = "CV",
            scale = TRUE)

R2 <- R2(mod)
Q2 <- colSums(R2$val, dims = 2)

ncomp <- which(Q2 == max(Q2)) - 1

ncomp <- unname(ncomp)
```

```
    if(ncomp == 0){
      preds <- t(replicate(nrow(sim_test_Y),
                           colMeans(sim_train_Y)))
      preds_ib <- t(replicate(nrow(sim_train_Y),
                              colMeans(sim_train_Y)))
    } else {
      preds <- predict(mod, sim_test_X)[,,ncomp]
      preds_ib <- predict(mod, sim_train_X)[,,ncomp]
       }

    mse <- sum((preds - sim_test_Y)^2)/nrow(sim_test_Y)
    rmse <- sqrt(mse)

    pred_mean_tr <- t(replicate(nrow(sim_train_Y),
                                colMeans(sim_train_Y)))

    RSS <- sum((preds_ib - sim_train_Y)^2)
    TSS <- sum((sim_train_Y - pred_mean_tr)^2)
    R2 <- 1 - (RSS/TSS)

    pred_mean_ts <- t(replicate(nrow(sim_test_Y),
                                colMeans(sim_test_Y)))

    QSS <- sum((preds - sim_test_Y)^2)
    TSS_Q <- sum((sim_test_Y - pred_mean_ts)^2)
    Q2 <- 1 - (QSS/TSS_Q)

  } else {
    colnames(sim_train_X) <- c(1:ncol(sim_train_X))
    colnames(sim_test_X) <- c(1:ncol(sim_train_X))
    colnames(sim_test_Y) <- c(1:ncol(sim_test_Y))
    colnames(sim_train_Y) <- c(1:ncol(sim_test_Y))

    tune <- tune.spls(sim_train_X, sim_train_Y,
                      validation = "Mfold",
```

```
                    folds = 10,
                    ncomp = 10,
                    mode = "regression")
  ncomp <- tune$choice.ncomp

  mod <- spls(sim_train_X,
              sim_train_Y,
              ncomp = ncomp,
              mode = "regression")


  if(ncomp == 0){
    preds <- t(replicate(nrow(sim_test_Y),
                         colMeans(sim_train_Y)))
    preds_ib <- t(replicate(nrow(sim_train_Y),
                            colMeans(sim_train_Y)))
    } else {
      preds <- predict(mod, sim_test_X)
      preds <- preds$predict[,,ncomp]

      preds_ib <- predict(mod, sim_train_X)
      preds_ib <- preds_ib$predict[,,ncomp]
     }

  mse <- sum((preds - sim_test_Y)^2)/nrow(sim_test_Y)
  rmse <- sqrt(mse)

  pred_mean_tr <- t(replicate(nrow(sim_train_Y),
                              colMeans(sim_train_Y)))

  RSS <- sum((preds_ib - sim_train_Y)^2)
  TSS <- sum((sim_train_Y - pred_mean_tr)^2)
  R2 <- 1 - (RSS/TSS)

  pred_mean_ts <- t(replicate(nrow(sim_test_Y),
                              colMeans(sim_test_Y)))
```

```
    QSS <- sum((preds - sim_test_Y)^2)
    TSS_Q <- sum((sim_test_Y - pred_mean_ts)^2)
    Q2 <- 1 - (QSS/TSS_Q)
  }

  # Returns the results as a vector

  out <- c(passed_arg, ncomp, rmse, R2, Q2, R2-Q2)

  return(out)
}
```

Noise Test Function

```
noise_eval <- function(noise_weight,
                       func = "ddsPLS",
                       n = 300,
                       p = 100,
                       q = 5){

  sim <- sim_data(n = n,
                  p = p,
                  q = q,
                  noise_weight = noise_weight,
                  noise_type = "rnorm",
                  struc = "complex")

  pls_test(n = n,
           sim = sim,
           func = func,
           passed_arg = noise_weight)

}
```

Predictors Test Function

```
p_eval <- function(p,
                   noise_weight = 1,
                   n = 150,
                   q = 5,
                   func = "ddsPLS",
                   struc = "complex",
                   D_method = "newest",
                   criterion = "diffR2Q2"){

