Using sensitivity analysis to understand deep neural network regression

Zhang,Xingjian

2023-08-01

Supervised by Professor Peter Challenor

Acknowledgement

Just an acknowledgement of the ChatGPT language model, I used it to assist me throughout the composition of this research project by structuring sentences, rectifying grammar errors, and refining my English academic writing.

Abstract

Deep learning, such as deep neural networks, has shown excellent performance in a variety of fields such as computer vision and natural language processing. But at the same time, its "black box" nature makes it difficult to understand what the model learns from the data and how it makes decisions. Therefore, there has been a focus on interpretability studies for deep learning. This paper uses global sensitivity analysis based on Sobol' index to understand the importance of each factor in the model.The data used are house prices in Ames, Iowa, USA. Subsequently, the dummy parameter method is integrated into our analysis to pinpoint dominant variables, thereby facilitating model simplification without compromising predictive accuracy. This multi-faceted approach allows not only for the identification of influential variables but also for a substantial model reduction. Furthermore, we extend our analysis to consider a probabilistic framework. Specifically, we estimate the posterior distributions of the Sobol' indices, offering a Bayesian perspective on variable importance.To obtain the posterior, we propose an idea to improve the approximate Bayesian computation, which aims at approximating the posterior samples of the function of model parameters such as Sobol' indices when the model is so complex that it is not possible to obtain both the prior of function and the likelihood function of the parameter for that model. We subsequently demonstrate the effectiveness of the algorithm on a simple polynomial model

Keywords: deep learning, sensitivity analysis, Sobol' indices, approximate Bayesian computation

1 Introduction

1.1 Background of deep learning

Deep learning techniques have been widely used in recent years in many fields such as computer vision, recommender systems, natural language processing, etc., and have demonstrated excellent performance. In this paper we will mainly focus on deep neural network (DNN) models for regression

Although DNN models have outperformed traditional machine learning methods in a large number of domains, their "black box" nature makes them less interpretable than traditional methods such as decision trees or generalized linear models. As the model itself is difficult to understand, this prevents us from understanding the influence of factors on the model on one hand, and on the other hand, the unknown reasons for wrong decisions greatly hinder the practical application of deep learning models. Therefore, the research on the interpretability of deep learning is of far-reaching significance.

1.2 Background of sensitivity analysis on deep learning model

In order to understand the "black box" of deep learning models, a intuitive approach is to judge the importance of the input variables or input samples by examining the impact of changes in the input layer on the results, which is also commonly referred to as sensitivity analysis (SA). Currently there are two main types of SA for neural network models: variable-specific and sample-specific.

There are the following methods of variable SA in neural networks: SA methods based on connection weights [1], SA methods based on partial derivatives [2], methods of observing the effects of input variables by changing them, and SA methods combined with statistical methods [3]. Sample-specific SA focus more on the effect on the model of increasing the weight of a training sample or applying a slight perturbation to the training sample [4].

Our goal is to understand the "black box" of deep learning models by understanding the global impact of variables on output. Therefore, we will not use a local SA such as the previously mentioned partial derivative based SA but will instead use a global sensitivity analysis (GSA), where the output uncertainty is analysed over the entire variation range of the factors [5]. More specifically, we will use the Sobol' method to analyse the deep learning model in a later section, with a view to gaining a deeper understanding of the model.

2 Deep neural network

Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level [6]. The rapid development of deep learning techniques depends on the following factors [7]: (i) the availability of much larger training sets, with millions of labeled examples; (ii) powerful GPU implementations, making the training of very large models practical and (iii) better model regularization strategies, such as dropout.

Deep neural networks, on the other hand, are a specific network structure in deep learning. Simply put, a deep neural network (DNN) model is an artificial neural network (ANN) with multiple hidden layers in between the input and output layers.

In addition, when we adopt the classical structure of feedforward networks to construct a DNN model, a n-hidden-layer DNN model will be possible to express it in the following form:

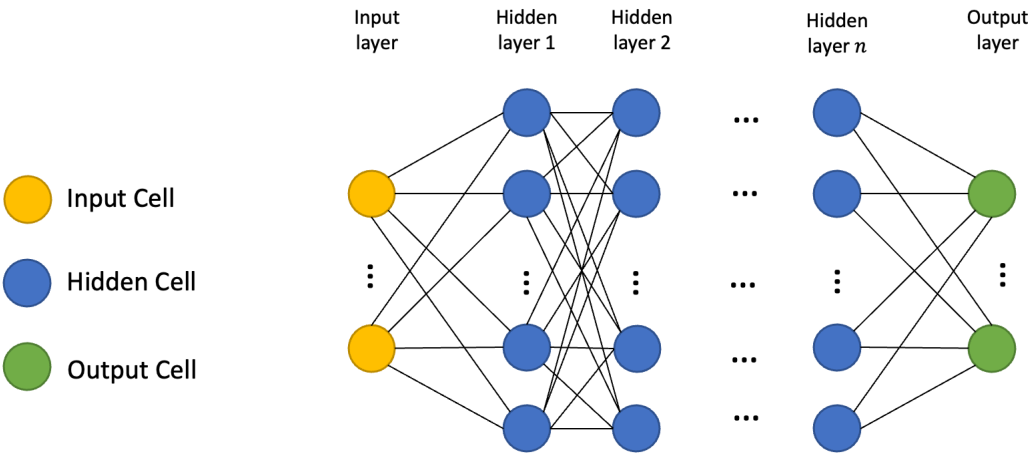

Figure 1: Structure of a n-hidden-layer DNN model

$$
\begin{aligned}
&\boldsymbol{l}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) \\
&\boldsymbol{l}_i = \sigma(\boldsymbol{W}_i\boldsymbol{l}_{i-1} + \boldsymbol{b}_i) \quad i = 2, ..., n \\
&\boldsymbol{y} = \boldsymbol{l}_n
\end{aligned}
$$

While $\boldsymbol{l}_i$ denotes the output of i-th hidden layer, $\boldsymbol{x}$ denotes the input, $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ denote the weight matrix and bias of i-th hidden layer respectively, and $\boldsymbol{y}$ denotes the output of our DNN model.

The DNN model is a very powerful universal function approximator. Hornik et al. rigorously proved that standard multilayer feedforward networks using arbitrary squashing functions can approximate almost any function of interest to any desired degree of accuracy with sufficiently many hidden units [8]. Thus theoretically, given sufficient computational power, we will almost certainly be able to construct suitable DNN models to fit arbitrary functions. Note, this is also true of other regression model such as polynomials.

Although DNN models are very good at fitting functions, their "black-box" nature prevents us from understanding and constructing them well. For example, the ''black box" nature of DNNs is a major weakness compared to traditional statistical approaches that can readily quantify the influence of the independent variables in the modeling process such as polynomial regression [3]. Besides, state-of-the-art DNNs can be sensitive to small perturbations. This vulnerability has called into question their usage in safety-critical applications, including self-driving cars and face recognition [9].

Now there is no suitable theory to guide us on how to design suitable DNN models (e.g., how many hidden layers should be, and how many nodes should be in each layer), so we often need to design the model structure through past experience, which is often very inefficient.

3 Variance-based sensitivity analysis

First we introduce the variance-based SA method. Consider a deterministic numerical model that its univariate output y is governed by an function (can be unknown) $f : \mathbb{R}^n \longmapsto \mathbb{R}$ with n-dimensional input $\boldsymbol{x} = (x_1, ..., x_d)^T$. If we do not treat each input factor as a fixed value, but as a random variable, even if the model mentioned before is determined, the final output will be a random variable since the input uncertainty induces the response uncertainty. Therefore, the model output can be written as follows if the random variables are represented by capital letters:

$$Y = f(\boldsymbol{X}),\ \boldsymbol{X} = (X_1, X_2, ..., X_n)^T$$

From this perspective, the variance-based SA method distributes the variance of the output Y to different sources of uncertainty in the inputs [10]. Among all the variance-based SA methods, one of the most commonly used is the Sobol' method.

3.1 Sobol' indices

The Sobol' method is a classical way of doing SA and has been successfully employed in various application areas [5]. The general idea is to decompose the function into different parts, each part contains only the part about a certain factor. More specific, The Sobol' method relies on the following functional ANOVA (FANOVA) decomposition scheme [11]:

$$f(\boldsymbol{X}) = f_0 + \sum_{i=1}^{n} f_i(X_i) + \sum_{i=1}^{n}\sum_{j>i} f_{i,j}(X_i, X_j) + ... + f_{1,2,...,n}(X)$$

where f_0 is a constant, and the rest of function terms are centred and and orthogonal with each other.

Now let $p(\boldsymbol{X})$ denotes the probability density function, so the variance of Y is:

$$\begin{aligned} Var(Y) =& Var(f(\boldsymbol{X})) \\ =& E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})] - E_{p(\boldsymbol{X})}[f(\boldsymbol{X})]^2 \end{aligned}$$

Since each term in functional ANOVA decomposition of $f(\boldsymbol{X})$ is centred except f_0, which means its expectation is zero, so:

$$E_{p(\boldsymbol{X})}[f(\boldsymbol{X})] = f_0$$

And for $E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})]$:

$$\begin{aligned} E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})] &= \int_{\boldsymbol{X}} p(\boldsymbol{X}) f^2(\boldsymbol{X}) d\boldsymbol{X} \\ &= \int_{\boldsymbol{X}} p(\boldsymbol{X})[f_0 + \sum_{i=1}^{n} f_i(X_i) + \sum_{i=1}^{n}\sum_{j>i} f_{i,j}(X_i, X_j) + ... + f_{1,2,...,n}(X)]^2 d\boldsymbol{X} \end{aligned}$$

Since all terms are orthogonal with each other, after the expansion of the above formula, the expectation of the multiplication of each term is 0. So:

$$\begin{aligned} E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})] = &\int_{\boldsymbol{X}} p(\boldsymbol{X}) f^2(\boldsymbol{X}) d\boldsymbol{X} \\ =&E_{p(\boldsymbol{X})}[f_0^2 + \sum_{i}^{n} f_i^2(X_i) + \sum_{i<j} f_{i,j}^2(X_i, X_j) + ... + f_{1,2,..,n}^2(X_1, ..., X_n)] \end{aligned}$$

Notice that $Var(x) = E(x^2) - E^2(x)$, and since all terms are centred, which means there expectations are 0, so:

$$E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})] = f_0^2 + \sum_{i=1}^{n} \mathbb{V}_i + \sum_{i<j} \mathbb{V}_{i,j} + ... + \mathbb{V}_{1,2,...,n}$$

where $\mathbb{V}_i = Var[f_i(X_i)],\ \mathbb{V}_{i,j} = Var[f_{i,j}(X_i, X_j)]$ and so on.

Finally we can get the variance of $Y = f(\boldsymbol{X})$:

$$\begin{aligned} \mathbb{V}(Y) =&E_{p(\boldsymbol{X})}[f^2(\boldsymbol{X})] - E_{p(\boldsymbol{X})}[f(\boldsymbol{X})]^2 \\ =&f_0^2 + \sum_{i=1}^{n} \mathbb{V}_i + \sum_{i<j} \mathbb{V}_{i,j} + ... + \mathbb{V}_{1,2,...,n} - f_0^2 \\ =&\sum_{i=1}^{n} \mathbb{V}_i + \sum_{i<j} \mathbb{V}_{i,j} + ... + \mathbb{V}_{1,2,...,n} \end{aligned}$$

Through FANOVA decomposition, we can express the variance of the output as above. Next, we need to find a suitable function structure to satisfy the centralization and orthogonalization requirements of FANOVA decomposition. The marginal probabilities for each factor do exactly what we want for orthogonal. Although the expectationz of marginal probabilities may not be zero, we can still centerlize it by item-by-item subtraction. Therefore we get:

$$\begin{aligned} f_i(X_i) =&E_{\boldsymbol{X}_{-i}}[Y|X_i] - E[Y] \\ =&\int_{\boldsymbol{X}} p(\boldsymbol{X}) f(\boldsymbol{X}) dX_1 dX_2 ... dX_{i-1} dX_{i+1} ... dX_n - f_0 \end{aligned}$$

And the first-order Sobol' indices, which shows the first order (or main) effect of X_i on output, are:

$$
\begin{aligned}
S_i =& \frac{\mathbb{V}_i}{\mathbb{V}(Y)} \\
=& \frac{\mathbb{V}[E_{\boldsymbol{X}_{-i}}[Y|X_i] - E[Y]]}{\mathbb{V}(Y)} \\
=& \frac{\mathbb{V}[\int_{\boldsymbol{X}} p(\boldsymbol{X}) f(\boldsymbol{X}) dX_1 dX_2 ... dX_{i-1} dX_{i+1} ... dX_n - f_0]}{\mathbb{V}(Y)} \\
=& \frac{\mathbb{V}[E_{\boldsymbol{X}_{-i}}[Y|X_i]]}{\mathbb{V}(Y)}
\end{aligned}
$$

The first-order Sobol' indices is of great significance, since it can directly reflect the contribution of factor X_i on the variance of output $\mathbb{V}(Y)$. And it can be interpreted as the expected reduction in the total variance $\mathbb{V}(Y)$ when X_i is fixed to a constant [10].

