Curse of Dimensionality

Jinyi Che, Rindala Fayyad, Daniel Herrera,

Yuning Liu, Lauren Mock, Yishan Zheng

December 16, 2021

1 Introduction

The curse of dimensionality refers to issues that occur specifically when utilizing high dimensional data. The term curse of dimensionality originated from Richard E. Bellmen's exploration of dynamic programming in 1957 *(Bellman, 1957)*. The dimensionality of a dataset refers to the number of predictors/features included in the model. The topic of the curse of dimensionality grew in popularity with the rise of computational power and machine learning. While the effects of the curse of dimensionality have important ramifications for machine learning methods such as K-nearest neighbors, Bayesian models, and others, the scope of the problem extends far beyond those topics into other aspects of the data and statistical framework. The focus of our simulation study will be to explore the effects of the curse of dimensionality within the classical statistical method of linear regression.

Linear regression is a commonly used modeling procedure for predicting a target variable from a set of predictor variables. In a linear regression model, the response variable, typically denoted by y_i, is a function of the predictors and beta coefficient estimates:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_d x_{id} + \epsilon$$

Ordinarily least squares (OLS) is a type of method used in linear regression to estimate the Beta coefficients. If the errors are independently identically distributed (normally distributed errors assumption met), then OLS is identical to the maximum likelihood estimator (Hayashi, 2000). OLS utilizes the following matrix function to derive estimates for the beta coefficients.

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

The method of OLS works to efficiently solve the beta coefficient estimates by minimizing the sum of the squared difference between observed and predicted values. However, this method runs into two main issues when working with high-dimensional data. These issues arise out of the curse of dimensionality within the context of linear regression. The curse of dimensionality occurs commonly in the study of genomic data, where there are a very large number of variables at play for any given individual. Our simulation study replicated a situation such as this by using randomly uniformly distributed predictor variables. In this context, we can presume our model is looking at variables such as mutations in a genomic sequence, which can be assumed to follow a uniform distribution. In this domain, it is quite common to run into issues of high dimensionality; however, it is important to note that this situation can occur across various domains and applies more broadly than this specific case which is used for illustrative purposes.

A consequence of having the number of predictors d greater than the sample size n is that the matrix X^TX is no longer invertible. When this occurs, the OLS estimates can no longer be calculated.

Another issue of large dimensionality problems in the context of linear regression concerns the relationship between our predictor variables as d increases. This is less discussed and not necessarily an explicit part of our simulation study; however, the topic deserves to

be addressed. When d increases, we are more likely to express multicollinearity between the predictors of our dataset. Multicollinearity refers to the ability to accurately predict one predictor variable from another — often accompanied by high correlations between predictors. This phenomenon stems from a redundancy in the explanatory power of our predictor variables and often leads to nonsensical estimates in our linear regression model, since it makes the condition of holding other multicollinear variables constant unachievable.

In either case mentioned above, there exist several methods to handle the curse of dimensionality at different stages of the statistical process. Principal component analysis is a popular technique that reduces the dimensionality of highly dimensional datasets. Regularization methods such as Lasso, Ridge, and Elastic Net regularize, or constrain, the estimated coefficients towards 0 by using a penalty term, denoted by λ. Preventative methods such as forward and backwards step selection reduce the number of predictor variables included in the modeling process by utilizing criteria such as adjusted R^2 to determine relevant predictors. Additionally, the removal of variables that are a direct combination of other predictor variables can reduce the number of predictors in a model.

