Fikos Dimitrios
Non-confidential

GRADUATION INTERNSHIP REPORT

INTERNSHIP SUBJECT:

Local Ordinary Least Squares for Dimension Reduction in Regression

INTERNSHIP LOCATION: ENSAI, 51 Rue Blaise Pascal

CITY: Rennes **COUNTRY:** France

Academic Year 2023-2024

Internship Supervisors: François Portier, Ikko Yamane

ENSAI Internship Advisor: François Portier

Abstract

Dimension reduction has become a popular topic due to the explosion of data volumes in the 21^{st} century. Several approaches have been proposed for presenting a dataset into a lower representation. This research is motivated by the problem of estimating gradients at a point, by the aid of k Nearest Neighbors. We consider a typical regression setup and use as a starting point the local version of OLS followed by the idea of adding the Lasso regularization term to OLS. In addition, two new methods are defined by using the covariance matrix. These methods take advantage of the diagonal elements and the trace of the covariance matrix respectively. Numerical experiments have been carried out with the aim of making a comparative analysis in terms of both efficiency and computational time. The novel algorithms have shown the ability to complete the procedure faster while reaching the efficiency of the OLS method.

Contents

1. Introduction

1.1. Definition and main challenges

The purpose of this research is to investigate the deep universe of Dimension Reduction, a crucial technique in modern Data Science and Machine Learning which objective is to simplify complex datasets with millions or billions of features, while preserving the majority of information within the data. In real life, datasets grow more and more, making it more difficult to treat them thus, pushing scientists to explore new ways for giving reliable solutions. From finance, to health sector and from image processing to video processing the "Big Data", as they are named, are causing new and increasing challenges. However, data always have a 'fundamental dimension' in other words, the least number of features needed to report for the observed attributes, Ahmad and Nassif (2022). This is the topic that scientists seek to investigate.

The challenges associated with this process are numerous. Firstly, processing, analyzing and visualizing extremely large datasets is computationally expensive. This can result in the 'curse of dimensionality', whereby the volume of data increases exponentially. This leads to the phenomenon of data sparsity, which makes traditional machine learning methods less effective at finding meaningful patterns. Furthermore, the probability of these datasets containing irrelevant features is increasing. These features contribute to the complexity of the analysis and may result in erroneous conclusions regarding the insights that can be derived from them. In the context of real-world scenarios, the ability to make prompt and precise decisions is of paramount importance. This is particularly evident in domains such as healthcare and finance, where rapid and accurate decision-making is crucial for optimal outcomes. Therefore, the models constructed for these purposes must be trained expeditiously in order to process the data for real-time applications and to enhance the user experience. A further issue is that of data storage. Despite the exponential growth in hardware capabilities observed in recent times, these remain unable to keep pace with the extreme augmentation of data. In this way, data cannot be treated, or if it can, it requires huge memory capabilities that are not available due to limitations of resources. Ultimately, the utilisation of high-dimensional data may result in the construction of overly intricate models that exhibit a tendency to overfit the training data, thereby impeding the model's capacity for accurate generalisation to novel datasets.

1.2. Existing methods

In the literature there are several ways for doing dimension reduction mostly separated in two categories, unsupervised and supervised techniques. In the unsupervised field, one of the most popular methods is Principal Component Analysis (PCA) see *e.g.* Jolliffe (1990). Its target is to project data onto directions of maximum variance by computing eigenvectors and eigenvalues of the covariance matrix. There can be found a number of difference versions of PCA like simple PCA (see Partridge and Calvo (1997)), sparse PCA see *e.g.* Zou et al. (2006); Mackey (2008) and kernel PCA for non-linear data (see Schölkopf et al. (1997)). Another significant method is the Singular Value Decomposition of a matrix. It is a matrix factorization method that decomposes a matrix in three components and uses the first d vectors taken from the first component (matrix), capturing the fundamental dimension see *e.g.* Wang and Zhu

(2017).

In the supervised perspective, where response variables are taken into consideration, firstly there is Linear Discriminant Analysis (LDA) which focuses on maximizing the separation between classes see *e.g.* Balakrishnama and Ganapathiraju (1998); McLachlan (2005). However, in recent years, a new technique for dimension reduction has emerged as a valuable addition to the scientific toolkit. It involves estimating a gradient of a predictive function in the context of local learning problems, like in regression. Its utility lies in the fact that it focuses on dimensions that capture the most significant local variations between the points close to each other. The objective is to find predictive directions or projections that capture the dataset in a lower dimension. There have been many methods based on that idea, like minimum average variance estimation (see Xia et al. (2002)), kernel-based gradient learning (see Mukherjee et al. (2006)), sparse gradient learning (see Ye and Xie (2012)) and others see *e.g.* Delecroix et al. (2006); Hristache et al. (2001); Dalalyan et al. (2008).

1.3. Proposed approach

The approach of the current research is to treat dimension reduction in regression by using the local version of the Ordinary Least Squares (OLS) to estimate the gradient at a given point. Then, these gradients will be used for applying Singular Value Decomposition and obtain in the end the new vectors-directions that consist of the lower dimension representation of a dataset. Moreover, apart from local OLS, a regularization way will be explored based on the idea of Ausset et al. (2021), followed by two suggestions which take advantage of the covariance between the points in the dataset. There are two main objectives. One is about keeping the efficiency of local OLS or even improve it. Secondly, trying to reduce the number of computations required for it and making the procedure more affordable and practical for applying it to real circumstances and scenarios.

The rest of the report is structured in the following way. Section 2, presents an introduction to regression framework and Ordinary Least Squares method to identify the unknown regression function enriched by use of nearest neighbors. Section 3, displays a detailed analysis of four methods applied for estimating the gradient of a regression function followed by a comparison of their computational requirements. Section 4, remarks two ways for obtaining an estimation of a projection matrix that leads to the dimension reduced dataset. Numerical experiments are conducted in Section 5 followed by graphical illustrations that highlight methods capabilities, while Section 6 discusses some conclusions made, regarding potential limitations as well as some improvements that can be done to the models.

2. Regression background

2.1. Regression framework

Supposing that we have a dataset (X_i, Y_i), $i \in (1, \dots, n)$ where both random variables (X, Y) are defined on the same probabilistic space $(\Omega, \mathbb{F}, \mathbb{P})$ with unknown probability distribution P. The random variable X takes its values in $\mathbb{R}^p$, $X = (x^{(1)}, \dots, x^{(p)})^T$ and it is called independent variable. On the other hand, random variable $Y \in \mathbb{R}$ is called dependent (or response) variable.

The principal objective is to build a prediction mapping g from covariates space $\mathbb{R}^p$ to response space $\mathbb{R}$ such that the prediction error $Y - g(X)$ is as small as possible.
The regression model with regression function $g : \mathbb{R}^p \longmapsto \mathbb{R}$ is given by

$$Y = g(X) + \varepsilon,$$

where ε represents the random noise or error in the model. It is assumed to have zero mean, $\mathbb{E}[\varepsilon] = 0$ and variance $Var(\epsilon) = \sigma^2$. The aim of function $g(X)$ is to predict the true value Y as accurately as possible so as to diminish the error. In other words, it attempts to find the solution of the risk minimization problem $min_g R_p(g)$ where

$$R_p(g) = \mathbb{E}\left[(Y - g(X))^2\right].$$

Assuming that $E\left[Y^2\right] < \infty$ we are looking for the minimum function attained at $g^*(X) = E[Y|X]$ where $g^* : \mathbb{R}^p \to \mathbb{R}$, so that:

$$\mathbb{E}\left[(Y - g^*(X))^2)\right] = \inf_g \mathbb{E}\left[(Y - g(X))^2\right].$$

The performance of empirical risk minimizer has been widely analyzed in the literature see *e.g.* Massart (2007); Lecué and Mendelson (2013). Nevertheless, in the current research, under the assumption that g is differentiable, we want to estimate the gradient $\nabla g(X)$ of this function which is defined as

$$\nabla g : \mathbb{R}^p \longmapsto \mathbb{R}^p,$$

where

$$\nabla g(x) = \left(\frac{\partial g(x^{(1)})}{\partial x^{(1)}}, \ldots, \frac{\partial g(x^{(p)})}{\partial x^{(p)}}\right)^T,$$

with $\partial / \partial x^{(j)}$ standing for partial derivative with respect to $x^{(j)}$.

2.2. Ordinary Least Squares

The core method for approximating the unknown regression function $g(X)$ and its gradient covariates, based on the observed data, is the Ordinary Least Squares method (OLS).
For the estimation of $g(X)$ under the assumption that there is a linear relationship between predictors X and response Y the linear model is:

$$g(X) \approx \alpha + \beta X,$$

where term α is the intercept term shifting the model output to fit the data and the vector of coefficients $\beta = (\beta_1, \ldots, \beta_p) \in \mathbb{R}^p$ corresponds to the effect of each feature in X.
OLS aims to estimate α and β by minimizing the following:

$$(\hat{\alpha}(x), \hat{\beta}(x)) \in \underset{(\alpha,\beta)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} (Y_i - \alpha - \beta^T X_i)^2.$$

The concept of the OLS method relies on the optimization of the sum of squared residuals and it has the following unique solution for estimating β:

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

It is called, the Normal Equation and it is applicable only when matrix $X^T X$ is invertible. These $\hat{\beta}$ estimators correspond to the gradient estimators under the assumption of linearity, since they represent the rate of change of Y depending on each covariate of X. Moreover, they are known for being unbiased of the true vector β, therefore $E[\hat{\beta}] = \beta$ and among linear estimators, $\hat{\beta}$ has the lowest variance. These properties ensure the OLS estimator to be the Best Linear Unbiased Estimator (BLUE).