To measure the total effect, including its higher order effects (interactions) with all the other factors, of X_i on $\mathbb{V}(Y)$, we can use total-order Sobol' indices denoted by S_{T_i}. Since it is very difficult to calculate the total order Sobol' indices by adding up the indices of each order when the number of factors is large, we have another alternative:

$$
S_{T_i} = 1 - \frac{\mathbb{V}[E_{\boldsymbol{X}_{-i}}[Y|\boldsymbol{X}_{-i}]]}{\mathbb{V}(Y)}
$$

3.2 Estimation of Sobol' indices

In practice, it is difficult to calculate the Sobol' indices through the above expressions, since we often need to calculate complex integrals when seeking expectations. To overcome issue above, there are several estimator based on sampling developed.

For example, there is a method called pick-freeze scheme to generate samples for further calculation first introduced by Sobol' [11]. This method provides samples by the following expression:

$$
\boldsymbol{A} = \begin{bmatrix} a_{11} & ... & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{N1} & ... & a_{Nn} \end{bmatrix}, \ \boldsymbol{B} = \begin{bmatrix} b_{11} & ... & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{N1} & ... & b_{Nn} \end{bmatrix}, \Rightarrow \boldsymbol{A}_B^{(i)} = \begin{bmatrix} a_{11} & ... & b_{1i} & ... & a_{1n} \\ \vdots & ... & \vdots & ... & \vdots \\ a_{N1} & ... & b_{Ni} & ... & a_{Nn} \end{bmatrix}
$$

where N is the number of samples, $\boldsymbol{A}$ and $\boldsymbol{B}$ are two random $N \times n$ matrices and $\boldsymbol{A}_B^{(i)}$ is the matrix that same as $\boldsymbol{A}$ except the i-column has been replaced by the i-column of $\boldsymbol{B}$.

After having all matrices above, we can estimate first and total-order Sobol' indices by following estimators proposed by Jansen [12]:

$$
\begin{aligned}
\hat{S}_i &= 1 - \frac{\frac{1}{2N}\sum_{i=1}^{n}[f(\boldsymbol{B}) - f(\boldsymbol{A}_B^{(i)})]^2}{\mathbb{V}(Y)} \\
\hat{S}_{T_i} &= \frac{\frac{1}{2N}\sum_{i=1}^{n}[f(\boldsymbol{A}) - f(\boldsymbol{A}_B^{(i)})]^2}{\mathbb{V}(Y)}
\end{aligned}
$$

To sample the three matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{A}_B^{(i)}$, the current main methods are the Monte Carlo (MC) method, the quasi-Monte Carlo (QMC) method and the Latin hypercube sampling (LHS) method. In the following sections we will use the LHS method for sampling due to its higher efficiency than the traditional MC method.

4 House price regression

Before using the SA method to try to understand the DNN model, we should first model a real problem, so that we can use prior knowledge such as real experience to analyze whether the results can really help us understand, or whether consistent with our perception.

Therefore, we choose to regress the housing prices in Ames, Iowa, USA. This is a classic data analysis data set, which contains a large number of factors that may affect house prices, covering almost all aspects. Generally speaking, people have a certain understanding of the main factors that affect housing prices, such as housing quality and location. But in general, housing prices are affected by a large number of factors and it is difficult for us to obtain a comprehensive understanding of their mechanisms. Therefore, a DNN model that is good at representation learning and extracting features from raw data [6] is suitable for this situation.

In the previous work, DNN models performed well on such problems [13]. Therefore, when the model fits well, it is meaningful to use methods like SA to further understand this "black box" since we can understand the complex housing price determination mechanism in reality by understanding the good-fitting model.

4.1 Dataset introduction and preprocessing

4.1.1 Dataset

We will use a data set of house prices from Ames, Iowa, USA for regression. This dataset is provided by Kaggle, compiled by Dean De Cock for use in data science education. It contains 1460 samples with 79 input variables describing (almost) every aspects. There are 37 numeric variables and 43 category variables, and a detailed description of these variables is given in the appendix.

Since the DNN model cannot handle categorical variables, we will use onehot encoding to convert it into a 0-1 vector form; in addition, for numerical variables, since the scale of each variable are different, we will perform all numerical variables and output standardization.

4.1.2 Metrics

Mean Squared Error

Mean Squared Error (MSE) is used as loss to evaluate the performance when we train our DNN model in this research. MSE is chosen because when we assume that the noise in the model is Gaussian, minimizing the mean squared error is equivalent to finding the maximum likelihood estimate of the model. The definition of MSE shows below:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

where n is the number of samples, y_i is the true value of i-th sample and $\hat{y}_i$ is the predicted value of i-th sample.

Mean Average Error

Mean Average Error (MAE) is used to evaluate the performance of our regression model in this research. MAE is chosen since it's a traditionally good indicator for regression. The definition of MAE shows below:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

where n is the number of samples, y_i is the true value of i-th sample and $\hat{y}_i$ is the predicted value of i-th sample.

4.1.3 Data standardization

Categorical variables

For categorical variables, we will use onehot encoding to transfer categorical variable into 0-1 vector. In onehot encoding, if there are n categories, a 0-1 vector of length n will be created. Compare to dummy variable method, onehot encoding is easier to understand. And since we are using DNN model, which is not susceptible to multicollinearity as linear regression, onehot encoding is a better method here compare to dummy variable.

After onehot encoding, we have 328 input features in total.

Numerical variables

For numerical variables, since all variables represent different values, some about money and some about square feet, in order to have a more generalized and accurate DNN regression model here, we need to make the input features of the model have the same scale or no scale, so as to avoid a certain characteristic becoming the dominant factor of the model just because it has an excessively large value. Therefore, the normalization below shows the process:

$$x_{i,j}^{norm} = \frac{x_{i,j} - \mu_j}{\sigma_j}$$

where $x_{i,j}^{norm}$ denotes the normalized j-th feature of i-th sample, $x_{i,j}$ denotes the original j-th feature of i-th sample, μ_j denotes the mean value of j-th feature and σ_j denotes the standard deviation of j-th feature.

4.2 Regression using deep learning

After the preprocessing of our dataset, we can build up our DNN model for regression.

4.2.1 Network structure

Our model is a fully connected DNN containing 1 input layer with 328 input features, 1 output layer with one single output, 4 hidden layers with 200, 100, 50, 25 nodes each. Firstly since there

are 80 input variables(a total of 326 input features after transforming the categorical variables into 0-1 vectors using onehot encoding) in total, we want the first layer of 200 nodes to capture as many features of the data as possible. By reducing the number of nodes in deeper layers, we hope the model is forced to represent the most important abstracted information with fewer neurons, effectively.

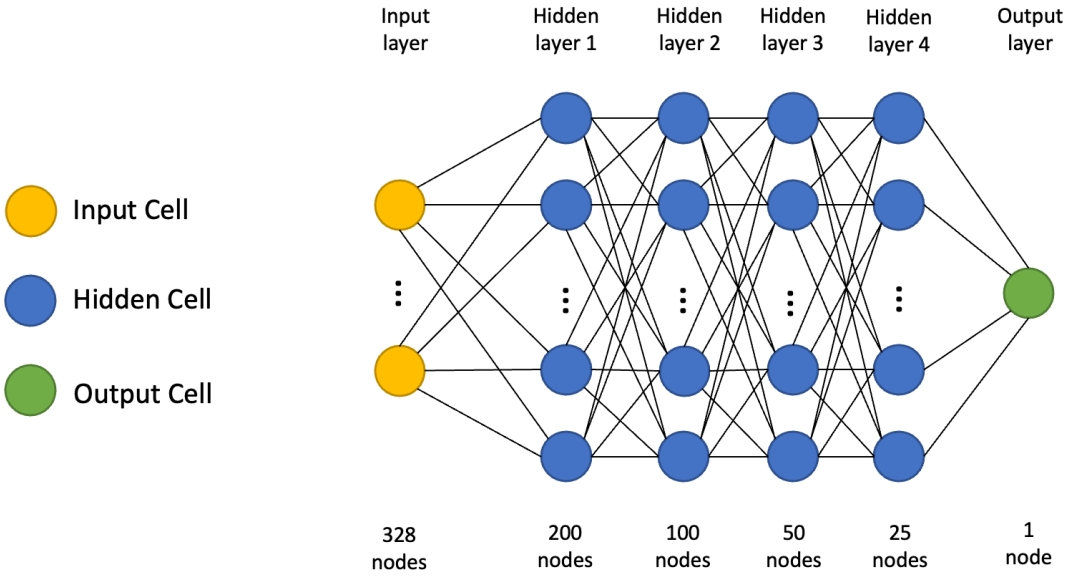

Figure 2: Structure of our DNN model

The sample fed in our DNN model is a $B \times d$ matrix, where B denotes the batch size, i.e. the number of samples, and $d = 328$ denotes the number of features after proprocessing. The structure shows above.

And for each note in hidden layer, we will choose rectified linear function (ReLU) as its activation function. Since the calculation of each layer is only a linear combination of the shape $\boldsymbol{W}_i\boldsymbol{x}+\boldsymbol{b}_i$ without an activation function, we need an activation function such as ReLU to introduce nonlinearity into the model, so that the model can fit the structure of the nonlinear function.

4.2.2 Result

In the training process, 1000 epochs and batch size 32 have been set for our DNN model, which shows in Figure 3. Model converged around 800 epochs, resulting in train loss = 0.4024 and validation loss = 0.3768, then validation error didn't decrease and increase again.

Then evaluate model performance on test set, the MSE = 0.2013 and MAE = 0.2621. The comparison results between the prediction set and the training set are shown in the figure 4:

Since the value of the test set should obey $N(0, 1)$ after normalization, we can find that most of the predicted values are basically consistent with the actual values, and only a small number of

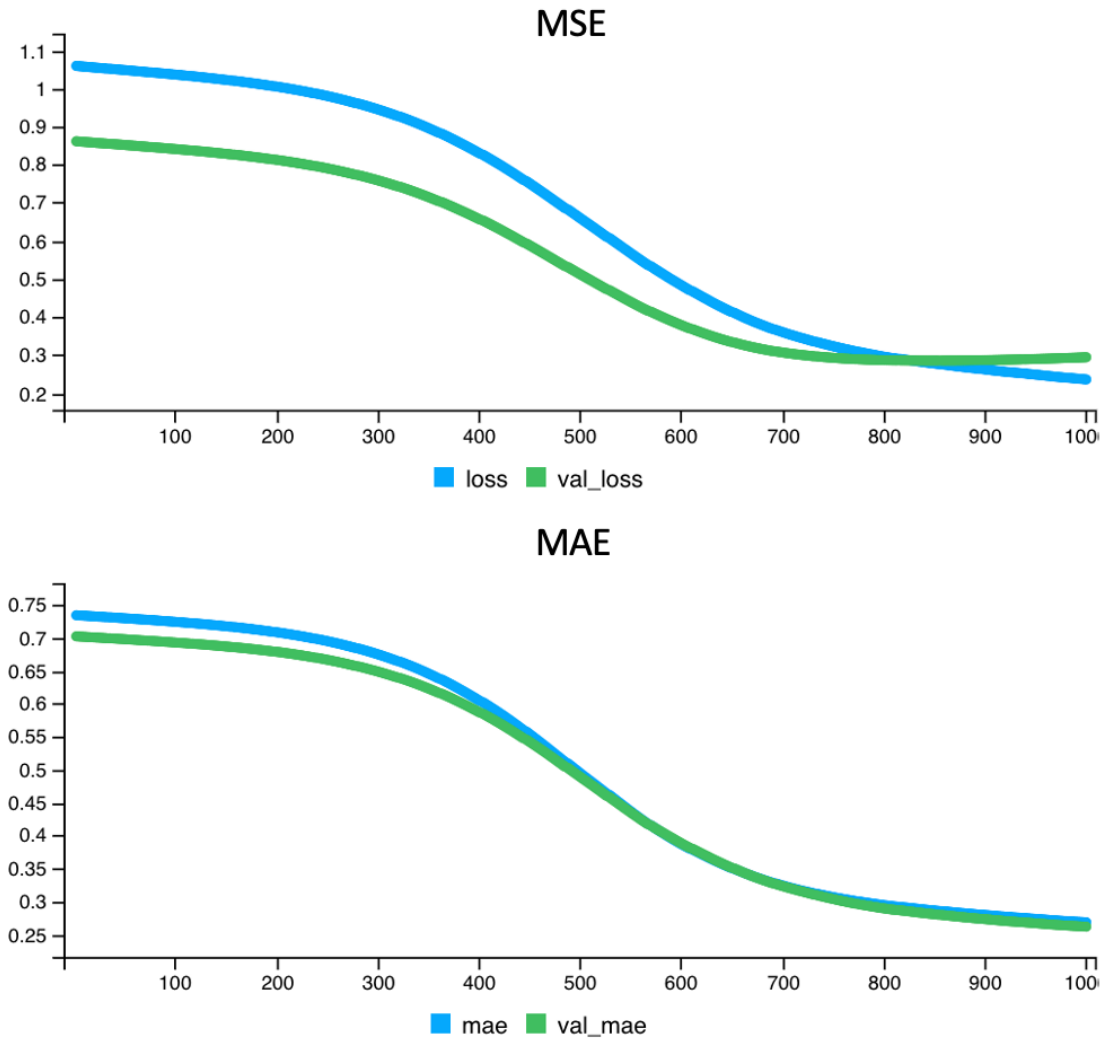

Figure 3: The MSE and MAE during training and validation processes, where the blue line represents the metrics of the test set and the green line represents the validation set