   # Randomly simulates data
   sim <- sim_data(n = n,
                   p = p,
                   q = q,
                   noise_weight = noise_weight,
                   noise_type = "rnorm",
                   struc = struc,
                   D_method = D_method)

   # Passes Data to test function
   pls_test(n = n,
            sim = sim,
            func = func,
            passed_arg = p,
            criterion = criterion)
}
```

Responses Test Function

```
q_eval <- function(q,
                   noise_weight = 0.5,
                   n = 150,
                   p = 100,
                   func = "ddsPLS",
```

```
                    struc = "simple",
                    D_method = "simple"){

  ## Note that if D_method != "simple" there are lower bounds on the
  # value of q

   # Randomly simulates data
   sim <- sim_data(n = n,
                   p = p,
                   q = q,
                   noise_weight = noise_weight,
                   noise_type = "rnorm",
                   struc = struc,
                   D_method = D_method)

   # Passes Data to test function
   pls_test(n = n,
            sim = sim,
            func = func,
            passed_arg = q)
}
```

References

Chun & Keles (2010), Chun & Keles (2007), Cloarec (2014), Efron (1979), Friedman, Hastie, & Tibshirani (2010), Geisser (1974), Hastie, Tibshirani, & Friedman (2009), Helland & Almøy (1994), Hu, Liu, Liu, & Xia (2022), Janitza & Hornung (2018), Johnstone & Yu Lu (2004), Kalina & Schlenker (2015), Kohavi (1995), Lê Cao, Rossouw, Robert-Granié, & Besse (2008), Liu, Trinchera, Tenenhaus, Wei, & Hero (2013), Lorenzo, Saracco, & Rodolphe (2010), Manne, Pell, & Ramos (2009), Stone (1974), Sutton, Thiébaut, & Liquet (2018), Tibshirani (1995), Wold, Sjöström, & Eriksson (2001), Wold et al. (2009), Zou & Hastie (n.d.), R Core Team (2021), Lorenzo (2022), Wickham, François, Henry, & Müller (2022), Wickham (2016), Aphalo (2022), Auguie (2017), Iannone, Cheng, & Schloerke (2022), Meschiari (2022), Venables & Ripley (2002), F, B, A, & K-A (2017), Liland, Mevik, & Wehrens (2021) ...

Aphalo, P. J. (2022). *Ggpmisc: Miscellaneous extensions to 'ggplot2'*. Retrieved from `https://CRAN.R-project.org/package=ggpmisc`

Auguie, B. (2017). *gridExtra: Miscellaneous functions for "grid" graphics*. Retrieved from `https://CRAN.R-project.org/package=gridExtra`

Chun, H., & Keles, S. (2007). Sparse Partial Least Squares Regression with an Application to Genome Scale Transcription Factor Analysis.

Chun, H., & Keles, S. (2010). Sparse partial least squares regression for simultaneous dimension reduction and variable selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *72*(1), 3–25. http://doi.org/10.1111/j.1467-9868.2009.00723.x

Cloarec, O. (2014). Can we beat over-fitting?: Can we beat over-fitting? *Journal of Chemometrics*, *28*(8), 610–614. http://doi.org/10.1002/cem.2602

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, *7*(1). http://doi.org/10.1214/aos/1176344552

F, R., B, G., A, S., & K-A, L. C. (2017). mixOmics: An r package for 'omics feature

selection and multiple data integration. *PLoS Computational Biology*, *13*(11), e1005752. Retrieved from http://www.mixOmics.org

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, *33*(1), 1–22. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2929880/

Geisser, S. (1974). A Predictive Approach to the Random Effect Model. *Biometrika*, *61*(1), 101–107. Retrieved from https://www.jstor.org/stable/2334290

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed). New York, NY: Springer.

Helland, I. S., & Almøy, T. (1994). Comparison of Prediction Methods when Only a Few Components are Relevant. *Journal of the American Statistical Association*, *89*(426), 583–591. http://doi.org/10.1080/01621459.1994.10476783