2 Methods

Our simulation is performed in batches using the R language. In each batch, we simulated 1000 datasets, each with d predictor variables and one response variable, Y. A total of 6 batches of simulations are run, where $d = 1$, 50, 97, 98, 99, and 100. For each batch, the first step is to generate datasets to which we fit our model. As our analysis is in the linear regression context, we generate data points and their corresponding response value in each batch from a given underlying function in the form of $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_d x_d + \epsilon$. The true beta vector $[\beta_0, \beta_1, \ldots, \beta_d]$, which we will use as the underlying distribution of our datasets,

is generated by sampling from a uniform distribution and is fixed for each round of the 1000 simulations in this batch. In each round, the predictor variables are generated in the form of a matrix X with size n x d, where d denotes the number of predictors and n denotes the number of observations. Each of $n = 100$ data points is generated from a jointly uniform distribution of dimension d. We then added random noise to our predictor matrix X. The noise vector is sampled from a normal distribution of dimension d with mean 0 and standard deviation 10. Finally, the response variable Y is calculated from the underlying function by multiplying the predictor matrix with our fixed beta vector and adding the noise. This way, we ensure that the data will not result in a perfect fit and at the same time the fit is still driven by the underlying function instead of the noise.

The second step in each round is to fit a linear model to the simulated dataset. We use the `lm` function in R to perform multiple linear regression. The predictor variables passed to this function are all columns in our predictor matrix X, and the response variable is our Y vector. The call to the `lm` function returns a linear model fitted with the OLS method. We save the estimated coefficients of the model, which we expect to roughly match our true beta coefficients, in a row of a matrix and repeat this process 1000 times. When all 1000 rounds in a batch are complete, we get a matrix with estimated coefficients from 1000 models each fitting data generated from the same underlying function. This concludes a batch. For each value of d, the variances of each of the $d + 1$ estimators are then estimated from all models fitted in this batch. Since the OLS method is the same as maximum likelihood estimation in a linear regression setting, the resulting estimators are unbiased by default. Therefore, the MSE of each estimator is simply equal to the variance. These MSE values are used as our primary measure for the influence of high dimensionality, with the purpose of this study being to observe the effect on the parameters' MSE when we increase $d + 1$, the number of parameters in each regression model. For the sake of simplicity, we used β_1 and β_{26} for our observation.

3 Results

For $d = 1$, we get two estimators, $\hat{\beta}_0$ (the intercept) and $\hat{\beta}_1$ (the slope). The table below shows the results for the first 6 out of the 1000 rounds of simulation we conducted for $d = 1$:

	$\hat{\beta}_0$	$\hat{\beta}_1$
Round 1	3.207981	9.648855
Round 2	5.133416	7.478920
Round 3	4.513584	6.583951
Round 4	2.317853	9.855818
Round 5	2.590637	6.551960
Round 6	12.767736	3.369407

We then calculated the variance of each estimator over all 1000 simulations. The variance of each estimator in the case where d = 1 are shown below:

$Var[\hat{\beta}_0]$	$Var[\hat{\beta}_1]$
2.35433	8.374659

We then repeated this process for $d = 50, d = 75, d = 97, d = 98$ and $d = 99$. There are $d + 1$ beta estimates for each of the 1000 simulations for each value of d. The table below presents the mean square error (MSE) of the first eight beta estimates for each value of d.

	$MSE(\beta_0)$	$MSE(\beta_1)$	$MSE(\beta_2)$	$MSE(\beta_3)$	$MSE(\beta_4)$	$MSE(\beta_5)$	$MSE(\beta_6)$	$MSE(\beta_7)$
d = 50	322.21	8.16	8.45	8.45	8.05	8.25	8.35	8.51
d = 75	1063.15	8.34	8.29	8.86	7.96	8.36	8.14	7.91
d = 97	26033.62	8.60	8.38	8.38	8.63	8.02	8.29	8.59
d = 98	134008.95	8.49	8.67	8.26	8.47	8.51	8.95	8.42
d = 99	13949652.14	405.74	658.07	103.85	105.57	853.82	1271.07	528.76

In fact, these estimators are the ordinary least squares estimators, which are unbiased estimators of the unknown parameters in each linear regression model. This means that the bias of the estimators is equal to zero, and the MSE of the estimators is simply equal to the variance.

$$\begin{aligned} MSE(\hat{\beta}) &= Var(\hat{\beta}) + Bias(\hat{\beta})^2 \\ &= Var(\hat{\beta}) + 0 \\ &= Var(\hat{\beta}) \end{aligned}$$

In other words, the variance values in the table above also represent the MSE of the estimators.