2.3. Nearest-neighbor Regression

Methods like OLS usually take advantage of the whole dataset for making estimations for a single point. However, it might be more profitable creating estimators based on the nearby points. Nearby points z, from the dataset can be topologically defined by points that belong to a ball, $B(x, r)$ in space, where the center is the point of interest $x \in \mathbb{R}^p$ and $r > 0$ is the radius. Formally,

$$B(x, r) = z \in \mathbb{R}^p : \|x - z\| \leq r.$$

Based on that approach the k-Nearest Neighbors (kNN), see *e.g.* in Devroye et al. (2013); Biau and Devroye (2015), is one of the most popular local methods used in regression, see *e.g.* Song et al. (2017); Jiang (2019) and in our case for the estimation of the gradient at a point. This method is strongly affected by the decision for the k parameter, see *e.g.* Zhang et al. (2018) which defines the number of nearest neighbors taken into account. Small value for k is highly sensitive to noise and may result in high variance, whereas a large value for k can help reduce variance but might also increase bias as estimates rely heavily on points that are further away. In that way, the kNN bandwidth is defined as:

$$\widehat{\tau}_{n,k}(x) := \inf \left\{ \tau \geq 0 : \sum_{i=1}^{n} \mathbb{1}_{B(x,\tau)}(X_i) \geq k \right\},$$

corresponding to the kNN radius $\tau \geq 0$, such that $B(x, \widehat{\tau}_k(x))$ contains the smallest ball with center x, including k points from the sample.
In order to ensure the probabilistic behavior of the above bandwidth several analysts, see *e.g* Ahmad et al. (2024), have used a proposition which points out that the distribution of X has a continuous density f_X with respect to the Lebesgue measure on $\mathbb{R}^p$ and $f_X(x) > 0$.
In particular, it is assumed that there are sufficiently many points around x for the NN procedure to be effective.
The nearest neighbor estimate at point x is given by

$$\hat{g}(x) = \frac{1}{k} \sum_{i:X_i \in B(x,\widehat{\tau}_k(x))} Y_i,$$

where k is the number of neighbors selected and Y_i are the response variables from the X_i neighbors considered for x. This approach can work without assuming an underlying function making it flexible and widely applicable especially when the true relationship is unknown.

3. Gradient estimation

The following section presents an analytical description of the four different methods for estimating the gradient. The first method is the Ordinary Least Squares method enriched with

k nearest neighbor, which serves as the baseline for this research. The second method requires the use of Lasso regularization parameter, which will be added in the local OLS algorithm as described in the paper by Ausset et al. (2021). Finally, two new methods that transform the normal equation and take advantage of the information contained in the diagonal elements and the trace of the covariance matrix are defined.

3.1. Local OLS

The algorithm for estimating gradient $\beta(x) = \nabla g(x)$ using OLS and kNN together is the following:

$$(\hat{\alpha}(x), \hat{\beta}(x)) \in \underset{(\alpha,\beta)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i:X_i\in B(x,\widehat{\tau}_k(x))} (Y_i - \alpha - \beta^T(X_i - x))^2. \tag{1}$$

As it can be seen the main difference with the prototype OLS method is that instead of using the whole dataset for the approximation, only the k-X_i points closer to point x are taken into consideration. Moreover, these points are centralized by the point x which is vital in order to reduce bias in the estimation of β since it minimizes the influence of trends or offsets that are not relevant to the local area.
The first approach, as described in (1), requires the computation of both the intercept $\hat{\alpha}(x)$ and the gradient $\hat{\beta}(x)$. However, in order to avoid redundant calculations, the data are going to be centered and in that way the intercept, for which there is no point of interest currently, is not going to be computed. Hence, firstly the mean values for the input features and the response are computed.

$$\hat{g}(x) = (1/k) \sum_{i\in NN_k(x)} Y_i, \qquad \hat{m}(x) = (1/k) \sum_{i\in NN_k(x)} X_i,$$

where for each $x \in \mathbb{R}^p$, $\hat{g}(x) \in \mathbb{R}$ and $\hat{m}(x) \in \mathbb{R}^p$. The two estimates $\hat{g}(x)$ and $\hat{m}(x)$ can be used to center the variables Y_i and X_i, respectively, leading to, for each i such that $X_i \in B(x, \widehat{\tau}_k(x))$,

$$Y_i^c = Y_i - \hat{g}(x), \qquad X_i^c = X_i - \hat{m}(x).$$

Based on the centered variables, the optimization problem in (1) might be simplified noting that

$$\hat{\beta}_{OLS}(x) = \underset{\beta\in\mathbb{R}^p}{\arg\min} \sum_{i:X_i\in B(x,\widehat{\tau}_k(x))} (Y_i^c - \beta^T X_i^c)^2, \tag{2}$$

meaning that there is no intercept anymore. The corresponding procedure is described in Algorithm 1.

In summary, from a dataset of n elements, we select a point x and try to find its k nearest neighbors. Once found, we calculate the average value of the inputs and outputs to centralize the data and then apply the optimization algorithm. The final output will be a p vector, $\hat{\beta}_{OLS}$, with gradient estimates per dimension.

Algorithm 1 Estimation of $\hat{\beta}_{OLS}$

Input: $(X_1, Y_1), \ldots, (X_n, Y_n) \in \mathbb{R}^p \times \mathbb{R}$, $x \in \mathbb{R}^p$, $n, k \in \mathbb{N}$, $k < n$
for $x \in \mathbb{R}^p$ **do**
  find $NN_k(x)$
  for $(X_i, Y_i) \in NN_k(x), i = 1, \ldots, k$ **do**
    compute: $\hat{m}(x) = (1/k) \sum_{i \in NN_k(x)} X_i$ and $\hat{g}(x) = (1/k) \sum_{i \in NN_k(x)} Y_i$
    compute: $X_i^c = X_i - \hat{m}(x)$ and $Y_i^c = Y_i - \hat{g}(x)$
  end for
  apply OLS optimization: $\underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i: X_i \in B(x, \hat{\tau}_k(x))} (Y_i^c - \beta^T X_i^c)^2$
end for
Return: $\hat{\beta}_{OLS} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)$

3.2. Local OLS with Lasso

The Lasso regularization parameter to the local OLS procedure is useful and strengthens the already existing model see *e.g.* Tibshirani (1996). Lasso term λ plays a crucial role to control the behavior of the model as one of its key characteristics is the introduce of sparsity, since by adding this l_1 penalty it shrinks some coefficients to zero. That allows the model to retain only the most important features and get rid of features that provide us with few information about the data. In other words, Lasso is very useful for dimension reduction where there is the necessity of retaining only the most useful features that provide us with the appropriate insights. The coefficients can be obtained by the upcoming optimization problem:

$$\hat{\beta}_{OLS-Lasso} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i: X_i^c \in B(x, \hat{\tau}_k(x))} (Y_i^c - \beta^T X_i^c)^2 + \lambda \|\beta\|_1. \tag{3}$$

Algorithm 2 Estimation of $\hat{\beta}_{OLS-Lasso}$

Input: $(X_1, Y_1), \ldots, (X_n, Y_n) \in \mathbb{R}^p \times \mathbb{R}$, $x \in \mathbb{R}^p$, $n, k \in \mathbb{N}$, $k < n$, $\lambda \in \mathbb{R}^*$
for $x \in \mathbb{R}^p$ **do**
  find $NN_k(x)$
  for $(X_i, Y_i) \in NN_k(x), i = 1, \ldots, k$ **do**
    compute: $\hat{m}(x) = (1/k) \sum_{i \in NN_k(x)} X_i$ and $\hat{g}(x) = (1/k) \sum_{i \in NN_k(x)} Y_i$
    compute: $X_i^c = X_i - \hat{m}(x)$ and $Y_i^c = Y_i - \hat{g}(x)$
  end for
  apply OLS optimization: $\underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i: X_i^c \in B(x, \hat{\tau}_k(x))} (Y_i^c - \beta^T X_i^c)^2 + \lambda \|\beta\|_1$
end for
Return: $\hat{\beta}_{OLS-Lasso} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)$

The steps of the Algorithm 2 resemble the ones in Algorithm 1 and that seems reasonable, since we are extending the existing local OLS model. However, the difference lies on the fact that the regularization parameter needs to be additionally decided. The selection of λ, is

based on a predefined grid and aims to balance the trade-off between model complexity and accuracy can be done by performing Cross Validation to the dataset consisting of the nearest neighbors of point x. After the selection, in order to solve the optimization problem, we aim to find the coefficients β that minimize the particular regularized cost function. This procedure is more computationally expensive since for doing Cross Validation requires training multiple models on different subsets of data. Nevertheless for reasons mentioned above, it can achieve more accurate gradient estimations.

3.3. Local OLS with Diagonal

The objective of the third algorithm is to introduce a new way of estimating the gradient which reduces the computational time and at the same time keeps the efficiency of the original OLS procedure. Considering the analytical solution of the OLS estimator

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y,$$

our attention is being driven to the matrix $\hat{G}(X) = X^T X$ where $\hat{G}(X) \in \mathbb{R}^{p \times p}$, which is called the Gram matrix and it is associated to the OLS problem. Each element of this matrix represents the covariance between $X^{(i)}$ and $X^{(j)}$ features, $(i, j \in 1, \ldots, p)$. However, the inverse of this square matrix, as it is required for the original solution, demands a large number of computations. Thus, it is feasible to take into account only the diagonal elements of the matrix which consist of the variance of $X^{(j)}$.

$$\hat{\beta}_{OLS} = diag((X^T X)^{-1}) X^T Y.$$

By doing this, when taking the inverse matrix the computations are simplified (since all non-diagonal elements are equal to zero) thus, we are guided to the type below:

$$\hat{\beta}^{(j)}_{OLS-diag} = \frac{X^{(j)T} Y}{X^{(j)T} X^{(j)}},$$

for each of the directions $j = 1, \ldots, p$, which is similar to the type that we are using for the Algorithm 3:

$$\hat{\beta}^{(j)}_{OLS-diag} = \frac{\mathrm{cov}(X^{(j)}_{NN_k(x)}, Y_{NN_k(x)})}{\mathrm{cov}(X^{(j)}_{NN_k(x)}, X^{(j)}_{NN_k(x)})}, \quad j = 1, \ldots, p. \tag{4}$$

More specifically, for each dimension j, the algorithm computes $\hat{\beta}^{(j)}_{OLS-diag}$ which is the ratio of the covariance between the j_{th} feature of the nearest neighbors and their output values Y divided by the variance of the j_{th} feature with itself.
This approach offers an efficient method for computing the gradient, while from a theoretical standpoint, it requires less computational time than the OLS and the OLS-Lasso. To be more precise, the main computation part of the gradient estimation requires k computations, one per neighbor, followed by p computations per dimension. Consequently, the final number of computations is $O(kp)$. The importance of the diagonal approximation can be better understood in high-dimensional settings, which we will encounter afterwards, where computing the whole matrix can be expensive or unnecessary.