Hu, J., Liu, X., Liu, X., & Xia, N. (2022). Some aspects of response variable selection and estimation in multivariate linear regression. *Journal of Multivariate Analysis*, *188*, 104821. http://doi.org/10.1016/j.jmva.2021.104821

Iannone, R., Cheng, J., & Schloerke, B. (2022). *Gt: Easily create presentation-ready display tables*. Retrieved from https://CRAN.R-project.org/package=gt

Janitza, S., & Hornung, R. (2018). On the overestimation of random forest's out-of-bag error. *PloS One*, *13*(8), e0201904. http://doi.org/10.1371/journal.pone.0201904

Johnstone, I. M., & Yu Lu, A. (2004). Sparse Principal Components Analysis.

Kalina, J., & Schlenker, A. (2015). A Robust Supervised Variable Selection for Noisy High-Dimensional Data. *BioMed Research International*, *2015*, 1–10. http://doi.org/10.1155/2015/320385

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.

Lê Cao, K.-A., Rossouw, D., Robert-Granié, C., & Besse, P. (2008). A Sparse PLS for Variable Selection when Integrating Omics Data. *Statistical Applications in Genetics and Molecular Biology*, *7*(1). http://doi.org/10.2202/1544-6115.1390

Liland, K. H., Mevik, B.-H., & Wehrens, R. (2021). *Pls: Partial least squares and principal component regression*. Retrieved from https://CRAN.R-project.org/package=pls

Liu, T.-Y., Trinchera, L., Tenenhaus, A., Wei, D., & Hero, A. O. (2013). Globally Sparse PLS Regression. In H. Abdi, W. W. Chin, V. Esposito Vinzi, G. Russolillo, & L. Trinchera (Eds.), *New Perspectives in Partial Least Squares and Related Methods* (pp. 117–127). New York, NY: Springer. http://doi.org/10.1007/978-1-4614-8283-3_7

Lorenzo, H. (2022). *ddsPLS2: Data-driven sparse PLS 2*.

Lorenzo, H., Saracco, J., & Rodolphe, T. (2010). Data-driven sparse partial least squares. *Stat. Anal. Data Min.: ASA Data Sci. J.*, 1–19. http://doi.org/10.1002/sam.11558

Manne, R., Pell, R. J., & Ramos, L. S. (2009). The PLS model space: The inconsistency persists. *Journal of Chemometrics*, *23*(2), 76–77. http://doi.org/10.1002/cem.1181

Meschiari, S. (2022). *latex2exp: Use LaTeX expressions in plots*. Retrieved from `https://CRAN.R-project.org/package=latex2exp`

R Core Team. (2021). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from `https://www.R-project.org/`

Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, *36*(2), 111–133. http://doi.org/10.1111/j.2517-6161.1974.tb00994.x

Sutton, M., Thiébaut, R., & Liquet, B. (2018). Sparse partial least squares with group and subgroup structure: Sparse partial least squares with group and subgroup structure. *Statistics in Medicine*, *37*(23), 3338–3356. http://doi.org/10.1002/sim.7821

Tenenhaus, M. (1998). *La regression PLS: Theorie et pratique*. Editions Technip.

Tibshirani, R. (1995). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *58*(1), 267–288. Retrieved from `https://www.jstor.org/stable/2346178?seq=1#metadata_info_tab_contents`

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth). New York: Springer. Retrieved from `https://www.stats.ox.ac.uk/pub/MASS4/`

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from `https://ggplot2.tidyverse.org`

Wickham, H., François, R., Henry, L., & Müller, K. (2022). *Dplyr: A grammar of data manipulation*. Retrieved from `https://CRAN.R-project.org/package=dplyr`

Wold, S., Høy, M., Martens, H., Trygg, J., Westad, F., MacGregor, J., & Wise, B. M. (2009). The PLS model space revisited. *Journal of Chemometrics*, *23*(2), 67–68. http://doi.org/10.1002/cem.1171

Wold, S., Sjöström, M., & Eriksson, L. (2001). PLS-regression: A basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, *58*(2), 109–130. http://doi.org/10.1016/S0169-7439(01)00155-1

Zou, H., & Hastie, T. (n.d.). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2),

301–320. Retrieved from https://www.jstor.org/stable/3647580