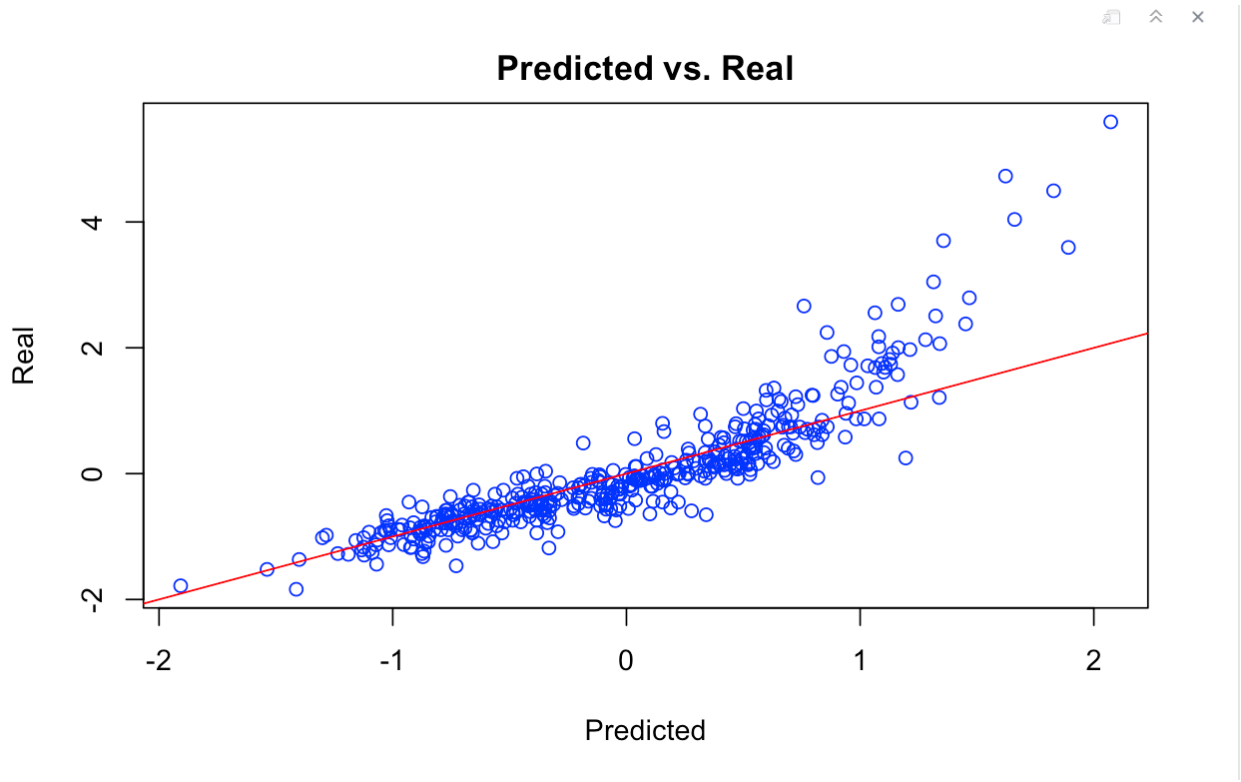

Figure 4: Comparison between predicted and real values

extreme cases (value > 1, indicate that the probability of occurrence is small) samples predict poorly. Therefore we can assume that our DNN model fit well on this dataset for most cases.

4.3 Sobol' sensitivity analysis

After getting a well-fitting DNN model, the packages **sensobol** and **sensitivity** are employed to conduct SA on it. We will choose Latin Hypercube sampling method to get sample points, and the number of sample points and bootstrap replicates to get confidence intervals for the estimates is $N = 3500$ and $R = 500$, respectively.

Then we get tables of first-order and total-order Sobol indices for the model factors, showing in Figure 5 and Figure 6:

In general, the largest first-order Sobol' index, i.e. first-order Sobol' index of OverallQual is only 0.069, which means the main effect of this factor contributes only 6.9% of the output variance in our model. And since the difference between the total order and the first-order Sobol' index can be regarded as whether the factor interacts significantly with other factors, while the maximum of those difference is only 0.007, we can assume that all factors have little interaction with other factors.

Therefore, by performing SA, we can gain some understanding of this "black box" model: this model with 326 output features has no obvious dominant factor according to Sobol' indices and all of them have little interaction with others.

```
##      original   bias std.error low.ci high.ci sensitivity            parameters
##   1:    0.002  0.000     0.009 -0.015   0.020          Si             MSSubClass
##   2:    0.029 -0.001     0.023 -0.015   0.075          Si                LotArea
##   3:    0.069  0.001     0.041 -0.013   0.149          Si            OverallQual
##   4:    0.011  0.001     0.016 -0.020   0.042          Si            OverallCond
##   5:    0.008  0.001     0.017 -0.026   0.040          Si              YearBuilt
##  ---
## 324:    0.001  0.000     0.003 -0.005   0.008          Si  SaleCondition=Alloca
## 325:    0.002  0.000     0.006 -0.010   0.014          Si  SaleCondition=Family
## 326:    0.000  0.000     0.001 -0.002   0.003          Si  SaleCondition=Normal
## 327:    0.000  0.000     0.002 -0.003   0.003          Si SaleCondition=Partial
## 328:    0.000  0.000     0.002 -0.004   0.005          Si      SaleCondition=NA
```

Figure 5: First-order Sobol' indices of all factors

```
##      original bias std.error low.ci high.ci sensitivity            parameters
##   1:    0.004    0     0.000  0.003   0.004          Ti             MSSubClass
##   2:    0.024    0     0.001  0.023   0.026          Ti                LotArea
##   3:    0.075    0     0.002  0.071   0.079          Ti            OverallQual
##   4:    0.010    0     0.000  0.010   0.011          Ti            OverallCond
##   5:    0.015    0     0.000  0.014   0.015          Ti              YearBuilt
##  ---
## 324:    0.000    0     0.000  0.000   0.000          Ti  SaleCondition=Alloca
## 325:    0.002    0     0.000  0.002   0.002          Ti  SaleCondition=Family
## 326:    0.000    0     0.000  0.000   0.000          Ti  SaleCondition=Normal
## 327:    0.000    0     0.000  0.000   0.000          Ti SaleCondition=Partial
## 328:    0.000    0     0.000  0.000   0.000          Ti      SaleCondition=NA
```

Figure 6: Total-order Sobol' indices of all factors

4.4 Refit model using dominant factors

Although in the previous section we fitted the house price data and obtained some understanding of the model through GSA, there are still improvements in the model. For example, the low values of the Sobol' indices for all the factors means that there are no factors that can significantly influence house prices. If it were not for the true mechanism itself that determines house prices this could indicate that the DNN model is not learning the true mechanism. However, since our goal is more focused on the process of how to use GSA to understand DNN models, we can hold off on optimising DNN models here.

In addition, another optimisable point is the selection of dominant factors to refit a simplified model. A DNN model with 326 input features is very unfriendly in terms of computation and comprehension. Since the Sobol' indices illustrates the percentage contribution of the input factor to the variance of the output, it can also be interpreted as the importance of the factor to the output. Therefore, we can select dominant factors by values of their Sobol' indices. However, what criteria should we use to select the dominant factor, or what is the value of Sobol' indices greater than which we can consider it as the dominant factor? Next we will describe the criteria for selection

4.4.1 Using a dummy parameter to identify non-influential factors

Theoretically, according to the formula derived in the previous section, the value of Sobol' index for a factor should be a constant value greater than 0. However, since it is often not possible to calculate this value directly, especially for complex models, we use a numerical approximation based on the pick-freeze scheme as mentioned above. Therefore, even for completely output-independent factors, which should theoretically have a Sobol' indices value of 0, it is possible to have a non-zero Sobol' value in our calculations.

In order to assess the threshold that identifies non-influential parameters in GSA methods, we propose to calculate the sensitivity index of a "dummy parameter", which has no influence on the model output [14]. For the same reasons as above, this dummy parameter should have a non-zero Sobol' index as well. Thus we can consider those factors with Sobol' indices values lower than the dummy parameter to be non-dominant and those with Sobol' values higher than the dummy parameter to be dominant factors.

4.4.2 Selection of domiant factors

Firstly, processing Sobol' SA on a dummy parameter and we get the Sobol' incides:

where the total order Sobol' index is 0 and the first order Sobol' index is 0.0036, which illustrates that there is a real possibility that we will identify non-influential factors as influential in the calculation process.

Then, we filtered out all the factors with first order Sobol' values greater than 0.0036 and displayed them as follows:

Therefore, we selected 76 features among all 326 features, which shows in Figure 8. After removing duplicate factors belonging to the same categorical variable, it means 55 variables of all 80 variables are considered influential to house price.

```
##      original          bias    std.error      low.ci     high.ci sensitivity
## 1 0.003621324  3.068020e-06 8.999236e-05 0.003441874 0.003794637          Si
## 2 0.000000000 -2.464447e-03 9.108050e-02 0.000000000 0.149825453          Ti
##   parameters
## 1      dummy
## 2      dummy
```

Figure 7: Sobol' indices of dummy parameter

```
##  [1] "LotArea"              "OverallQual"          "OverallCond"
##  [4] "YearBuilt"            "YearRemodAdd"         "BsmtFinSF1"
##  [7] "BsmtFinSF2"           "BsmtUnfSF"            "TotalBsmtSF"
## [10] "X1stFlrSF"            "X2ndFlrSF"            "GrLivArea"
## [13] "BsmtFullBath"         "FullBath"             "HalfBath"
## [16] "BedroomAbvGr"         "KitchenAbvGr"         "TotRmsAbvGrd"
## [19] "Fireplaces"           "GarageCars"           "GarageArea"
## [22] "WoodDeckSF"           "OpenPorchSF"          "ScreenPorch"
## [25] "MSZoning=RM"          "LotShape=IR3"         "LotShape=Reg"
## [28] "LotShape=NA"          "LandSlope=Gtl"        "LandSlope=Sev"
## [31] "Neighborhood=NoRidge" "Condition1=NA"        "Condition2=RRAe"
## [34] "BldgType=Duplex"      "HouseStyle=SLvl"      "RoofStyle=NA"
## [37] "RoofMatl=Membran"     "RoofMatl=WdShngl"     "Exterior1st=HdBoard"
## [40] "Exterior2nd=CBlock"   "Exterior2nd=Stucco"   "MasVnrType=BrkFace"
## [43] "MasVnrType=NA"        "ExterQual=Fa"         "ExterQual=Gd"
## [46] "ExterCond=TA"         "Foundation=PConc"     "BsmtQual=Ex"
## [49] "BsmtQual=Gd"          "BsmtCond=TA"          "BsmtExposure=Av"
## [52] "BsmtExposure=No"      "BsmtFinType1=NA"      "Heating=Grav"
## [55] "HeatingQC=Ex"         "HeatingQC=Gd"         "HeatingQC=Po"
## [58] "HeatingQC=TA"         "HeatingQC=NA"         "Electrical=SBrkr"
## [61] "KitchenQual=Gd"       "KitchenQual=TA"       "KitchenQual=NA"
## [64] "Functional=Mod"       "Functional=NA"        "FireplaceQu=Gd"
## [67] "FireplaceQu=NA"       "GarageFinish=Fin"     "GarageFinish=RFn"
## [70] "GarageFinish=NA"      "PoolQC=Ex"            "PoolQC=NA"
## [73] "Fence=GdPrv"          "Fence=NA"             "MiscFeature=Othr"
## [76] "MiscFeature=Shed"
```

Figure 8: Dominat factors selected by Sobol' indices

4.4.3 Refitting model

From the above analysis, we refit the DNN model using the 76 features selected, and we will train a model which has same structure as before.

According to our calculation, the sum of total order indices of selected factors is 0.763, which suggests that the old model believes that 76.3% of the variance of output should come from the selected factors. So when we fit using only selected features, do we need to add white noise to account for the variance originally provided by the other features?