Algorithm 3 Estimation of $\hat{\beta}_{OLS-diag}$

Input: $(X_1, Y_1), \ldots, (X_n, Y_n) \in \mathbb{R}^p \times \mathbb{R}$, $x \in \mathbb{R}^p$, $n, k \in \mathbb{N}$, $k < n$
for x $\in \mathbb{R}^p$ **do**
  find $NN_k(x)$
  for $(X_i, Y_i) \in NN_k(x), i = 1, \ldots, k$ **do**
    for $j \in (1, \ldots, p)$ **do**
      compute: $\hat{\beta}^{(j)}_{OLS-diag} = \frac{\text{cov}(X^{(j)}_{NN_k(x)}, Y_{NN_k(x)})}{\text{cov}(X^{(j)}_{NN_k(x)}, X^{(j)}_{NN_k(x)})}$
    end for
  end for
end for
Return: $\hat{\beta}_{OLS-diag} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)$

3.4. Local OLS with Trace

Finally, it is the moment to introduce the last algorithm which instead of using the diagonal approximation as the $\hat{\beta}_{OLS-diag}$ does, it involves a trace-based approximation that leads to the estimation of β coefficients. More precisely, the main difference is that rather than taking each element of the main diagonal of the Gram matrix separately, at this time the sum of all these elements is calculated and used for the estimation of each j_{th}-dimension of β.
For each input variable X_i belonging to the kNN of point x, this algorithm calculates the total variance by summing up the variances of each individual feature $X^{(j)}_{NN_k(x)}$

$$d(x) = \sum_{j=1}^{p} \text{cov}(X^{(j)}_{NN_k(x)}, X^{(j)}_{NN_k(x)}).$$

This is the so-called trace approximation since we are taking the sum of all diagonal elements of the matrix $\hat{G}(X)$.
Then, the estimation of each dimension j is computed using the ratio of the covariance between the j_{th} feature of the nearest neighbors and their output values Y divided by the above sum $d(x)$, plus scaling the estimation with the number of total dimensions p.

$$\hat{\beta}^{(j)}_{OLS-trace} = p \cdot \frac{\text{cov}(X^{(j)}_{NN_k(x)}, Y_{NN_k(x)})}{d(x)}, \quad for \quad j = 1, \ldots, p. \tag{5}$$

Trace algorithm (4) is equally efficient to the diagonal version in terms of computational time. It is important to mention that this algorithm introduces dependence in the estimated features since the trace approximation for each coordinate considers the overall variance computation from the p coordinates.

3.5. Computational requirements

After a comprehensive review of the ways in which these algorithms can be implemented, it is now appropriate to offer a concise overview of their computational capabilities. It is widely accepted that the computational time required for the OLS framework and the computation of gradient is particularly significant when dealing with large datasets of size n, particularly when

Algorithm 4 Estimation of $\hat{\beta}_{OLS-trace}$

Input: $(X_1, Y_1), \ldots, (X_n, Y_n) \in \mathbb{R}^p \times \mathbb{R}$, $x \in \mathbb{R}^p$, $n, k \in \mathbb{N}$, $k < n$
for $x \in \mathbb{R}^p$ **do**
  find $NN_k(x)$
  for $(X_i, Y_i) \in NN_k(x), i = 1, \ldots, k$ **do**
    compute: $d(x) = \sum_{j=1}^{p} \text{cov}(X^{(j)}_{NN_k(x)}, X^{(j)}_{NN_k(x)})$
    for $j \in (1, \ldots, p)$ **do**
      compute: $\hat{\beta}^{(j)}_{OLS-trace} = p \cdot \frac{\text{cov}(X^{(j)}_{NN_k(x)}, Y_{NN_k(x)})}{d(x)}$
    end for
  end for
end for
Return: $\hat{\beta}_{OLS-trace} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)$

the work is to be implemented in high-dimensional data, with a high number of dimensions, size p. When considering operations between matrices such as inversion, the number of operations may increase rapidly when either p or n, or indeed both, are of a considerable magnitude. A comparative analysis will be conducted, focusing on the specific aspects that differentiate the algorithms, particularly in the estimation of coefficients for β.
To begin with, for the estimation of the β_{OLS} (1) where we have the OLS formula $\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y$, computation of $X^T X$ involves $O(k \cdot p^2)$ operations since $X_i \in \mathbb{R}^p$ and there are k such points (nearest neighbors). Finally for inverting the above matrix and doing the multiplication with $X^T Y$, $O(p^3)$ operations are required thus, in the end overall fitting cost is $O(k \cdot p^2 + p^3)$.
Secondly, for the estimation of $\beta_{OLS-Lasso}$ (2) since the regularization term $\lambda \|\beta\|_1$ is included, computational complexity is added due to the optimization needed for solving the problem. Moreover, for the selection of λ an a-fold Cross Validation is performed where for each fold, it fits the Lasso model for each λ that belongs to a specified grid. Hence, in the end a complexity $O(a \cdot l)$ is added to the local OLS procedure where a folds of Cross Validation and l the number of λ parameters tested. Hence, the total number is $O(a \cdot l(k \cdot p^2 + p^3))$
Thirdly, for the estimation of $\beta_{OLS-diag}$ (3) as mentioned before, less computations are expected than the first two versions and that is the advantage of this algorithm. More explicitly, for each coordinate $j \in (1, \ldots, p)$ for the covariance between $X^{(j)}_{NN_k(x)}$ and $Y_{NN_k(x)}$ k operations are required and the same for the variance of $X^{(j)}_{NN_k(x)}$. If the p number of dimensions is added we have the total cost of $O(k \cdot p)$.
Finally, for the estimation of $\beta_{OLS-trace}$, (4) we are facing the same behavior with the OLS diagonal algorithm. For each coordinate $j \in (1, \ldots, p)$ we have again k operations for the covariance between $X^{(j)}_{NN_k(x)}$ and $Y_{NN_k(x)}$. Moreover, for computing the trace, $O(k)$ operations for p features are required. In the end the total is the same as the diagonal, meaning $O(k \cdot p)$ computations.
In conclusion, from theoretical perspective the diagonal and trace algorithm seem to be low cost algorithms, whereas when the Lasso regularization parameter is included, then this specific version can become the most computationally expensive.

4. Dimension reduction

The four different ways, estimation of the gradient, presented in the previous section, is designed to facilitate the attainment of the principal objective of the research, which is dimension reduction. The present section will build on the aforementioned procedures with a view to determining whether these methods can yield accurate and computationally inexpensive results for the purpose of obtaining lower-dimensional representations.
Let us begin by introducing the concept of dimension reduction. To illustrate this, we will consider a dataset comprising of a set of points, each of which is a p-dimensional vector, $X \in \mathbb{R}^p = (X^{(1)}, \ldots, X^{(p)})^T$. The objective is to project this dataset into a lower-dimensional subspace of dimension d, where $d < p$.
Considering the regression framework, $Y = g(X) + \varepsilon$, the function $g(X)$ needs to be expressed in another way. This can be achieved by writing $g(X) = g_0(\eta^T X)$ where $\eta^T X \in \mathbb{R}^d$ is the projection of X onto a d-dimensional subspace spanned by projection vectors $\eta_1, \eta_2, \ldots, \eta_d$. Specifically, we have

$$\eta^T X = (\eta_1^T X, \ldots, \eta_d^T X)^T \in \mathbb{R}^d.$$

If for example, $d = 1$ then $\eta = \eta_1^T = (1, 0, \ldots, 0)$ and

$$\eta_1^T X = (1, 0, \ldots, 0) \cdot (X_1, \ldots, X_p)^T = X_1,$$

where it is assumed that the η_i vectors are orthogonal inducing $\eta^T \eta = I_d$.
Since projection vectors are orthogonal, the gradient $g(X) = g_0(\eta^T X)$ can be computed by applying the chain rule and getting:

$$\nabla g(X) = \eta \cdot \nabla g_0(\eta^T X).$$

It is evident that the gradient estimation methodology is effective for approximating these vectors η required for projecting the dataset in a lower dimension, as demonstrated by the fact that the gradient vector, is contained within the subspace created by these vectors, $\nabla g(X) \in span(\eta)$.
In order to estimate these projections there are two strategies analyzed in the following parts.

4.1. Estimating projection vectors η with Eigenvalue Decomposition

The initial method for estimating the projection from the gradients is to compute a matrix, designated by M, which contains the gradient estimations per point, having p gradient coefficients. The points are selected at random and derived directly from the dataset. If we assume that there are m such points, the matrix M can be computed as follows:

$$M = \frac{1}{m} \sum_{i=1}^{m} \hat{\nabla} g(x_i) \cdot \hat{\nabla} g(x_i)^T \approx E[\hat{\nabla} g(X) \cdot \hat{\nabla} g(X)^T], \quad M \in \mathbb{R}^{p \times p}.$$

This matrix is essentially an empirical estimate of the expected outer product between the gradients of g. That's a covariance matrix capturing the variation in the gradients over the data points. In order to estimate the projection vectors η, it is necessary to calculate the eigenvectors of this matrix. Accordingly, an eigen decomposition may be performed, whereby M can be expressed as follows:

$$M = VUV^T,$$

where $V \in \mathbb{R}^{p \times p}$ matrix is an orthogonal matrix containing the eigenvectors of M as its columns. $U \in \mathbb{R}^{p \times p}$ contains the eigenvalues of M.
Then we select the first d eigenvectors from V that correspond to the largest eigenvalues from U. These eigenvectors form the estimated projection matrix:

$$\hat{\eta} = (\hat{\eta}_1, \ldots, \hat{\eta}_d).$$

The objective of the eigenvalue decomposition is to identify the d directions that define the subspace, which explains the variation of the dataset in the lower representation, anticipated to be ideally, $span(M) = span(\eta)$.