We notice that for $d = 50, 75, 97, 98$, the MSE values of the slope estimators are always around 8, which is due to the underlying distribution of the data we simulated; all of the covariates follow a uniform distribution Unif(0,100). The variance of a uniform distribution (a, b) is $(b - a)^2/12$.

In this case, the variance is 8.3, which makes sense given our results. As such, we would not expect the MSE of one slope estimator to be significantly different from the MSE of another slope estimator. The variance of the intercept estimator increases as d increases

because it accounts for the random noise that we added to our simulation, so it is no longer uniformly distributed.

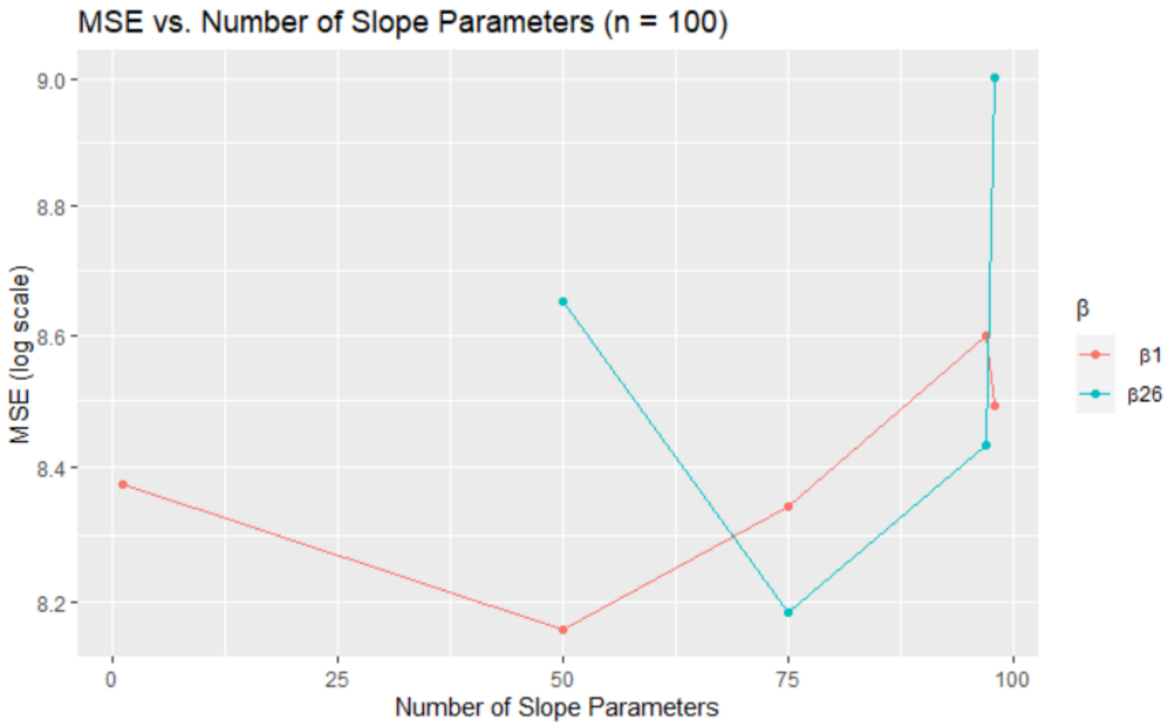

Figure 1: MSE calculations for the estimates of β_1 and β_{26} with d up to 98.