From the above analysis, since the Sobol' values of the discarded factors are even lower than the completely irrelevant dummy parameter, we can assume that these discarded factors actually do not contribute at all to the variance of the output, and are incorrectly identified and fitted by our DNN. Therefore in the new model we do not need to include white noise to account for this.

From the above analysis, we refit the DNN model using the 76 features selected. Since the number of input features has been significantly reduced, we can assume that the complexity of the model should be reduced as well. Therefore, we design new DNN model as a fully connected DNN containing 1 input layer with 76 input features, 1 output layer with one single output, 3 hidden layers with 200, 100, 50 each.

In the training process, 500 epochs and batch size 32 have been set for our new DNN model, which shows in Figure 9. Model converged around 460 epochs, resulting in train loss = 0.2656 and validation loss = 0.3034, then validation error didn't decrease and increase again.

Then evaluate model performance on test set, the MSE = 0.2134 and MAE = 0.2697. The comparison results between the prediction set and the training set are shown in the figure 10:

Now we can compare the new DNN model with old model:

Model	Total number of parameters	Number of trained epochs at convergence	MSE on test set	MAE on test set
Old model	92251	800	0.2013	0.2621
New model	40601	460	0.2134	0.2697

By comparison, the new model constructed by discarding unaffected factors using GSA has significantly lower model complexity and training time than the old model when the metrics (MSE and MAE) are at roughly the same levels, which demonstrates the success of using the GSA results to understand and simplify the model.

4.4.4 The final optimised model

However, after analysing the new model again using the dummy parameter method, it was found that a large number of factors still belonged to the non-influential factors. Therefore, we repeated the cycle of "build model - screen factors using dummy parameter method - refit on new selected factors" until the model finally did not contain any non-influential factors.

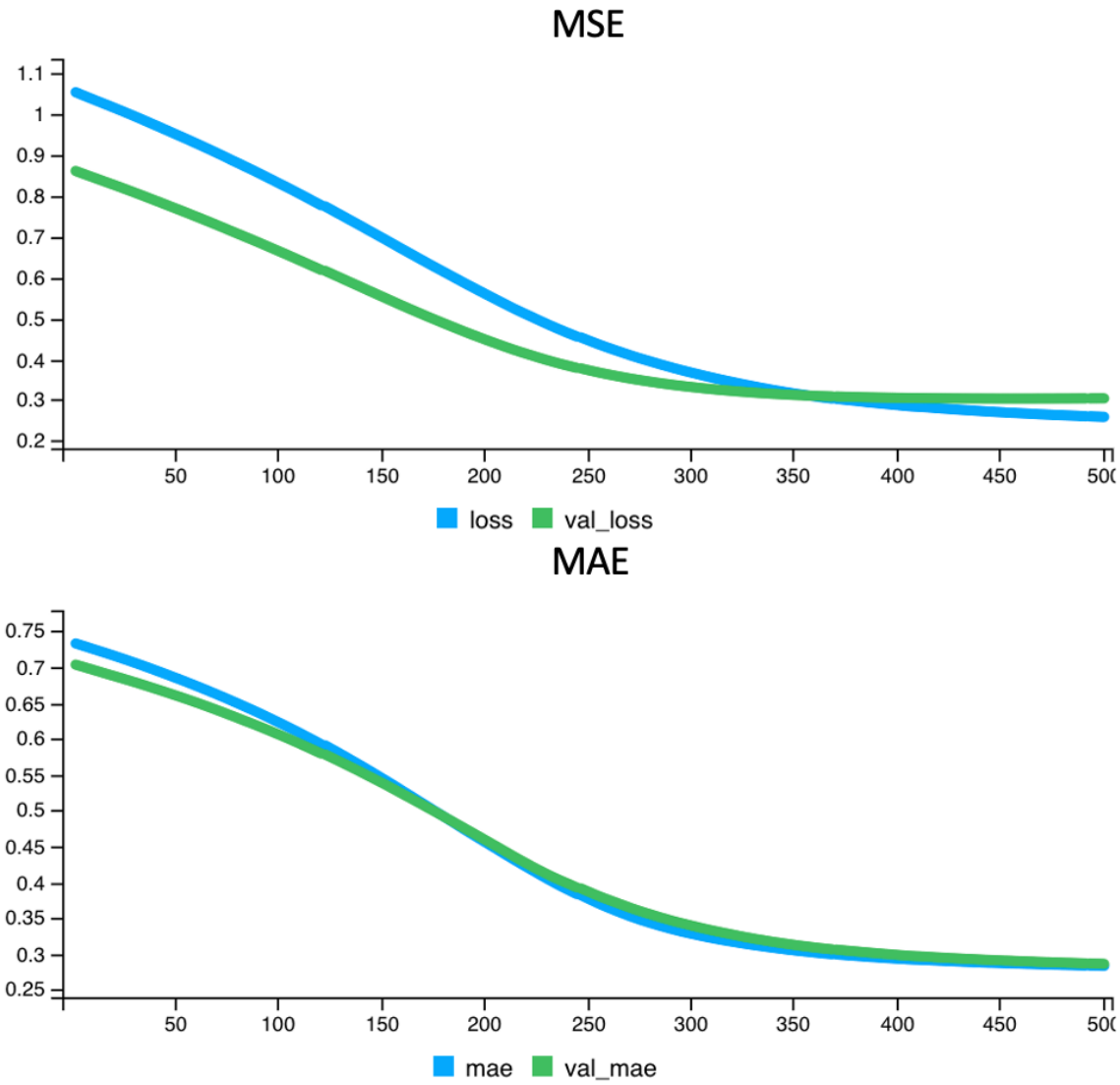

Figure 9: The MSE and MAE during training and validation processes (New model)

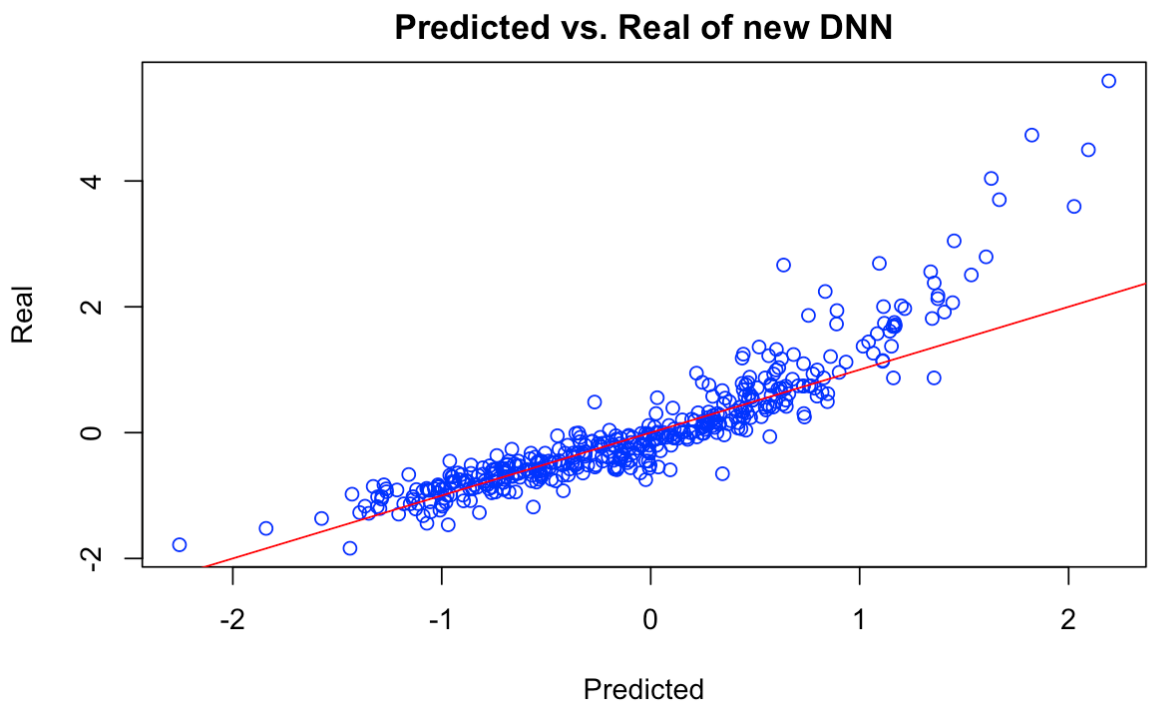

Figure 10: Comparison between predicted and real values (New model)

Ultimately, this is the 12 dominant factor that we screened for, in descending order of the Sobol' indices:

Table 2: Name and significance of the dominant variables

Variable Name	Meaning
GrLivArea	Above grade (ground) living area square feet
OverallQual	Rates the overall material and finish of the house
YearRemodAdd	Remodel date (same as construction date if no remodeling or additions)
GarageArea	Size of garage in square feet
1stFlrSF	First Floor square feet
Fireplaces	Number of fireplaces
KitchenQual=TA	Kitchen quality, KitchenQual=TA means the quality is Typical/Average
TotalBsmtSF	Total square feet of basement area
LotArea	Lot size in square feet
OpenPorchSF	Open porch area in square feet
BsmtFullBath	Basement full bathrooms
BsmtFinSF1	Type 1 finished square feet

As can be seen from the above table, after multiple screening using the dummy parameter method,

the final factors obtained are very similar to the prior knowledge we have from our daily experience, for example, above grade (ground) living area and overall quality of the house are the most important determinants of the house price. This also proves the reasonableness of using GSA with dummy parameter method to optimise the model to a certain extent.

The final DNN model contains 1 input layer with 12 input features, 1 output layer with one single output, 3 hidden layers with 200, 100, 50 each.

In the training process, 500 epochs were planned to processed (400 epochs actually) and batch size 32 have been set for our new DNN model, which shows in Figure 11. Model converged 394 epochs, resulting in train loss = 0.3149 and validation loss = 0.3819, then validation error didn't decrease and increase again.

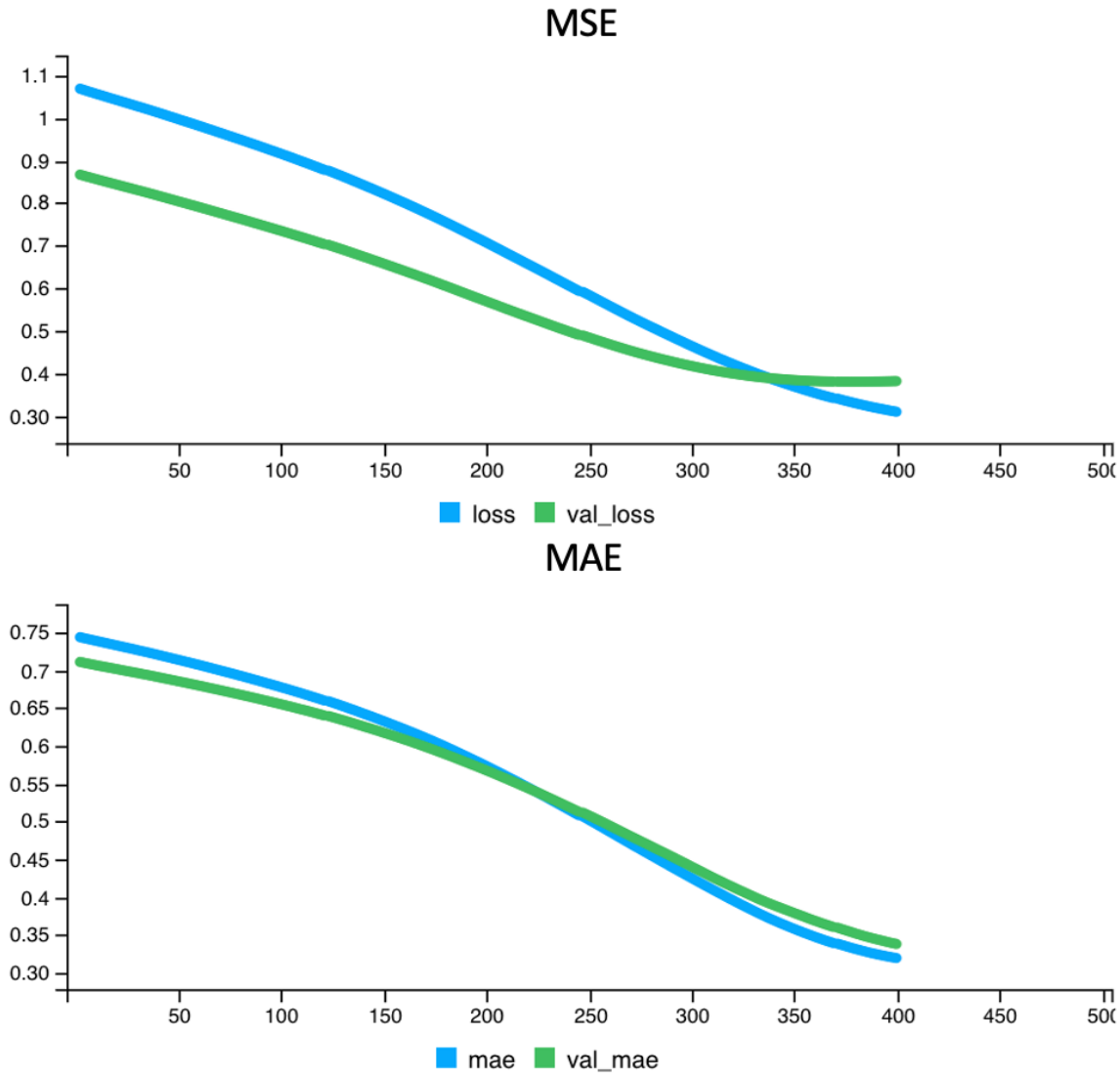

Figure 11: The MSE and MAE during training and validation processes (Final model)