4.2. Estimating projection vectors η with Singular Value Decomposition

The alternative way to approximate these projections is by using SVD. Specifically, a matrix M′ is constructed, comprising of the p covariates for each gradient estimation at each of the m points. Thus, matrix M′ is given by the following form:

$$M' = \begin{pmatrix} \nabla g(x_1^{(1)}) & \ldots & \nabla g(x_m^{(1)}) \\ \vdots & \vdots & \vdots \\ \nabla g(x_1^{(p)}) & \ldots & \nabla g(x_m^{(p)}) \end{pmatrix} \in \mathbb{R}^{p \times m}.$$

By applying SVD to M′ we obtain this decomposition: $M' = USV^T$. Matrix $U \in \mathbb{R}^{p \times p}$ is orthogonal and its columns (left singular vectors) represent the directions of maximum variance in the data. Matrix $S \in \mathbb{R}^{p \times m}$ is a diagonal matrix containing the singular values. Matrix $V^T \in \mathbb{R}^{m \times m}$ contains the right singular vectors. For the computation of the projections we select the first d columns of the U-matrix and compute the outer product of each column with itself

$$u_i u_i^T, \quad \forall i \in \{1, \ldots, d\}.$$

In the end, we obtain the final projection matrix by summing the d matrices created such that

$$\hat{H} = \sum_i^d u_i u_i^T, \quad \hat{H} \in \mathbb{R}^{p \times p}.$$

SVD is a valuable tool for the estimation of projections. It identifies the principal directions of variation in the data, thereby capturing the most significant structural features. Furthermore, the left singular vectors are orthogonal, thereby ensuring that the projections obtained are independent.
The SVD procedure can be summed up with the following steps:

Algorithm 5 SVD method for projections η estimation

1: **Input:** $M' \in \mathbb{R}^{p\times m}$, $d, p \in \mathbb{N}$
2: apply SVD to M' such that $M' = USV^T$, with $U \in \mathbb{R}^{p\times p}$, $S \in \mathbb{R}^{p\times m}$, $V^T \in \mathbb{R}^{m\times m}$
3: from U, select $u_1, \ldots, u_d$ vectors from first columns, where $d < p$
4: **for** each $i \in d$: **do**
5: compute: $\hat{\eta}_i = u_i u_i^T$
6: **end for**
7: **Return:** $\hat{H} = \sum_i^d \hat{\eta}_i$

5. Numerical experiments

In this section, we are proceeding to the experimental part, where the algorithms given in the previous parts will be tested to determine their ability to succeed in their tasks. This is going to be a multi-level analysis, covering all possible parameters, such as the number k of nearest neighbors, the number of samples n, the number of dimensions p, as well as the number of reduced dimensions d.

5.1. Presentation of models - Objectives of study

5.1.1. Regression models

The simulation study will be based on two model functions. From these functions, we will extract their gradients $\nabla g(x)$ at a randomly selected points x and then calculate the real projection matrix H. The aim is then to compare the estimations obtained by the proposed algorithms and, with regard to some specific metrics, to observe their distance from the real values (error rate).
The **first model** is:

$$g_1(x) = x_1^2, \quad x \in \mathbb{R}^p. \tag{6}$$

This indicates that a single input feature is used to compute the response variable, although there may be more than one in the dataset, implying that $p \geq 1$. Evidently, this example is utilized for simulation to ascertain the capacity of the algorithms to examine a dataset with p input features and identify the primary one that is most influential on the response variable. Gradient for this function at a point $x \in \mathbb{R}^p$ is the following:

$$\nabla g_1(x) = (2x_1, 0, \ldots, 0)^T.$$

Moving on to dimension reduction, here we have $d = 1$ and as mentioned above, function $g(X) = g_0(\eta^T X)$ where $\eta^T X \in \mathbb{R}^d$ is the projection of X onto a d-dimensional subspace spanned by projection vectors $\eta_1, \eta_2, \ldots, \eta_d$. In that case only the first vector is selected $\eta = \eta_1^T = (1, 0, \ldots, 0)$ and true projection matrix is:

$$H_{True} = \eta_1 \cdot \eta_1^T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

The **second model** is:

$$g_2(x) = x_1 + x_2^2 + x_3, \quad x \in \mathbb{R}^p. \tag{7}$$

This example is slightly more complicated since there is a variable x_2^2 which contributes quadratically to the response variable and two linearly contributions by variables x_1 and x_3. Evidently, we expect from the square variable to have a greater influence than the linear variables separately. Gradient for this function at a point $x \in \mathbb{R}^p$ is the following:

$$\nabla g_2(x) = (1, 2x_2, 1, 0, \ldots, 0)^T.$$

For dimension reduction, the influence will be splitted in two parts the quadratic and the linear, so we have $d = 2$ and: $g(x) = g_0(\eta^T x, \theta^T x) = \eta^T x + (\theta^T x)^2$ where $\eta^T x, \theta^T x \in \mathbb{R}$. Moreover,

$$\eta = (1, 0, 1, 0, \ldots, 0)^T \in \mathbb{R}^p, \quad \theta = (0, 1, 0, \ldots, 0)^T \in \mathbb{R}^p, \tag{8}$$

where η corresponds to the vector for x_1 and x_3 and θ for x_2^2. In the end, the projection matrix is:

$$H_{True} = \eta \cdot \eta^T + \theta \cdot \theta^T = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

5.1.2. Framework of experiments

Starting from the construction of random datasets, it needs to be highlighted that for each experiment 100 random datasets are taken into account, with the intention of being assured that our results are solid and pointing out advantages and drawbacks of the algorithms. As for the input variables, they follow normal distribution and for computing the response variables based on the two model functions (6), (7) a random error is additionally applied $\epsilon \sim N(0, 0.7)$. Four different sample sizes are considered: $n = \{500, 1000, 2000, 5000\}$ and three different total dimension sizes, $p = \{3, 6, 12\}$. So, in general our random datasets are size n and dimension p hence, in total there are 12 different combinations.
The gradient as mentioned above is going to be computed at a point. So, for that reason $m = 100$ points are selected randomly. These points do not belong to the dataset however they have the same characteristics as dataset points (follow Normal distribution). For each of these points their k nearest neighbors are computed. A grid is specified for the selection of k with the aim to identify the best value of k optimizing each algorithm. Note that the choice of k-grid is adapted to different sample sizes. Specifically, the following grids are chosen:

- $n = 500$: $k = [10, 20, 30, 40, 50, 60, 80, 100, 200, 375]$
- $n = 1000$: $k = [10, 30, 50, 80, 100, 200, 300, 400, 600, 750]$
- $n = 2000$: $k = [30, 50, 80, 100, 200, 300, 400, 600, 1000, 1500]$
- $n = 5000$: $k = [30, 50, 80, 100, 200, 350, 500, 1000, 2000, 3750]$

For the OLS Lasso algorithm the grid for obtaining the λ consists of 20 values belonging to the following interval $\lambda \in [10^{-6}, 1]$.
For the evaluation of gradient estimation of the algorithms the squared norm is used computing the difference between the estimation and the real value

$$\|\nabla g(x_i) - \hat{b}(x_i)\|^2, \quad x_i \in \mathbb{R}^p, \quad i = 1, \ldots, m. \tag{9}$$

Given that we are taking m points per dataset, the sum of these m squared errors is calculated and normalized by the division with the number of dimensions p. This process is then repeated for each of the randomly generated datasets. Ultimately, the mean value derived from the aforementioned replications is calculated and applied to the analysis for each value of k in the grid.

For the evaluation of projection matrices H, the Frobenius norm is applied. It is essentially the square root of the sum of squares of all elements in the matrix. Here, the subtraction between the real matrix and the estimated one is computed and then the Frobenius norm is applied. Similarly to the gradient, the sum deriving from m points is computed for each dataset and then by taking the mean quantity of all these sums, the procedure is repeated for each k in the grid. Hence,

$$\|A\|_F = \sqrt{\sum_{i=1}^{p} \sum_{j=1}^{p} |H_{ij} - \hat{H}_{ij}|^2}. \tag{10}$$

5.1.3. Objectives of numerical experiments

The first objective concerns the efficiency of the four algorithms. It is important to show their ability to perform well in different circumstances, such as the two model functions being used here. The experiments are mainly based on the parameters like sample size n, number of dimensions p and the selection of k neighbors for a point x. Thus, we try to find out what is the role of each of them and how they affect the estimation error. We are also analyzing the relationships between them and how they affect each other. Obviously, another goal is to identify the algorithm that achieves the minimum error and requires the fewest number of neighbors to achieve this. Likewise, with a predefined number of neighbors, the algorithms are tested for their ability to predict accurately the response variable.
The same procedures enriched by graphical illustration of the results have been followed for both gradient and projection matrix estimation. Furthermore, scree plots are generated for the purpose of identifying the optimal number of principal components, which are then used to discern the fundamental dimensions of a given dataset.
Finally, in terms of time efficiency, k is frozen at a certain value to determine whether the theoretical computation time results are in agreement with results in practice.

5.1.4. Programming language and packages

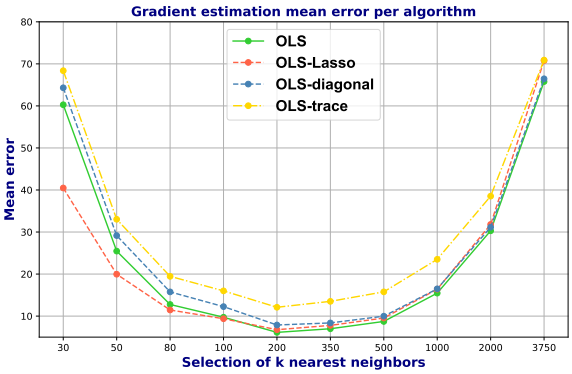
The current research employs Python programming language and the Jupyter Notebook computing platform for numerical experimentation. To facilitate task completion, we utilize Python packages. The *numpy* library is deployed for the purpose of computing covariance matrices, for carrying out the singular value decomposition (SVD) of matrix $\hat{H}$ and calculating squared and Frobenius norm for the evaluation of the algorithms. Furthermore, the exploration of nearest neighbors per point is conducted using the *kneighbors* command from the *scikit-learn* package. Moreover, the aforementioned package facilitates the implementation of the OLS and OLS-Lasso through cross-validation, algorithms (using the *LinearRegression* and *LassoCV* modules). In order to construct the graphs, the Python module *matplotlib* is employed for line plots and scree plots, with *seaborn* package used for heatmaps.

5.2. Experiment results for gradient

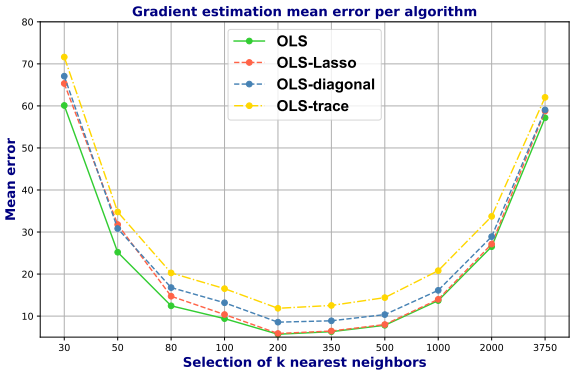
5.2.1. Efficiency

In the first group of graphics (Figure 1) the number of samples n remains constant ($n = 5000$), while the dimensions vary, $p = \{3, 6, 12\}$. The procedures are applied to both models (6), (7). These lineplots present the mean error of gradient estimations per different selection of NN, the way presented above (9). It is clear, the number of neighbors taken into account affects the mean error. If k is small, then the error gets bigger and the same happens correspondingly, when k reaches n, leading to this convex shape across all the algorithms. This shape is not entirely affected by the change of dimension, except the lower values of k that when p increases, they tend to give smaller errors. Finally, since k-grid covers most of the sample size, starting from $k = 30$ until $k = 3750$ and shows that shape, identifies that indeed the minimum error belongs to the selected grid.
Now, comparing the algorithms for their efficiency, it can be seen that OLS and OLS-Lasso tend to be slightly more efficient than OLS-diagonal and OLS-trace. Moreover, it can be highlighted that when Lasso regularization is applied then this algorithm can be efficient with even a lower k-NN selection especially when $p = 12$. The OLS can still have the smallest minimum mean error but with Lasso, this error becomes satisfyingly small with less neighbors needed to be found.