The graph above shows the different MSE values of $\hat{\beta}_1$ and $\hat{\beta}_{26}$ for $d = 50, 75, 97, 98$. We notice that the MSEs of the estimators are always somewhere between 8 and 9 for these values of d. There is not any particular trend for the values of these MSEs as they fluctuate between 8 and 9. But it is important to note that although there is not any particular trend here for the MSE, it is always taking a "small" value for $d < 99$, which tells us that the estimators are still somewhat reliable even for $d = 98$. Later, we will talk about what happens when $d \geq 99$.

However, the MSE of the intercept estimator takes on completely different values as d increases. For every increase in the number of parameters d, the MSE of the intercept esti-

mator increases significantly: for $d = 1$, the MSE of the intercept estimator is equal to 12.35, but for $d = 99$, it is approximately equal to 1.39×10^7. Hence, as the number of parameters increases, the intercept estimator becomes less and less consistent.

For $d = 99$, when the number of estimators is equal to the sample size, the MSEs of the estimators increase drastically, and consistency is completely lost. In this case, the estimators are no longer reliable and they are no longer mathematically stable. For such a large number of parameters, we would need a very large sample size to get statistically reliable, accurate and precise estimators again.

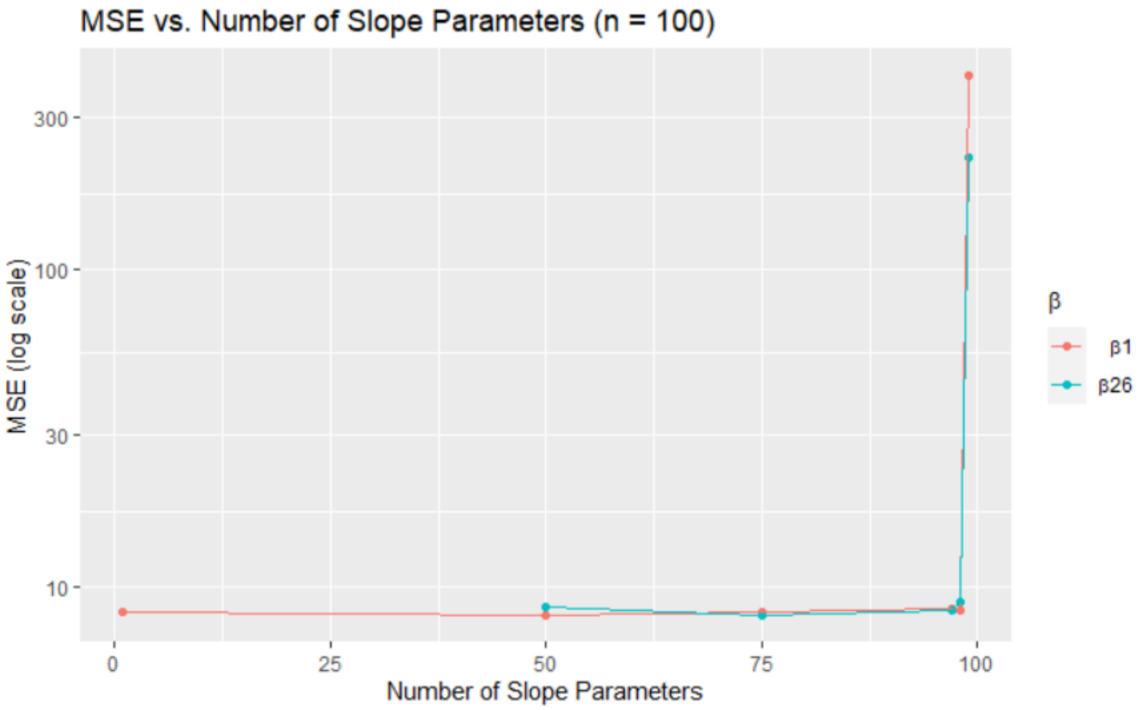

Figure 2: MSE calculations for the estimates of β_1 and β_{26} including d = 99.