Then evaluate model performance on test set, the MSE = 0.2606 and MAE = 0.3039. The comparison results between the prediction set and the training set are shown in the figure 12:

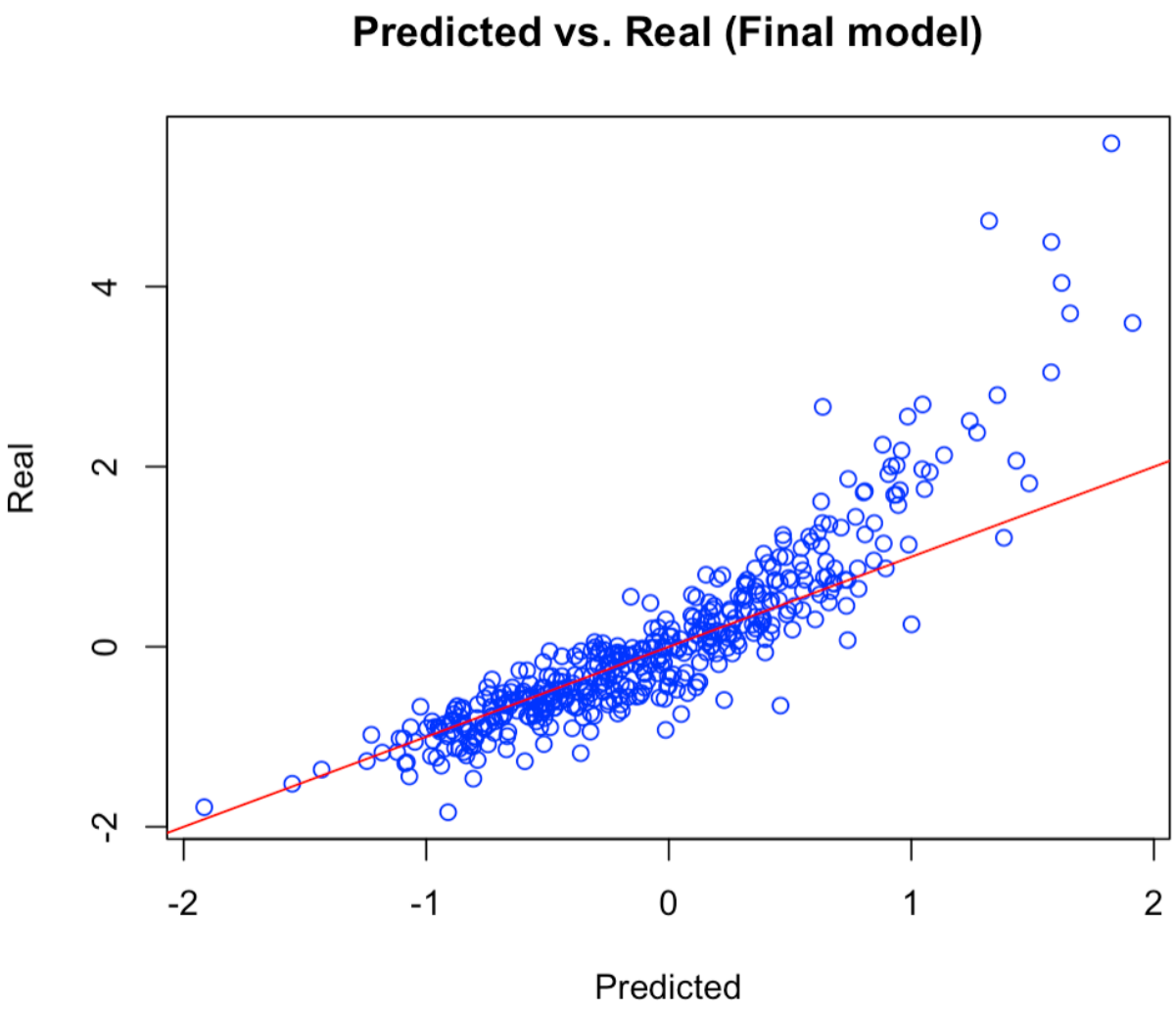

Figure 12: Comparison between predicted and real values (Final model)

Subsequently we can compare all fitted models (the model in the iterative process is represented by the first improved model):

Model	Total number of parameters	Number of trained epochs at convergence	MSE on test set	MAE on test set	Number of input features	Number of variables
Old model	92251	800	0.2013	0.2621	326	80
First improved model	40601	460	0.2134	0.2697	76	55
Final model	40601	394	0.2606	0.3039	12	12

Overall, as the number of optimisations based on the GSA with dummy parameter method increased, the performance of the model on the test set remained at the same level (albeit reduced); the complexity of the model and the number of epochs required for training was significantly reduced; and the number of input features to the model was significantly reduced, and when the optimisation is complete, the screened dominant factor matches our prior knowledge.

Then processing GSA on final model, and result shows in Figure 13.

Compare to the Sobol' indices of dummy parameter in Figure 14, it can be found that the Sobol' values of all factors are greater than dummy's, so we can assume that each factor in the model has an effect on the outputs,and since we have optimized from all the factors step-by-step to get these 12 factors, we can assume that this is all of the factors that can be identified as having an effect under this optimization method (no omissions).

And we can notice that the four factors with the largest Sobol' values are *Above grade (ground) living area*, *overall quality of the house, Remodel date* and *Size of garage*, and the sum of their total order Sobol' indices is 0.659, compared to the sum of the initial model which is 0.158. It means that final model suggests that 65.9% of the variance of house prices is determined by these four factors, while original model suggests that only 15.8% of the variance of house prices is determined by these four factors. In contrast, the final model's inference of these four factors is closer to our common perception that "room size and house quality are the main determinants of house prices". Therefore, in this case, using GSA with the dummy parameter method of optimization can make the DNN model more in line with our prior knowledge, which justifies the optimization method from another perspective

Thus we can argue that if we optimise the model using the GSA results, we can simplify the model significantly and reduce the arithmetic requirements while maintaining the generalisation ability of the model, and (with the possibility of applying it to other cases, as yet unproven) obtain a simplified model that is more in line with our everyday experience.

	original	bias	std.error	low.ci	high.ci	sensitivity	parameters
1	0.029	-0.001	0.018	-0.006	0.065	Si	LotArea
2	0.158	0.004	0.039	0.078	0.229	Si	OverallQual
3	0.117	0.001	0.036	0.046	0.185	Si	YearRemodAdd
4	0.006	0.000	0.007	-0.009	0.020	Si	BsmtFinSF1
5	0.048	-0.001	0.024	0.002	0.096	Si	TotalBsmtSF
6	0.111	-0.001	0.032	0.049	0.175	Si	X1stFlrSF
7	0.258	0.000	0.053	0.155	0.363	Si	GrLivArea
8	0.026	0.001	0.015	-0.003	0.055	Si	BsmtFullBath
9	0.084	0.000	0.030	0.025	0.142	Si	Fireplaces
10	0.113	-0.003	0.035	0.047	0.185	Si	GarageArea
11	0.028	0.001	0.018	-0.007	0.062	Si	OpenPorchSF
12	0.050	0.001	0.023	0.003	0.094	Si	KitchenQual=TA
13	0.031	0.000	0.001	0.029	0.033	Ti	LotArea
14	0.149	0.000	0.004	0.141	0.156	Ti	OverallQual
15	0.114	0.000	0.003	0.108	0.119	Ti	YearRemodAdd
16	0.005	0.000	0.000	0.005	0.005	Ti	BsmtFinSF1
17	0.054	0.000	0.001	0.051	0.056	Ti	TotalBsmtSF
18	0.092	0.000	0.002	0.087	0.096	Ti	X1stFlrSF
19	0.259	0.001	0.007	0.245	0.271	Ti	GrLivArea
20	0.022	0.000	0.001	0.021	0.023	Ti	BsmtFullBath
21	0.075	0.000	0.002	0.071	0.079	Ti	Fireplaces
22	0.128	0.000	0.003	0.121	0.134	Ti	GarageArea
23	0.029	0.000	0.001	0.027	0.030	Ti	OpenPorchSF
24	0.054	0.000	0.001	0.051	0.057	Ti	KitchenQual=TA

Figure 13: Sobol' indices of final model

	original	bias	std.error	low.ci	high.ci	sensitivity	parameters
1	0.004964883	-1.497718e-06	0.0001191643	0.004732823	0.005199939	Si	dummy
2	0.000000000	-5.665518e-05	0.1043529147	0.000000000	0.166970766	Ti	dummy

Figure 14: Sobol' indices of dummy parameter in final model

5 Sobol' indices in a Bayesian perspective

From the above analysis, the Sobol' indices derived from the FANOVA decomposition should be a constant value. However, we can try to extend it under the Bayesian perspective.

When we take the Sobol' indices of a factor in the model to be a random variable, we need to point out why it is uncertain rather than a fixed value as above. Although in general perspective, DNN model parameters $\boldsymbol{\theta} = (W, \boldsymbol{b})$, which denotes all weights and biases respectively, derived using methods such as gradient descent should be a set of fixed values, training process is stochastic due to random decisions as the order of the data, random initialization, or random regularization as augmentation or dropout and the randomness in the training process in general leads to different local optima $\hat{\boldsymbol{\theta}}$ [15]. Therefore, we can regard values of all parameters as random variables, and the Sobol' indices are function of those random variables, and thus the Sobol' indices is also random variable rather than a fixed value in this perspective.

Therefore, we can use posterior inferences of the Sobol' indices rather than a numerical value to describe the importance of factors.

5.1 Posterior of Sobol' indices

Let $\boldsymbol{S} = S(\boldsymbol{\theta})$ denotes Sobol' indices, while function $S(\cdot)$ decided by the structure of our fitted DNN model. Since Sobol' indices are function of DNN model parameters $\boldsymbol{\theta} = (W, \boldsymbol{b})$, which are rondom, $\boldsymbol{S}$ is also random variable. Therefore, we can revisit the Sobol' indices in a Bayesian perspective, e.g. now we can try to use posterior inference of Sobol' indices rather a fixed value to understand the DNN model. But it is difficult especially we want to set a informative prior on Sobol' indices.

Recall that in the case where the white noise is Gaussian, we can think that the data is sampled from a normal distribution, where the mean is fitted by the DNN model and the variance is the variance of the Gaussian noise. At this point we can think of the data in the dataset as being generated from the above normal distribution, which is mathematically expressed as a Bayesian model:

$$\begin{aligned} data\ layer &: y_i \sim N(f(\boldsymbol{x}_i; \boldsymbol{\theta}), \sigma^2) \\ prior\ layer &: \boldsymbol{S} = S(\boldsymbol{\theta}) \sim \pi(\boldsymbol{S}) \\ & \pi(\sigma^2) \propto \frac{1}{\sigma^2} \end{aligned}$$

where $f(\ \cdot\ ; \boldsymbol{\theta})$ denotes the DNN model we fitted, σ^2 denotes the variance of white noise.

In this case, the posterior of Sobol' indices $\boldsymbol{S}$ according to Bayes theorem:

$$\pi(\boldsymbol{S}|\sigma^2, \boldsymbol{y}) \propto \pi(\boldsymbol{S}) P(\boldsymbol{y}|\boldsymbol{S}, \sigma^2)$$

in order to get the posterior of Sobol' indices, we need the prior of Sobol' indices $\pi(\boldsymbol{S})$ and the likelihood $P(\boldsymbol{y}|\boldsymbol{S}, \sigma^2)$.

However we cannot obtain the likelihood function directly. Since we fitted the DNN model, we

can get likelihood based on model parameters $\boldsymbol{\theta} = (W, \boldsymbol{b})$, which is $P(\boldsymbol{y}|\boldsymbol{\theta}, \sigma^2)$. However, the Sobol' indices $\boldsymbol{S} = S(\boldsymbol{\theta})$ are estimated by the method described above, which means likelihood based on Sobol' indices $P(\boldsymbol{y}|\boldsymbol{S}, \sigma^2)$ has no explicit analytical expression.