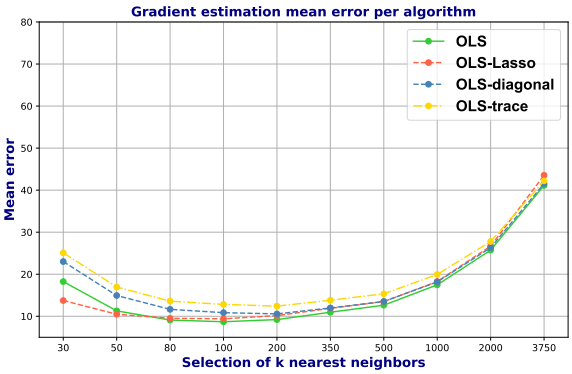
The second group (Figure 2) presents heatmaps of the mean error for gradient estimation but in that case, p remains the same ($p = 12$) in order to make now conclusions about the effect of different sample size ($n = \{500, 1000, 2000, 5000\}$). Hence, it is important to mention that n plays a crucial role for enhancing algorithms efficiency, because when n increases the mean error decreases in all occasions. The number of k required to obtain the global minimum depends on n as well. For example, in OLS-trace when $n = 500$, optimal $k = 100$ and when $n = 5000$, optimal $k = 200$. Finally, as mentioned above OLS-Lasso takes smaller mean error values earlier than the other algorithms in terms of k. This is an advantage because fewer calculations are required to achieve the same result. These plots refer to model (7). The behavior is the same for model (6) and the graphs are presented in appendix A.



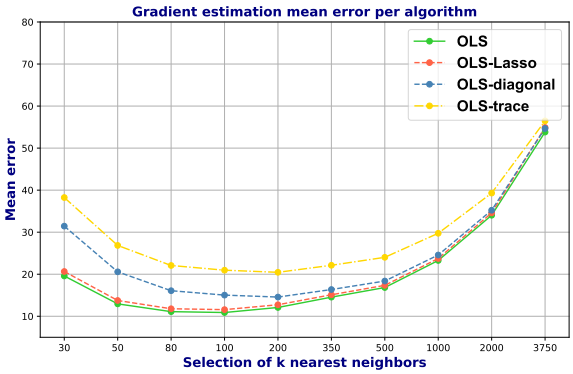
(a) $p = 3$, 1st Model



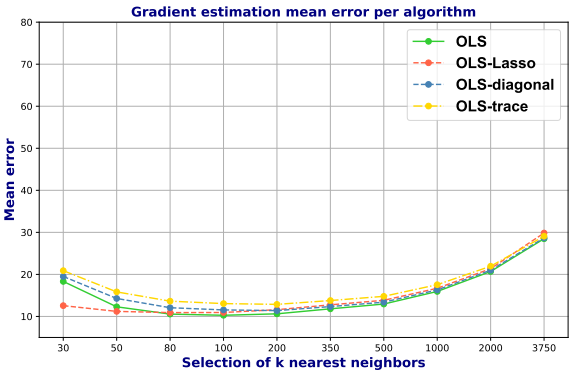
(b) $p = 3$, 2nd Model



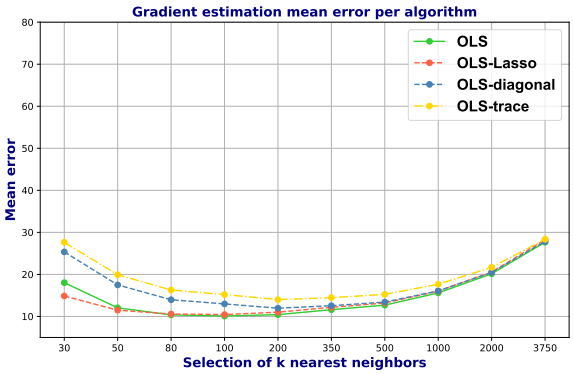
(c) $p = 6$, 1st Model



(d) $p = 6$, 2nd Model

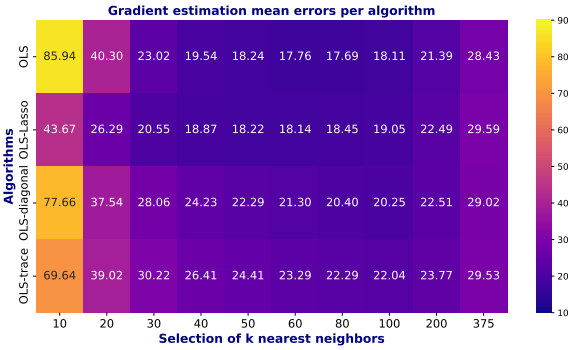


(e) $p = 12$, 1st Model

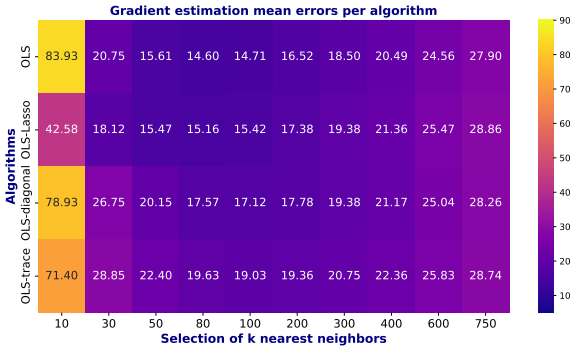


(f) $p = 12$, 2nd Model

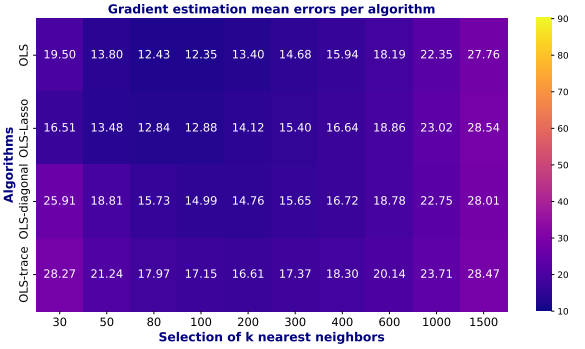
Figure 1: Comparison of algorithms when $n = 5000$ for (6) and (7)



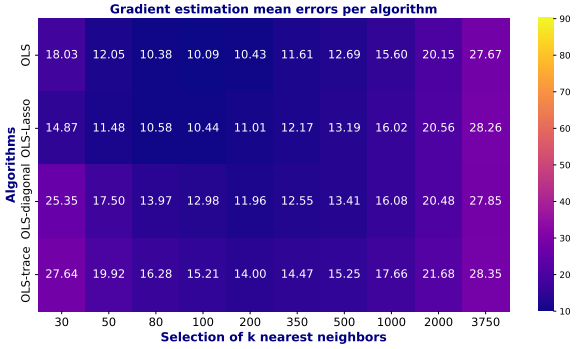
(a) Heatmap for $n = 500$



(b) Heatmap for $n = 1000$



(c) Heatmap for $n = 2000$



(d) Heatmap for $n = 5000$

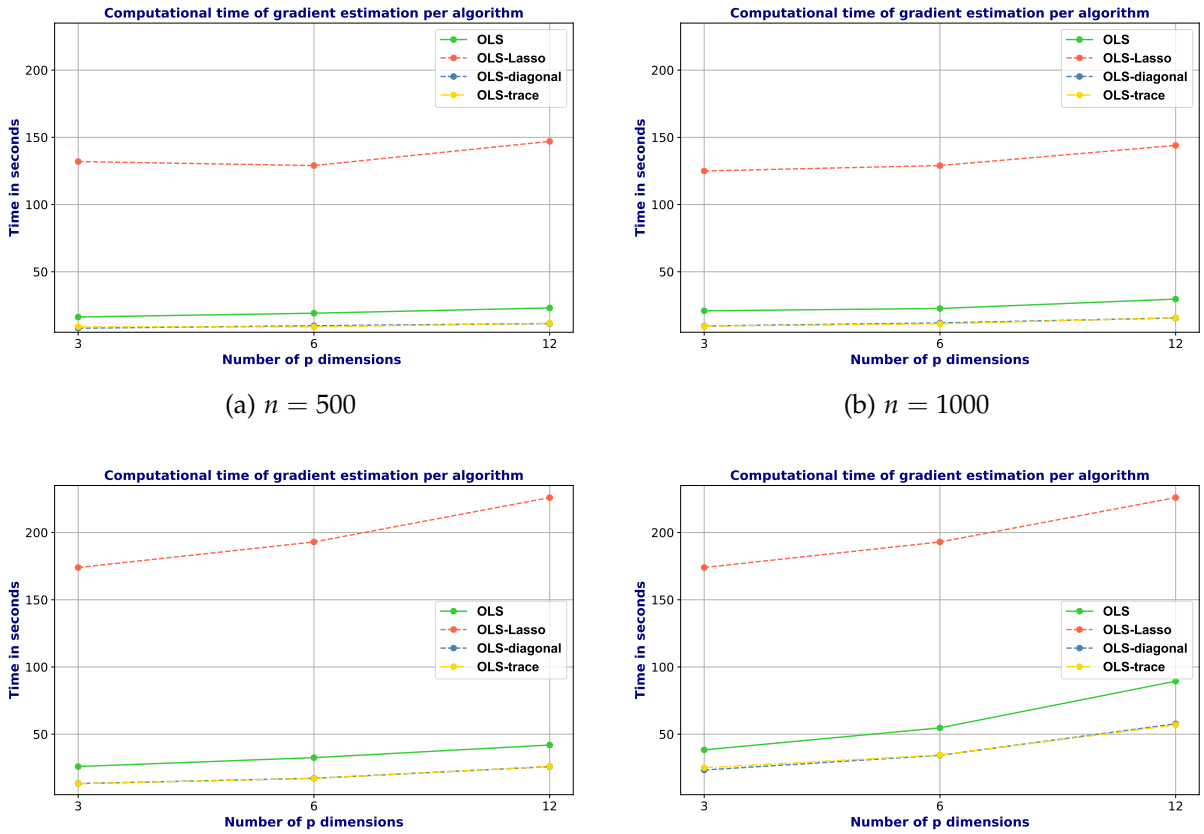
Figure 2: Heatmaps of mean error for $p = 12$ and different number of samples at Model 2 (7)

5.2.2. Computational time analysis

With regard to the comparative evaluation of computational time, the framework is as follows: the gradient error is calculated again in a way like in 5.1.1. However, at this point of the analysis, a specific value of k was selected, which is dependent on the number of samples, n. In particular, the value of k is set to be $k = \sqrt{n}$. This specific value of k has been proposed in some researches as an effective nearest neighbors option see *e.g.* (Lall and Sharma (1996); Ahmad et al. (2024)). Therefore, 10 repetitions of the aforementioned strategy were conducted per each p and the resulting mean time was computed in seconds, to create the lineplots shown in Figure 3. Finally, for computing the duration of each method, the same python packages applied as mentioned in section 5.1.4.