In the graph above we show how the MSEs of $\hat{\beta}_1$ and $\hat{\beta}_{26}$ change depending on the number of parameters. The y-axis was transformed using a $\log_{10}$ scale to account for the very large

values of the MSE of the intercept estimator. This graph shows how significantly the MSEs of the estimators increase when $d = 99$. With 99 parameters for each predictor and one parameter for the slope, this is the point at which the number of parameters exactly equals the number of observations ($n = 100$). For visual purposes, we did not plot the MSE of the intercept, but the table above shows how much larger the MSE of the intercept parameter is compared to the MSE of the slope parameters.

For $d = 100$ (1 intercept and 100 slopes), our code in R does not run, because it is unable to generate beta estimators for a linear regression model. This is due to the way the beta estimators are constructed. Theoretically,

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

where X is an $n \times d$ matrix of covariates. When $d \geq n$, then the rank of X is less than d. This means that there exists a vector $\boldsymbol{v}$ belonging to $\mathbb{R}^d$ such that $X\boldsymbol{v} = 0$. Then $X^TX\boldsymbol{v} = 0$, which implies that X^TX is not invertible. Consequently, the parameters can no longer be estimated.

4 Conclusion and Discussion

In the regression model that used ordinary least squares (identical to maximum likelihood estimator) to estimate beta coefficients, we found out that we could get constant and valid results when the number of estimated parameters $d + 1$ is at most one less than the number of observations n. The MSE of the intercept parameter increased smoothly overall, and the MSEs of slope parameters does not change with our uniformly distributed data. When d is one less than n, where number of estimators equal to the sample size, the MSE will increase dramatically with our uniform distributed data. We could not get results when d is equal to n because of the way that the $\boldsymbol{\beta}$ estimator was constructed.

With the result we found, ways to solve the high dimensionality problem could be further investigated. There are also many real-life examples where the curse of dimensionality may occur. For example, when reading a digital camera image, there are millions of pixels that will be recorded as data points, and one hour video will contain more than 130000 images. Also, in a business setting, when trying to figure out customers' preference, Netflix needs to deal with data from millions of users rating about thousands of movies. The majority of machine learning and natural language processing algorithms could be used to deal with huge numbers of variables that may lead to the low accuracy and recall of the model. The machine learning methods could be supervised or unsupervised depending on whether the label of class matters *(Delon, n.d.)*.

There are mainly three ways of solving the high dimensionality problem. First, using parsimonious models where we manually select certain features in the model using expert knowledge. For example, when predicting the weather, the population data could be dismissed. This could be cost-effective. Second is the dimension reduction. Lots of machine learning models could be used, and the principal components analysis (PCA) is the most popular one. The idea of PCA is by creating a separating hyperplane and mapping all data to a two dimensions data set. In addition, multidimensional scaling, locally linear embedding and many other models could also be used depending on the data set. However, lots of them have problems dealing with certain number of features. Third is regularization where the parameters estimated are unstable, so the regularization like Lasso, Ridge, and Elastic Net regularization could help making correct estimation. Each way has its own pros and cons, the method should be chosen based on the feature of the data set *(Oliinyk, 2018)*.

Using the methods mentioned above, they could help achieve better model generalization and make better cross-validation results.

5 References

[1] Bellman, Richard Ernest; Rand Corporation (1957). Dynamic programming. Princeton University Press. p. ix. ISBN 978-0-691-07951-6. Delon, J. (n.d.). The curse of dimensionality. Paris; Université Paris Descartes.

[2] Hayashi, Fumio (2000). Econometrics. Princeton University Press. p. 15.

[3] Oliinyk, H. (2018, March 20). Why and how to get rid of the curse of dimensionality right (with breast cancer dataset...Medium. Retrieved December 6, 2021, from https://towardsdatascience.com/why-and-how-to-get-rid-of-the-curse-of-dimensionality-right-with-breast-cancer-dataset-7d528fb5f6c0.

6 Appendix

Code: https://github.com/rindalafayyad17/222-Project/blob/main/project.Rmd