So the first difficulty we facing is we cannot get the likelihood $P(\boldsymbol{y}|\boldsymbol{S}, \sigma^2)$, which is conditional on our setting some particular form of prior for the Sobol' indices $\boldsymbol{S}$.

So thinking in the opposite direction, our goal is to get the posterior distribution of $\boldsymbol{S}$. Since we don't get the $\boldsymbol{S}$-based likelihood $P(\boldsymbol{y}|\boldsymbol{S}, \sigma^2)$, can we use the known $\boldsymbol{\theta}$-based likelihood, $P(\boldsymbol{y}|\boldsymbol{\theta}, \sigma^2)$, to solve for the posterior distribution of $\boldsymbol{S}$?

Then we realised that we had hit another wall: the "black box" of DNN models. Since we don't understand the single parameter in DNN model, for example we cannot state the significance of a particular weight, and thus it is difficult for us to set a prior distribution for it other than a non-informative prior like $N(0, 1000)$. However if we just set non-informative prior on all parameters $\boldsymbol{\theta}$, since our DNN model has a large number of parameters, under the action of the central limit theorem, the combination of the prior of each parameter will make the prior of Sobol' indices $\boldsymbol{S}$ also a normal distribution with mean 0 and large variance, in other words the prior of $\boldsymbol{S}$ can only be a non-informative prior under this condition.

If we want to give Sobol indices $\boldsymbol{S}$ an informative prior such as the beta distribution as in the previous section, we need to design the priors of the individual model parameters very delicately so that they can be combined in such a way that the prior distribution of $\boldsymbol{S}$ is exactly the beta distribution. However, because of the "black box" nature of DNN model, this is almost impossible in practice.

To ascertain the posterior distribution of Sobol' indices $\boldsymbol{S}$, it is requisite to acquire both the prior distribution of $\boldsymbol{S}$ and an unambiguous formulation of its corresponding likelihood function. But we are in a dilemma: when we set a prior on $\boldsymbol{S}$ we cannot get a likelihood function with an explicit expression; when we look at a likelihood function with an explicit expression we cannot make $\boldsymbol{S}$ have a meaningful prior other than a non-informative prior like $N(0, 1000)$.

It seems that we have reached a dead end, but approximate Bayesian computation (ABC) can be obtained a posterior even without obtaining an explicitly expressed likelihood function. Therefore our target now is: firstly setting prior on parameter function $\boldsymbol{S}$, and obtaining its posterior by using ABC method.

Next we present the ABC method.

5.2 Approximate Bayesian Computation

As we cannot get likelihood directly, and our model is intractable but can be simulated from, we can apply Approximate Bayesian Computation (ABC) methods which as a rejection technique bypassing the computation of the likelihood function via a simulation from the corresponding distribution [16].

The general idea of ABC method can be expressed as [17]:

- Firstly we should have a data generate model $f(\cdot|\theta)$ which can generate pseudo data $\hat{y}$ when giving specific value of parameter θ.

- Sampling parameter value θ_i from its prior distribution $\pi(\theta)$.
- Using parameter value θ_i and data generate model $f(\cdot|\theta)$ to generate pseudo data $\hat{y}$.
- Compare pseudo data $\hat{y}$ with real data y, if they are "close" or similar enough, the corresponding parameter value θ_i is probably a posterior sample of θ.

And the figure below shows the original ABC algorithm:

Algorithm 1 Likelihood-free rejection sampler 1

for $i = 1$ to N **do**
  repeat
    Generate $\boldsymbol{\theta}'$ from the prior distribution $\pi(\cdot)$
    Generate $\mathbf{z}$ from the likelihood $f(\cdot|\boldsymbol{\theta}')$
  until $\mathbf{z} = \mathbf{y}$
  set $\boldsymbol{\theta}_i = \boldsymbol{\theta}'$,
end for

(ZHU Wanchuang, JI Chunlin et al., 2019)

From the above, it can be found that the original ABC algorithm is essentially a special kind of reject-accept sampling algorithm, whose acceptance condition is a sample being 'almost' identical to the (true) observed sample.

5.3 The ABC method of the two rejection conditions

Back to our target: setting prior on parameter function $\boldsymbol{S}$, and obtaining its posterior by using ABC method.

We can see that although the original ABC method bypasses the difficulty of finding the likelihood function, at this point we encounter a new difficulty in the step of "sampling the model parameters from the prior distribution".

To be more specific, we only have the prior distribution of parameter function $\boldsymbol{S} = S(\boldsymbol{\theta})$. However, for our DNN model, the model parameter that satisfies the condition of "generating pseudo-samples given a specific value of model parameters" should be $\boldsymbol{\theta} = (W, \boldsymbol{b})$. Therefore we should make some adjustment on original ABC algorithm.

Notice that the central idea of the ABC approach is "if you have a data generating model, you can generate pseudo-data to compare with the conditions to get the desired sample". And the process of estimating Sobol' indices from a set of specific values of parameters $\boldsymbol{\theta}$ just satisfy the "data generate

model" required in ABC method, except that what is generated changes from pseudo data to pseudo Sobol' indices.

Therefore, we can process ABC method twice. Firstly we use ABC to generate samples of $\boldsymbol{\theta}$ which has the Sobol' indices value that belongs to prior of $\boldsymbol{S}$ (for example still beta distribution). Then we use the samples of $\boldsymbol{\theta}$ selected from step 1 to generate pseudo data by DNN model, and comparing pseudo data with real data to determine which samples of $\boldsymbol{\theta}$ are satisfied with all condition. The samples of parameter $\boldsymbol{\theta}$ thus obtained are both such that their Sobol' indices prior obeys the beta distribution (first screening) and generate pseudo-data that approximate the actual data (second screening). Thus we indirectly obtain a posterior sample of Sobol' indices $\boldsymbol{S}$ by filtering the eligible samples of parameter $\boldsymbol{\theta}$.

And we can expand this algorithm to all parameter function not only Sobol' indices:

Let $\vec{\theta}$ denotes all parameters of a model, $\vec{\phi}$ denotes a function satisfy $\vec{\phi} = g(\vec{\theta})$, $f(\cdot|\vec{\theta})$ denotes a data generate model, so:

Algorithm Likelihood-free rejection sampler for variables function

for $i = 1$ *to* N **do**
  repeat
    repeat
      Generate samples $\vec{\theta}'$
      Calculate $\vec{\phi}' = g(\vec{\theta}')$
    until $\vec{\phi}'$ is accepted as sample from prior $\pi(\vec{\phi})$
    generate $\vec{z}$ from $f(\cdot|\,\vec{\theta}')$
  until $\vec{z}$ is similar enough with $\vec{y}$
  set $(\vec{\theta}_i\,, \vec{\phi}_i) = (\,\vec{\theta}'\,, \vec{\phi}')$
end for

Figure 15: 2-step ABC for variable function

5.4 Test the above algorithm on a simple model

Before applying our algorithm to a complex model, we can try it on the simplest model, such as a polynomial model, to see if it works.

5.4.1 Testing on a polynomial model

Let us test the two-step ABC algorithm on the following simple polynomial function:

$$Y = 3X_1^2 + X_2X_3 - 2X_4, \quad X_1, ..., X_4 \sim U(0, 1)$$

Due to the simplicity of its form, we can directly derive the Sobol' indices of the factors of the function:

	First order	Total order
X_1	0.677	0.677
X_2	0	0.041
X_3	0	0.041
X_4	0.282	0.282

Now let's design the fitted model. In order to minimise the number of model parameters for our subsequent implementation of the two-step ABC algorithm, we will fit the following model:

$$y = a_1 X_1^2 + a_2 X_2 X_3 + a_3 X_4$$

where $\boldsymbol{a} = (a_1, a_2, a_3)^T$ denotes the parameters in model.

5.4.2 Screening parameter samples based on prior of Sobol' indices

We will test the algorithm by using the Sobol' index of X_4 as an example. Since the Sobol' index can be viewed as the percentage of the factor's contribution to the variance of the output, we can use the beta distribution as its a prior.

In addition, from the calculations above, it is clear that the mean value of X_4's Sobol' index should be close to 0.282, so we might as well set its prior to $Beta(300, 700)$, which satisfies the probability that the Sobol' value falls in $[0.25, 0.3]$ with a probability of about 0.5. And we make the peak of the prior deviate slightly from 0.282 (the reason for the only slight deviation is to prevent a plunge in computational efficiency caused by rejecting too many samples), thus checking whether our algorithm can really select posterior samples which close to 0.282.

We then perform a first ABC to filter out samples of model parameters $\boldsymbol{a} = (a_1, a_2, a_3)^T$ that satisfy prior of X_4's Sobol' index is $Beta(300, 700)$, the specific algorithm is as following algorithm 3 in Figure 16:

Algorithm 3 Parameter sampler conditional on prior

for $i = 1$ to N **do**
  repeat
    Generate samples $\boldsymbol{a}' = (a_1', a_2', a_3')^T$ from there prior $\pi(\cdot)$
    Generate X_4's Sobol' index bootstrap samples $\boldsymbol{S}'$ based on sample $\boldsymbol{a}'$
  until $d\{\rho(\boldsymbol{S}'), \rho(\boldsymbol{S})\} \leq \epsilon$
  set $\hat{a}_i = \boldsymbol{a}'$
    $\widehat{\boldsymbol{S}}_i = \boldsymbol{S}'$
end for

Figure 16: Parmaeter sampler based on condition of prior of Sobol' indices

Where the parameters of the algorithm are:

- ρ, a function defining a statistic which summarize the information of samples. Here we will use sufficient statistic $\sum log(x_i)$, $\sum log(1 - x_i)$ for beta distribution.
- $d > 0$, Euclidean distance,
- $\epsilon > 0$, a tolerance level,
- $\boldsymbol{S}$, samples generated from $Beta(300, 700)$.

The specific code implementation is then:

- First generate N=10000 samples in the prior $a_1 \sim U(2.75, 3.25)$, $a_2 \sim U(0.75, 1.25)$, $a_3 \sim U(-2.25, -1.75)$ using Latin hypercube sampling.
- Calculate the Sobol' value of X_4 under each sample using bootstrap method for R=200 each.
- Calculate the Euclidean distance between $(\sum log(S_i'), \sum log(1 - S_i'))$ of the generated sample and $(\sum log(S_i), \sum log(1 - S_i))$ of the beta prior sample, if it is less than the rejection level ϵ then accept the generated sample and record the corresponding Sobol' value sample.

After processing above steps, we have obtained a sample of parameters $\boldsymbol{a}'$ that enable the Sobol' index of X_4 to conform to the prior $Beta(300, 700)$, with the following distribution of samples shows in Figure 17.

Also we can learn from the following figure that the parameter samples screened after this step have a corresponding Sobol' index of X_4 which has a $Beta(300, 700)$ prior shows in Figure 18.

Before processing algorithm above, we can only get the prior of Sobol' index shaped as depicted by the purple curve above, when the prior of model parameters are uniform distributions. But now we get samples of the parameters such that the Sobol' index approximately obeys a $Beta(300, 700)$. so we can assume our algorithm to be valid in this step!