Upon observation of these plots, it is evident that there is a consistency in the behavior exhibited by the algorithms, which is in accordance with the theoretical computations presented in section 3.5. Furthermore, this behavior is independent of the number of samples. Thus, we have first of all OLS_{Lasso}, which is by far the most computational expensive method since as described the extra step of choosing the regularization parameter by using Cross Validation obviously costs more. Second, the two new methodologies OLS_{diag} and OLS_{trace} are cheaper than the OLS due to the avoidance of the inverse matrix calculation. Moreover, these two have the same computational behavior and their associated lines in the lineplots are difficult to distinguish.

A further conclusion that can be drawn is that there is a correlation between the number of samples and the time required to complete the above-described procedure. It is evident that as the value of n increases, so too does the elapsed time for all algorithms. Of course, identically time increases when the samples are on a bigger dimension. In conclusion, we can clearly see the importance of reducing the dimensions of a dataset in order to reduce the computational complexity of the models applied. Once again, for first model, (6), results are on appendix A, having the same manner except the fact that the computational time for the procedure is faster due to the simplified modeling approach.



(a) $n = 500$

(b) $n = 1000$

(c) $n = 2000$

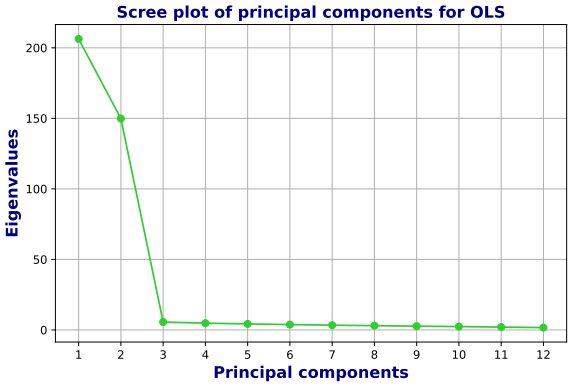
(d) $n = 5000$

Figure 3: Time comparison for different number of samples with $k = \sqrt{n}$ by using (7)

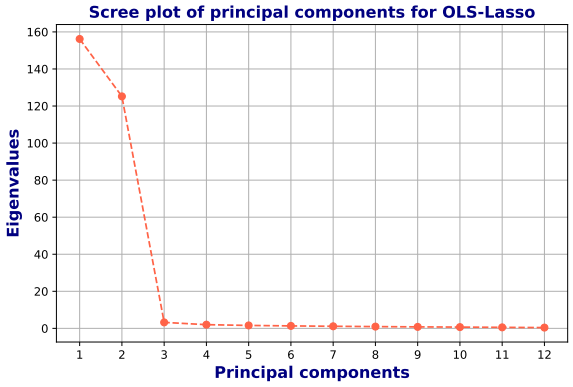
5.3. Experiment results for dimension reduction

5.3.1. Principal components efficiency

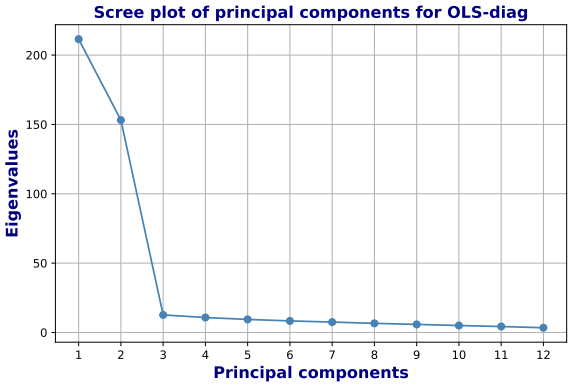
The first method applied aims to distinguish the principal components after doing SVD. For that reason we introduce in Figure 4, scree plots, for each algorithm separately. This visual tool reveals the amount of variance captured by each principal component in a dimensionality reduction technique. It should be noted that the eigenvalues are simply the squared singular values taken by matrix S. For the plots presented here, model (7) is used. Parameters selected are $n = 5000$, $p = 12$ and $k = \sqrt{n}$.



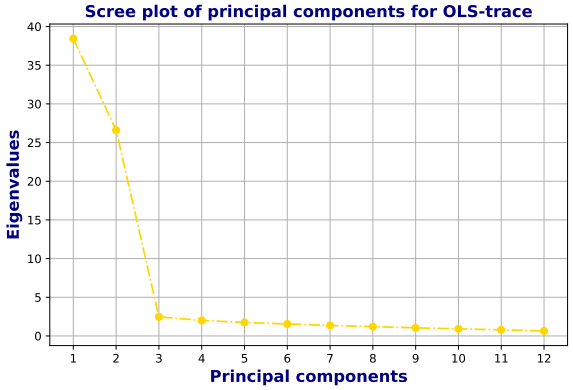
(a) OLS



(b) OLS-Lasso



(c) OLS-diag



(d) OLS-trace

Figure 4: Principal components per algorithm when $n = 5000$, $p = 12$ and $k = \sqrt{n}$ for (7)

Inspection of the plots generated for (7) reveals that all the algorithms are capable of discerning that the initial two components exert a predominant influence on the dataset. Consequently, the objective of dimension reduction can be achieved by considering only these two dimensions, thereby establishing a dimensionality of $d = 2$. Thus, they succeed in distinguishing the number of dimensions that we expected in theory from this model. For the first model scree plots, are equally successive for identifying the first principal component as expected (see appendix B).

Moreover by using left singular matrix and taking the first two columns that belong to the first and second principal component respectively, we can observe the influence that each of the p features has on them. Results presented in Tables 1 and 2 show that the linear features (x_1 and x_3) influence more the first principal component, whereas the squared feature x_2 affects more the 2nd principal component. Each table introduces the explained variability of each feature to principal components.

Table 1: Feature importance: *first principal component*

	OLS	**OLS-Lasso**	**OLS-diag**	**OLS-trace**
x_1	0.688	0.664	0.689	0.675
x_2^2	0.209	0.302	0.191	0.229
x_3	0.687	0.666	0.689	0.683

Table 2: Feature importance: *second principal component*

	OLS	**OLS-Lasso**	**OLS-diag**	**OLS-trace**
x_1	0.142	0.208	0.145	0.172
x_2^2	0.972	0.941	0.970	0.958
x_3	0.153	0.218	0.127	0.155

5.3.2. Prediction ability

The capability of a dimension reduced dataset obtained by the first d principal components, to predict accurately the response variable Y, is assessed. We are taking a dataset where $n = 5000$ and $p = 12$. This dataset is splitted into the training set and the test set. The model that we are working on here is 7 and as highlighted before $d = 2$. Gradients are computed by using $k = \sqrt{n}$ and then SVD is applied to the training set. Then the first two vectors from the left singular matrix are selected. These are multiplied with the full data matrix in order to retrieve the dimension reduced dataset with dimensions $(n_{train}, 2)$ and $(n_{test}, 2)$ for training and test set respectively. This procedure is implemented for each of the proposed algorithms. Approach applied for the comparison is k nearest neighbors for regression by taking advantage of *KNeighborsRegressor* module from *scikit-learn* package. Here, we have $k = 10$. Six datasets in total are compared. The full dataset, $p = 12$, the four dimension reduced datasets derived from the estimated projection vectors with $d = 2$ and finally the dimension reduced dataset earned by the true projections η and θ (8), again with $d = 2$. The evaluation is made by the aid of Mean Squared Error (MSE) between the true response variable and the predicted one. Besides, the coefficient of determination, R^2 that measures the proportion of variance of the response variable explained, ranging from 0 to 1. Results are presented in the Table below.

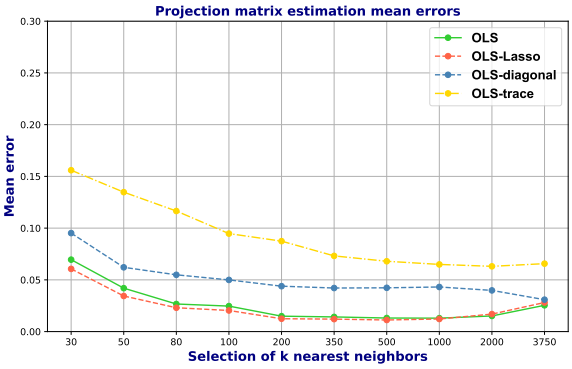
Table 3: Prediction performance comparison of Full and Dimension-Reduced datasets

	Full dataset	**True Reduced dataset**	**OLS**	**OLS-Lasso**	**OLS-diag**	**OLS-trace**
MSE	1.871	0.593	0.594	0.593	0.654	0.634
R^2	0.574	0.866	0.865	0.865	0.851	0.856

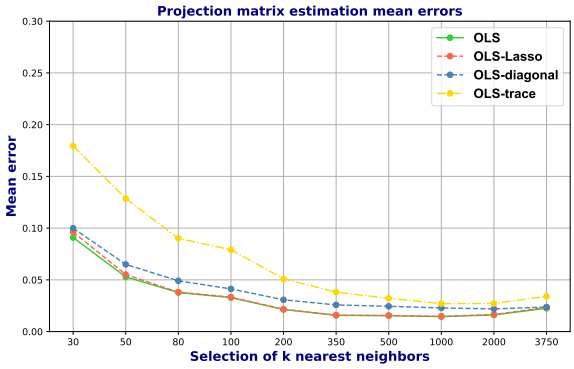
The full dataset shows the poorest performance with $MSE = 1.87$ and 57% of the explained variability. All dimension reduced datasets outperform the full dataset indicating the importance of dimension reduction in the specific model. In particular, all R^2 coefficients are more

than 28% higher than the full dataset and at the same time MSE is lower. The true reduced dataset is obviously outperforming all other datasets and it is used in the experiment as a baseline, with the aim to see the four algorithms results reaching its performance and that seems to happen here.