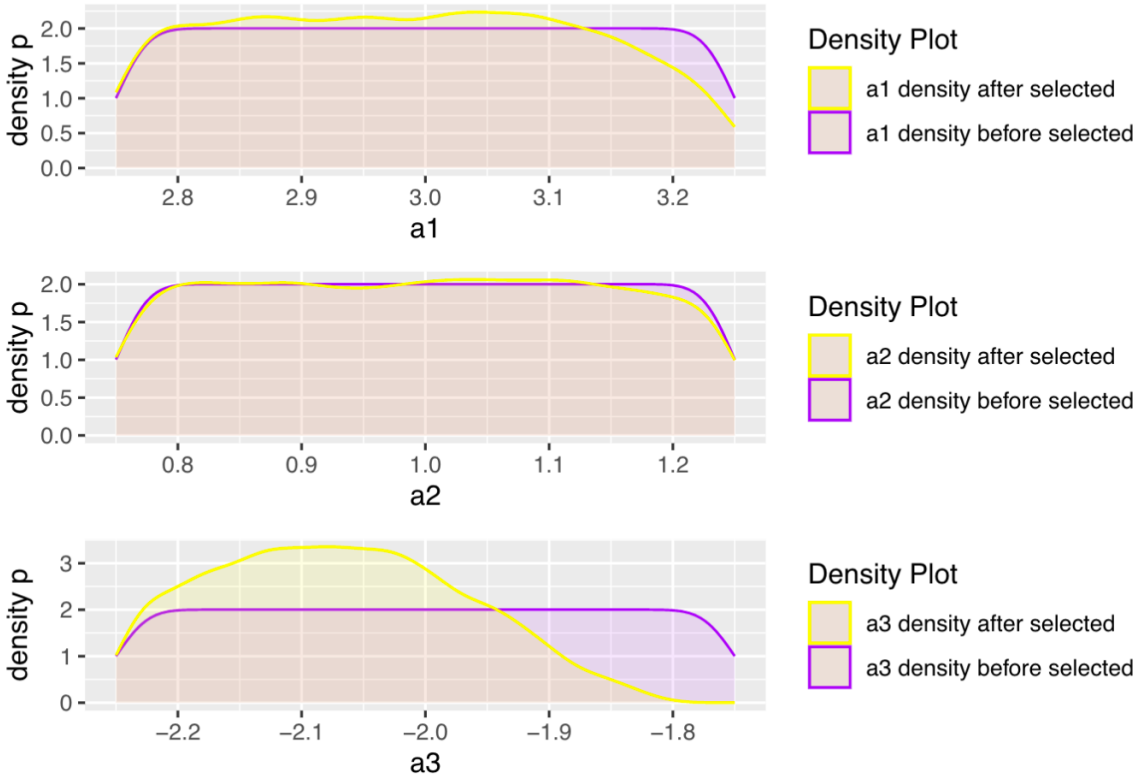

Figure 17: Parameter distributions of post-screening polynomial models conditional on Sobol's prior

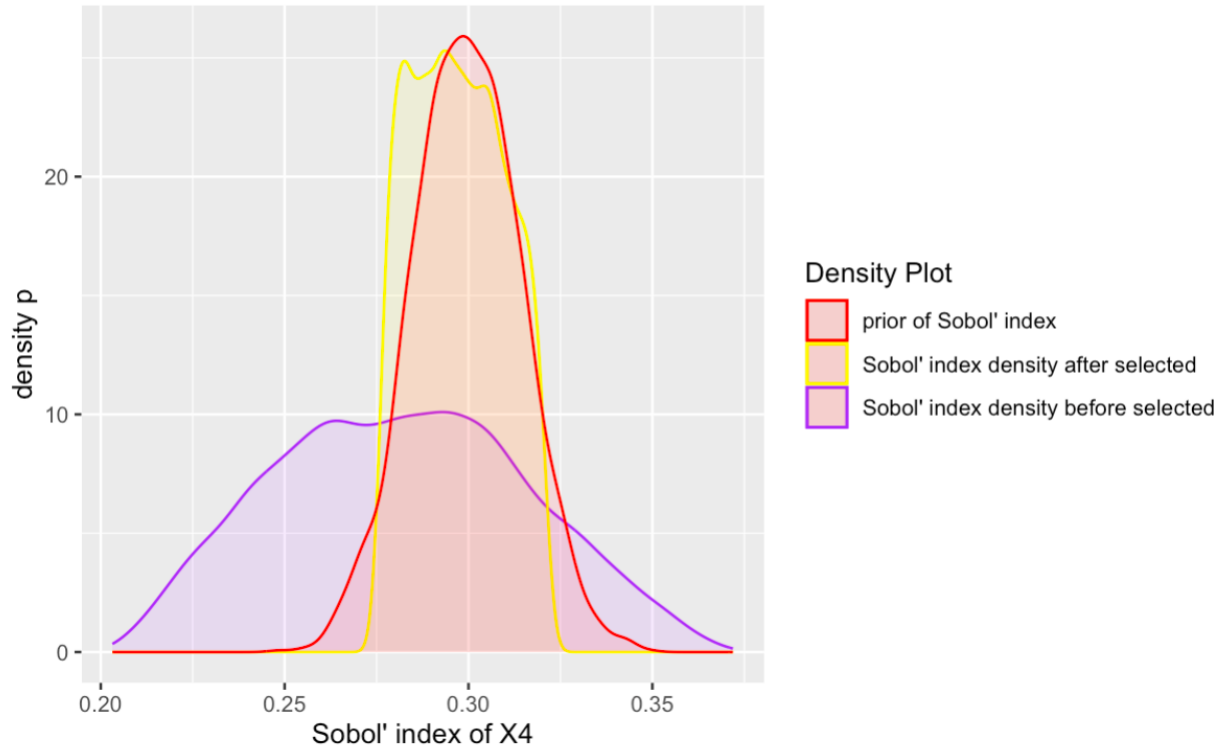

Figure 18: Prior and selected samples of Sobol' index

5.4.3 Screening parameter samples based on data of polynomial model

After we have obtained parameter samples that satisfy the condition for prior of Sobol' index, we can further perform another ABC to obtain parameter samples that generate pseudo-data similar to the actual data, i.e., posterior samples determined by the actual data.

Eventually we obtained the parameter distributions shown below, where the purple curve represents the initial uniform distribution of the parameters, the yellow line represents the prior distribution of the parameters that satisfy the Sobol' index requirement, and the red line represents the posterior distribution of the parameters in Figure 19.

We can then compare the prior and posterior distributions of the Sobol' index for X4 and compare it to the theoretical Sobol' value which represents by black vertical line in Figure 20.

From the above figure we can conclude that our algorithm is able to obtain good posterior samples of Sobol' index as its peaks coincide with the theoretical values obtained from our calculation.

Overall, our algorithm avoids the dilemma mentioned in the previous section. It can obtain samples of model parameters that satisfy the condition that "the prior of the function of the model parameters (e.g., Sobol' index) is our predetermined prior" and further obtain the posterior of the function of the model parameters without specifically designing the prior of the model parameters.

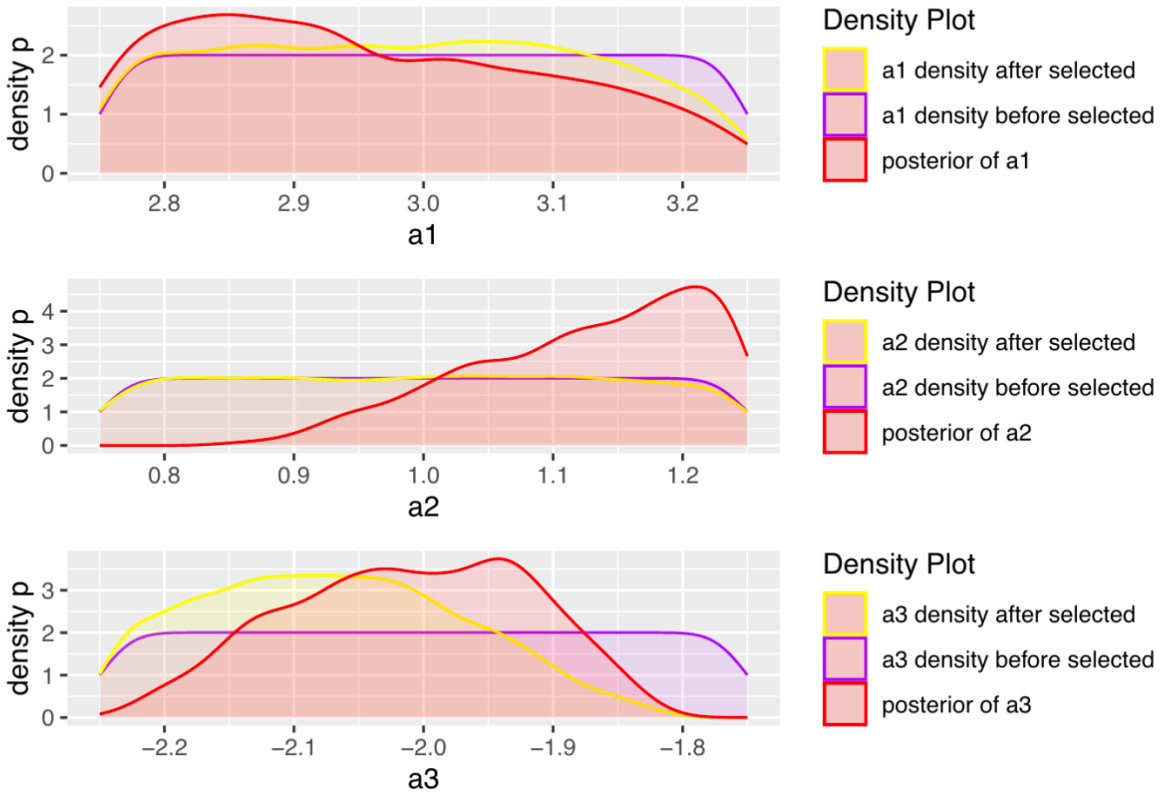

Figure 19: Posterior and prior samples of model parameters

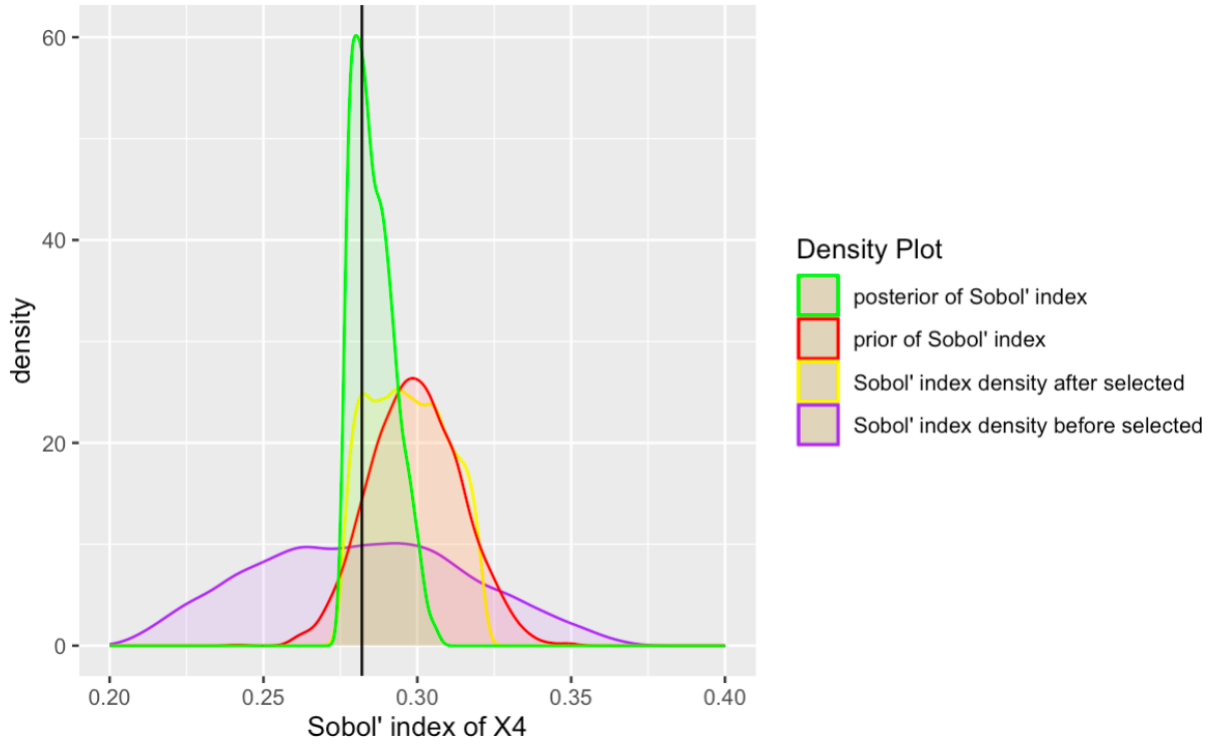

Figure 20: Posterior and prior samples of Sobol' index

6 Future work

Although we have devised an algorithm that is capable of obtaining the posterior distribution of the Sobol' indices, however, in practice it is very difficult to obtain the desired results in complex models such as DNN due to the extreme inefficiency of the algorithm.

This inefficiency is caused by following reason:

A major limitation of the classical ABC framework is that the candidate samples of parameters are drawn from the prior distribution of the parameters, which generally results in a high rejection rate when the prior distribution differs significantly from the posterior distribution [17]. In our algorithm, the acceptance of a sample requires that the candidate meets both conditions, which leads to a much lower acceptance rate.

In addition, the curse of dimensionality leads to extreme inefficiencies in finding suitable samples in the parameter space. As an example, the first 4-hidden layer DNN model in the previous section has a total of 92,251 parameters, which means that using a rejection sampling method such as original ABC for such a high-dimensional space is extremely inefficient.

We also encounter the same difficulty in Bayesian neural networks (BNN), and correspondingly, the main solutions in the field of BNNs are variational inference and Markov Chain Monte Carlo (MCMC) etc [18]. Since our aim is to get posterior samples of functions on model parameters, we will focus more on finding inspiration in the MCMC field.

For MCMC methods, Hamiltonian Monte Carlo (HMC) is probably the only Bayesian inference algorithm that can remain efficient in high-dimensional parameter spaces [19]. Therefore we can consider using the HMC method to improve the two-step ABC method we designed in section 5.3. However, one reason why HMC is efficient is because it uses the gradient of the log-likelihood as additional information to optimise the iteration step, , whereas our ABC algorithm itself is built on likelihood-free conditions. Therefore this may seem to be contradictory.