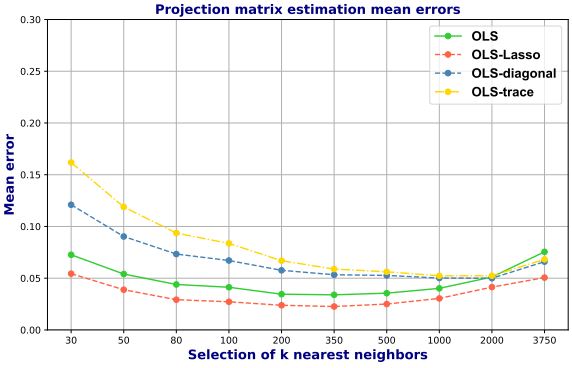
5.3.3. Projection matrix efficiency



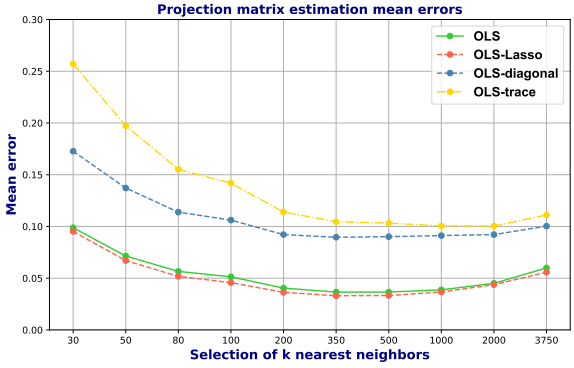
(a) $p = 3$, 1st Model



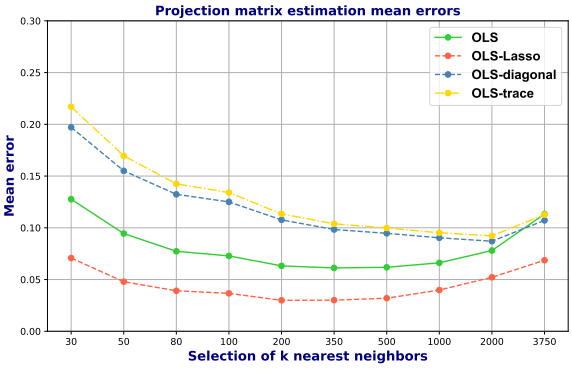
(b) $p = 3$, 2nd Model



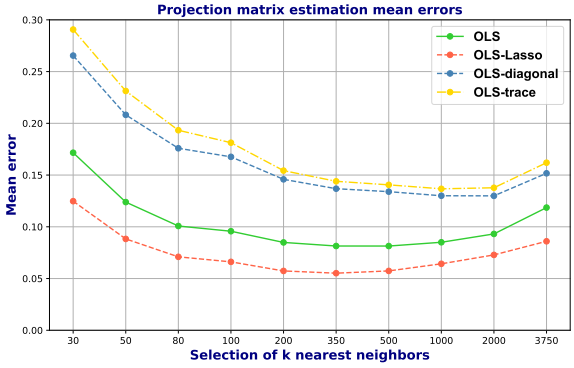
(c) $p = 6$, 1st Model



(d) $p = 6$, 2nd Model



(e) $p = 12$, 1st Model



(f) $p = 12$, 2nd Model

Figure 5: Comparison of projection matrices when $n = 5000$ for models (6) and (7)

Moving on to the evaluation of estimated projection matrices, in Figure 5, these lineplots illustrate the mean error derived from Frobenius norm (5.1.2) for each k in the grid.
To begin our comparison with the two models considered, the 2nd model, as discussed above, is more complicated than the 1st, with two linear and one quadratic variables applied. This ultimately, leads to larger mean errors throughout the k grid, especially as p increases, because only in case where $p = 3$, the behavior of the algorithm lines seems to be almost identical.
Now, we continue with comparing the algorithms. The one being the most effective in all cases is OLS-Lasso showing once more the influence that regularization parameter λ has for optimizing OLS. The difference between these two is not discernible when $p = 3$ but when p increases it is clearly visible. Regarding to OLS-diag and OLS-trace the first one is slightly better during all p however these two are identical when k increases a lot.
In terms of approximating the influence of parameter p, in both cases the increase of the number of dimensions causes a decrease in the efficiency of all algorithms as expected.
Finally, regarding the influence of k nearest neighbors selection, all algorithms tend to need more neighbors in order to obtain their minimum mean error than when only the gradient estimation is assumed. Moreover, at a specific value of k, different for each n, the mean error stabilizes and does not start to worsen as it does in the gradient. Of course, it is obvious that our purpose is to select the lowest number of neighbors possible so as to avoid exceeding computational time. Nevertheless, by connecting these lineplots with the scree plots, we are figuring out that even when $k = \sqrt{n}$, the algorithms are able to identify the correct number of fundamental dimensions d.

6. Conclusion

In this study, we have examined techniques for reducing the dimensionality of a dataset through the estimation of the gradient in a regression context, with the assistance of k nearest neighbors. Based on the local version of Ordinary Least Squares method, we wanted to compare three algorithms with the aim to ascertain their relative efficiency in identifying the optimal lower representation of a dataset, whereby the noise is reduced and only the most pertinent information is retained. Furthermore, it has been of great significance to evaluate the algorithms in terms of their computational performance.

OLS Lasso algorithm, the extension of OLS with a regularization parameter λ, showed better adaptability in estimating the projection matrix, by requiring fewer nearest neighbors. This performance improved as the number of initial dimensions increased. Of course, in real settings, datasets will have even more features than $p = 12$, which is the highest number in our experiments and this may lead to even better performance than OLS. The drawback of this algorithm is the computational complexity. Due to the addition of λ which has been obtained by cross validation, the procedure takes much longer. A possible improvement may be to apply cross-validation to λ only once when estimating the gradient at a random point x, and then obtain the same value of λ for all subsequent points. In this study, cross validation was applied before estimating each different point's gradient.

The novel algorithms presented in the research are the OLS-diagonal and the OLS-trace. The results of their application show their ability to correctly identify the number of fundamental dimensions of a dataset and, moreover, to achieve a satisfactory efficiency in predicting the response variable. Although OLS appears to be slightly more efficient overall, they are able to

perform at a convincing level. However, their major advantage is that they can complete their task much faster than OLS by avoiding inverse computation of the Gram matrix. Therefore, in datasets with large p and n, these two algorithms lead to faster results while maintaining satisfactory efficiency.

In the future, the research can be extended by trying to further improve the two techniques mentioned above, and also by finding solutions to make the Lasso parameter computationally cheaper than it is, at the moment. The increase of the number of features p in the numerical experiments will lead to further conclusions about the strength of the algorithms. Moreover, eigenvalue decomposition can be implemented instead of SVD for obtaining the principal components. Finally, for the prediction evaluation, other methods can be used except from kNN Regressor, like Random Forest. Furthermore, in order to broaden the use of these algorithms in a practical setting, they should be tested on some real data scenarios and applications.

Overall, this study has managed to propose new methods that are able to stand as strong competitors of the OLS, for responding to the task of dimension reduction.

References

Ahmad, N. and A. B. Nassif (2022). Dimensionality reduction: Challenges and solutions. In *ITM Web of Conferences*, Volume 43, pp. 01017. EDP Sciences.

Ahmad, T., F. Portier, and G. Stupfler (2024). Local logistic regression for dimension reduction in classification. *arXiv preprint arXiv:2407.08485*.

Ausset, G., S. Clémençon, and F. Portier (2021). Nearest neighbour based estimates of gradients: Sharp nonasymptotic bounds and applications. In *International Conference on Artificial Intelligence and Statistics*, pp. 532–540. PMLR.

Balakrishnama, S. and A. Ganapathiraju (1998). Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing 18*(1998), 1–8.

Biau, G. and L. Devroye (2015). *Lectures on the nearest neighbor method*. Springer.

Dalalyan, A. S., A. Juditsky, and V. Spokoiny (2008). A new algorithm for estimating the effective dimension-reduction subspace. *The Journal of Machine Learning Research 9*, 1647–1678.

Delecroix, M., M. Hristache, and V. Patilea (2006). On semiparametric m-estimation in single-index regression. *Journal of Statistical Planning and Inference 136*(3), 730–769.

Devroye, L., L. Györfi, and G. Lugosi (2013). *A probabilistic theory of pattern recognition*, Volume 31. Springer Science & Business Media.

Hristache, M., A. Juditsky, J. Polzehl, and V. Spokoiny (2001). Structure adaptive approach for dimension reduction. *Annals of Statistics*, 1537–1566.

Jiang, H. (2019). Non-asymptotic uniform rates of consistency for k-nn regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 33, pp. 3999–4006.

Jolliffe, I. T. (1990). Principal component analysis: a beginner's guide—i. introduction and application. *Weather 45*(10), 375–382.

Lall, U. and A. Sharma (1996). A nearest neighbor bootstrap for resampling hydrologic time series. *Water resources research 32*(3), 679–693.

Lecué, G. and S. Mendelson (2013). Learning subgaussian classes: upper and minimax bounds (2013). *Topics in Learning Theory-Societe Mathematique de France,(S. Boucheron and N. Vayatis Eds.)*.

Mackey, L. (2008). Deflation methods for sparse pca. *Advances in neural information processing systems 21*.

Massart, P. (2007). *Concentration inequalities and model selection: Ecole d'Eté de Probabilités de Saint-Flour XXXIII-2003*. Springer.

McLachlan, G. J. (2005). *Discriminant analysis and statistical pattern recognition*. John Wiley & Sons.

Mukherjee, S., D.-X. Zhou, and J. Shawe-Taylor (2006). Learning coordinate covariances via gradients. *Journal of Machine Learning Research 7*(3).

Partridge, M. and R. Calvo (1997). Fast dimensionality reduction and simple pca. *Intelligent data analysis 2*(3), 292–298.

Schölkopf, B., A. Smola, and K.-R. Müller (1997). Kernel principal component analysis. In *International conference on artificial neural networks*, pp. 583–588. Springer.

Song, Y., J. Liang, J. Lu, and X. Zhao (2017). An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing 251*, 26–34.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology 58*(1), 267–288.

Wang, Y. and L. Zhu (2017). Research and implementation of svd in machine learning. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pp. 471–475. IEEE.

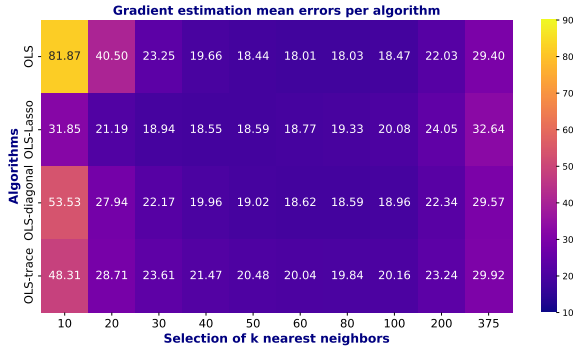
Xia, Y., H. Tong, W. K. Li, and L.-X. Zhu (2002). An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society Series B: Statistical Methodology 64*(3), 363–410.

Ye, G.-B. and X. Xie (2012). Learning sparse gradients for variable selection and dimension reduction. *Machine learning 87*, 303–355.

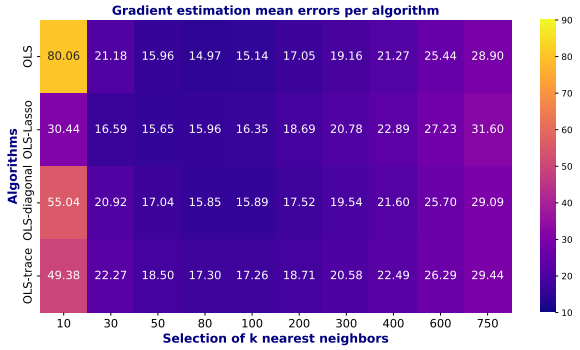
Zhang, S., D. Cheng, Z. Deng, M. Zong, and X. Deng (2018). A novel knn algorithm with data-driven k parameter computation. *Pattern Recognition Letters 109*, 44–54.

Zou, H., T. Hastie, and R. Tibshirani (2006). Sparse principal component analysis. *Journal of computational and graphical statistics 15*(2), 265–286.

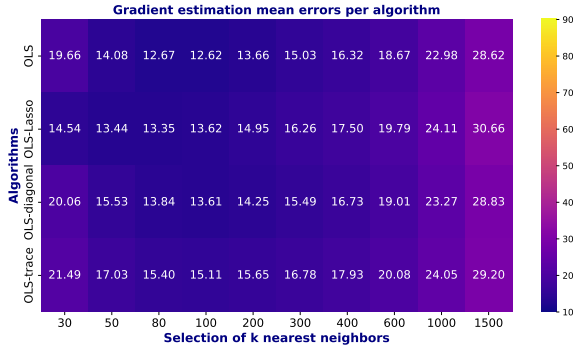
A. Additional graphs for gradient estimation



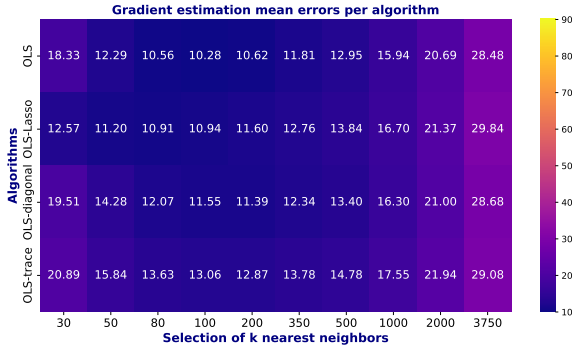
(a) Heatmap for $n = 500$



(b) Heatmap for $n = 1000$

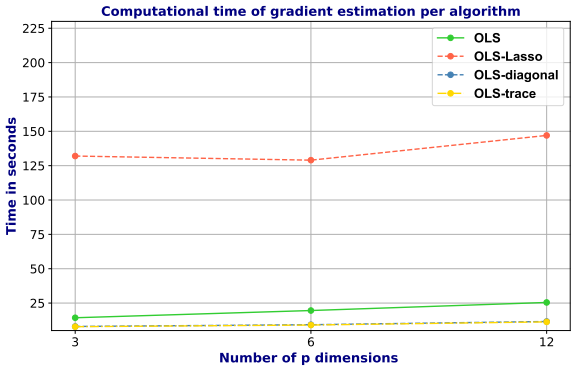


(c) Heatmap for $n = 2000$

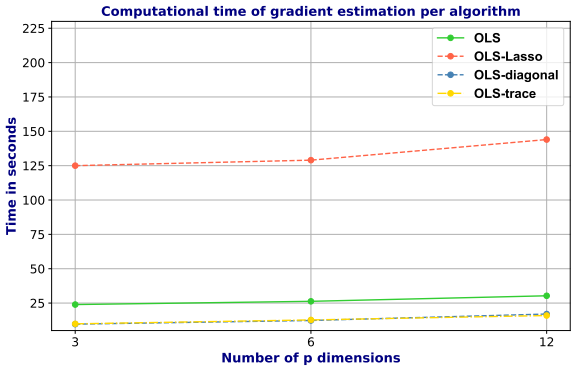


(d) Heatmap for $n = 5000$

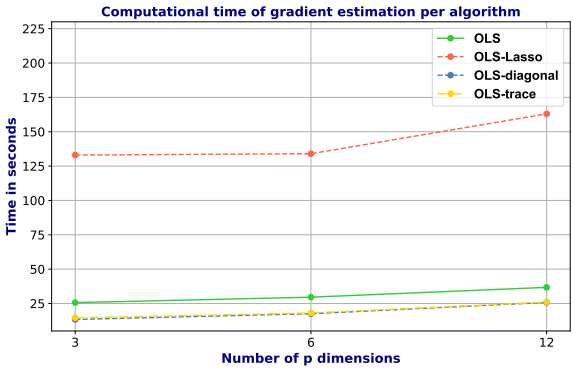
Figure 6: Heatmaps of mean error for $p = 12$ and different number of samples at Model 1 (6)



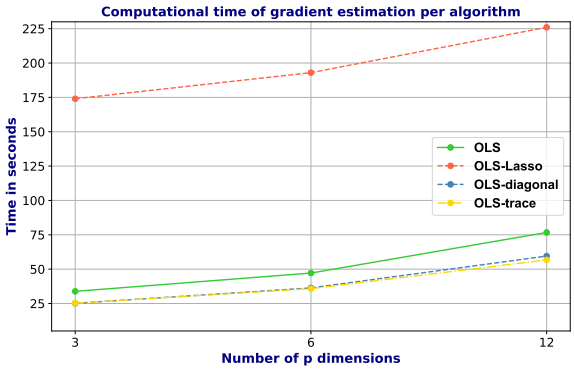
(a) $n = 500$



(b) $n = 1000$



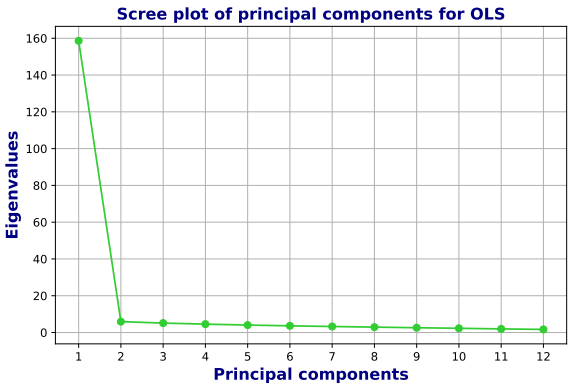
(c) $n = 2000$



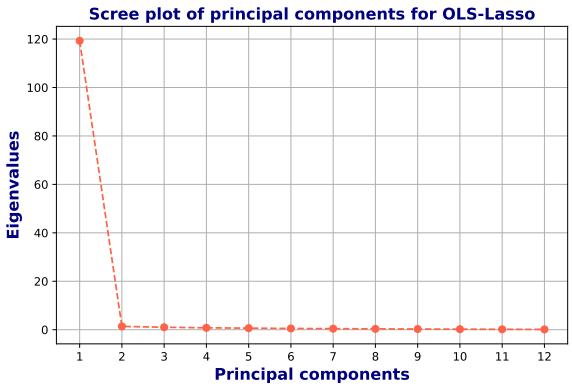
(d) $n = 5000$

Figure 7: Time comparison for different number of samples with $k = \sqrt{n}$, by using Model 1 (6)

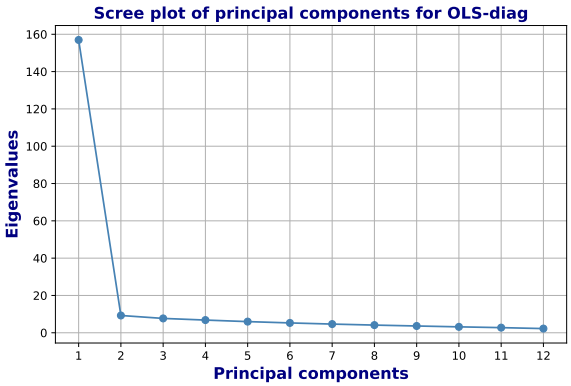
B. Additional graphs for dimension reduction



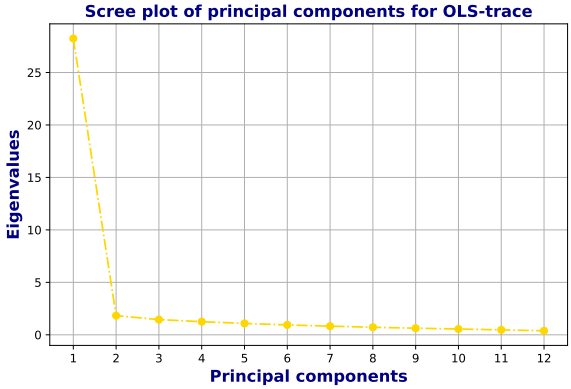
(a) OLS



(b) OLS-Lasso



(c) OLS-diag



(d) OLS-trace

Figure 8: Principal components per algorithm when $n = 5000$, $p = 12$ and $k = \sqrt{n}$ for Model 1 (6)

Table 4: Feature importance: *first principal component*

	OLS	**OLS-Lasso**	**OLS-diag**	**OLS-trace**
x_1^2	0.998	0.999	0.995	0.994