However, some research has already been done in this direction. Edward Meeds et al. proposed Hamiltonian ABC (HABC), a set of likelihood-free algorithms that apply recent advances to Bayesian learning using Hamiltonian Monte Carlo (HMC) and stochastic gradients [20]. With the combination of stochastic gradients and the HMC method, in the presence of well-established gradient approximations, we can employ efficient HMC for sampling even on likelihood-free models. Therefore, integrating the HABC method into the improved ABC algorithm of section 5.3 will be able to help solve the inefficiencies caused by the curse of dimensionality, which will be a valuable future research direction.

In addition, although we adopt the same idea as the original ABC method, i.e., to design the improved algorithm by generating pseudo data to compare with the actual data to filter out the possible posterior samples of parameter, which probability implies that our improved algorithm is theoretically feasible as well, it still requires rigorous mathematical proofs. This is also a direction for further research in the future.

7 Conclusion

In this paper, we first fit a DNN model on house price data for the city of Ames, Iowa, USA. First we used all factors for fitting and obtained a DNN model with 4 hidden layers, which was generally well fitted except for some extreme values.

We then performed a Sobol' index-based GSA on this model and used the dummy parameter method to filter out 76 influential factors from the 326 input factors of the original model and used them to re-fit a simplified 3-hidden-layer DNN model, which we found to show the same level of model performance as original one, with the model complexity and the arithmetic required for training are substantially reduced.

We then repeated the steps above until we finally filtered out 12 factors, and the DNN models fitted on these 12 factors also all proved to be all influential after using the dummy parameter method, so we consider that all dominant factors have been selected. The performance of the final simplified model is still similar to that of the original model, and it can be concluded that it is feasible to optimise the model using GSA and dummy parameter method. In addition, the selected factors are similar to our prior knowledge, which can also prove the success of selecting the dominant factors from another perspective.

Finally, we adopt a Bayesian perspective, not by using point estimates of Sobol' indices but by using their posterior inferences to obtain a further understanding of our DNN model.

However, since 1) Sobol is essentially a function of the model parameters, resulting in our inability

to obtain its explicit expression 2) the DNN model itself is unusually complex, these two factors lead to our inability to obtain the likelihood function, and thus the posterior. In order to solve this problem, we proposed an idea of improving on the original approximate Bayesian computation by designing an improved ABC algorithm with two filtering conditions, which is theoretically capable of directly obtaining the posterior samples of Sobol' indices without obtaining likelihood, and thus carry out the posterior inference. And we prove the effectiveness of this algorithm on a simple polynomial model, which can indeed obtain the posterior samples of Sobol' index.

References

[1] Garson, G. David. 'Interpreting Neural-Network Connection Weights'. AI Expert 6, no. 4 (1 April 1991): 46–51.

[2] Dimopoulos, Yannis, Paul Bourret, and Sovan Lek. 'Use of Some Sensitivity Criteria for Choosing Networks with Good Generalization Ability'. *Neural Processing Letters* 2, no. 6 (1 December 1995): 1–4.

[3] Olden, Julian, and Donald Jackson. 'Illuminating the "Black Box": A Randomization Approach for Understanding Variable Contributions in Artificial Neural Networks'. *Ecological Modelling* 154 (1 August 2002): 135–50. https://doi.org/10.1016/S0304-3800(02)00064-9.

[4] Koh, Pang Wei, and Percy Liang. 'Understanding Black-Box Predictions via Influence Functions'. In *Proceedings of the 34th International Conference on Machine Learning*, 1885–94. PMLR, 2017. <https://proceedings.mlr.press/v70/koh17a.html>.

[5] Mohammadi, Hossein, Peter Challenor, and Clémentine Prieur. 'Variance-Based Global Sensitivity Analysis of Numerical Models Using R'. arXiv, 22 June 2022. <http://arxiv.org/abs/2206.11348>.

[6] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 'Deep Learning'. *Nature* 521, no. 7553 (May 2015): 436–44. <https://doi.org/10.1038/nature14539>.

[7] Zeiler, Matthew D., and Rob Fergus. 'Visualizing and Understanding Convolutional Networks'. In *Computer Vision – ECCV 2014*, edited by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, 818–33. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014. <https://doi.org/10.1007/978-3-319-10590-1_53>.

[8] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 'Multilayer Feedforward Networks Are Universal Approximators'. *Neural Networks* 2, no. 5 (1 January 1989): 359–66. https://doi.org/10.1016/0893-6080(89)90020-8.

[9] Shu, Hai, and Hongtu Zhu. 'Sensitivity Analysis of Deep Neural Networks'. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, no. 01 (17 July 2019): 4943–50. <https://doi.org/10.1609/aaai.v33i01.33014943>.

[10] Saltelli, Andrea, Tarantola, S., Campolongo, F., & Ratto, M. (2004). Sensitivity analysis in practice: A guide to assessing scientific models. Wiley. <https://doi.org/10.1002/0470870958>

[11] Sobol', I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. Mathematics and Computers in Simulation, 55 (1), 271–280. <https://doi.org/ht> tps://doi.org/10.1016/S0378-4754(00)00270-6

[12] Jansen, M. J. W. (1999). Analysis of variance designs for model output. Computer Physics Communications, 117 (1), 35–43. https://doi.org/https://doi.org/10.1016/S0010-4655(98)001544

[13] Lim, Wan Teng, Lipo Wang, Yaoli Wang, and Qing Chang. 'Housing Price Prediction Using Neural Networks'. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 518–22, 2016. <https://doi.org/10.1109/FSKD.2016.7603227>.

[14] Khorashadi Zadeh, Farkhondeh, Jiri Nossent, Ann van Griensven, and Willy Bauwens. 'Parameter Screening: The Use of a Dummy Parameter to Identify Non-Influential Parameters in a Global Sensitivity Analysis', 1 April 2017, 3878.

[15] Lakshminarayanan B, Pritzel A, Blundell C (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. In: Advances in neural information processing systems 30

[16] Marin, Jean-Michel, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. 'Approximate Bayesian Computational Methods'. *Statistics and Computing* 22, no. 6 (November 2012): 1167–80. <https://doi.org/10.1007/s11222-011-9288-2>.

[17] ZHU Wanchuang, JI Chunlin, DENG Ke. 'Recent progress of approximate Bayesian computation and its applications'[J]. Applied Mathematics and Mechanics, 2019, 40(11): 1179-1203.

[18] Jospin, Laurent Valentin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. 'Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users'. *IEEE Computational Intelligence Magazine* 17, no. 2 (May 2022): 29–48. <https://doi.org/10.1109/MCI.2022.3155327>.

[19] Neal, Radford M. Mcmc using hamiltonian dynamics. Handbook of Markov Chain Monte Carlo, 2, 2011.

[20] Meeds, Edward, Robert Leenders, and Max Welling. 'Hamiltonian ABC'. arXiv, 6 March 2015.

Appendix A: Variable Name Explanation

Table 5: Introduction of variables in house price dataset

Variable Name	Meaning
MSSubClass	Identifies the type of dwelling involved in the sale.
MSZoning	Identifies the general zoning classification of the sale.
LotFrontage	Linear feet of street connected to property
LotArea	Lot size in square feet
Street	Type of road access to property
Alley	Type of alley access to property
LotShape	General shape of property
LandContour	Flatness of the property
Utilities	Type of utilities available
LotConfig	Lot configuration
LandSlope	Slope of property
Neighborhood	Physical locations within Ames city limits
Condition1	Proximity to various conditions
Condition2	Proximity to various conditions (if more than one is present)
BldgType	Type of dwelling
HouseStyle	Style of dwelling
OverallQual	Rates the overall material and finish of the house
OverallCond	Rates the overall condition of the house
YearBuilt	Original construction date
YearRemodAdd	Remodel date (same as construction date if no remodeling or additions)
RoofStyle	Type of roof
RoofMatl	Roof material
Exterior1st	Exterior covering on house
Exterior2nd	Exterior covering on house (if more than one material)
MasVnrType	Masonry veneer type
MasVnrType	Masonry veneer type
MasVnrArea	Masonry veneer area in square feet
ExterQual	Evaluates the quality of the material on the exterior
ExterCond	Evaluates the present condition of the material on the exterior
Foundation	Type of foundation
BsmtQual	Evaluates the height of the basement
BsmtCond	Evaluates the general condition of the basement
BsmtExposure	Refers to walkout or garden level walls
BsmtFinType1	Rating of basement finished area
BsmtFinSF1	Type 1 finished square feet
BsmtFinType2	Rating of basement finished area (if multiple types
BsmtFinSF2	Type 2 finished square feet
BsmtUnfSF	Unfinished square feet of basement area

Variable Name	Meaning
TotalBsmtSF	Total square feet of basement area
Heating	Type of heating
HeatingQC	Heating quality and condition
CentralAir	Central air conditioning
Electrical	Electrical system
1stFlrSF	First Floor square feet
2ndFlrSF	Second floor square feet
LowQualFinSF	Low quality finished square feet (all floors)
GrLivArea	Above grade (ground) living area square feet
BsmtFullBath	Basement full bathrooms
BsmtHalfBath	Basement half bathrooms
FullBath	Full bathrooms above grade
HalfBath	Half baths above grade
Bedroom	Bedrooms above grade (does NOT include basement bedrooms)
Kitchen	Kitchens above grade
KitchenQual	Kitchen quality
TotRmsAbvGrd	Total rooms above grade (does not include bathrooms)
Functional	Home functionality (Assume typical unless deductions are warranted)
Fireplaces	Number of fireplaces
FireplaceQu	Fireplace quality
GarageType	Garage location
GarageYrBlt	Year garage was built
GarageFinish	Interior finish of the garage
GarageCars	Size of garage in car capacity
GarageArea	Size of garage in square feet
GarageQual	Garage quality
GarageCond	Garage condition
PavedDrive	Paved driveway
WoodDeckSF	Wood deck area in square feet
OpenPorchSF	Open porch area in square feet
EnclosedPorch	Enclosed porch area in square feet
3SsnPorch	Three season porch area in square feet
ScreenPorch	Screen porch area in square feet
PoolArea	Pool area in square feet
PoolQC	Pool quality
Fence	Fence quality
MiscFeature	Miscellaneous feature not covered in other categories
MiscVal	$Value of miscellaneous feature
MoSold	Month Sold (MM)
YrSold	Year Sold (YYYY)
SaleType	Type of sale

Variable Name	Meaning
SaleCondition	Condition of sale

Appendix B: Code

1.Code for section 4.1.3

```
#read data from csv file
TrainSet <- read.csv("/Users/zxj/Desktop/coding/dissertation/house-prices-advanced-regression-te

#split data into contaiong numeric and category variables
TrainSetNumeric <- TrainSet[ , -1] %>% select_if(is.numeric)
TrainSetCategory <- TrainSet[ , -1] %>% select_if(is.character)

#transfer category variables via onehot encoding
encoder <- onehot(as.data.frame(TrainSetCategory),
                                  stringsAsFactors = TRUE, addNA = TRUE, max_levels = 30)
TrainSetCategoryOnehot <- as.data.frame(predict(encoder, as.data.frame(TrainSetCategory)))

#normalize all numeric variables and output
yPre <- TrainSetNumeric$SalePrice
XPre <- TrainSetNumeric[,-37]
X <- XPre
for(j in 1:36){
  mu_j <- mean(XPre[ , j])
  var_j <- var(XPre[ , j])
  X[ , j] <- (XPre[ , j] - mu_j) / sqrt(var_j)
}

y <- (yPre - mean(yPre)) / sqrt(var(yPre))
X <- X[, !names(X) %in% c("LotFrontage", "MasVnrArea", "GarageYrBlt")]
Xtotal <- cbind(as.data.frame(X), TrainSetCategoryOnehot)

#split data into train and test set
set.seed(228)
SampleIndices <- sample(1:1460, 0.7*1460)
XTrainTotal <- Xtotal[SampleIndices, ]
XTestTotal <- Xtotal[-SampleIndices, ]
yTrain <- y[SampleIndices]
yTest <- y[-SampleIndices]
```

2.Code for section 4.2.2

```
#fit the first DNN model for regression
set.seed(228)

#build up structure of DNN
```

```
model3 <- keras_model_sequential()
model3 %>%
  layer_dense(units = 200, kernel_initializer='normal', activation='relu',
              input_shape = c(dim(XTrainTotal)[2])) %>%
  layer_dense(units = 100, kernel_initializer='normal', activation='relu') %>%
  layer_dense(units = 50, kernel_initializer='normal', activation='relu') %>%
  layer_dense(units = 25, kernel_initializer='normal', activation='relu') %>%
  layer_dense(units = 1, kernel_initializer='normal')

#compile a NN model:
model3 %>% compile(loss = "mse",
                   optimizer = "adadelta",
                   metrics = "mae")

#fit model:
model3 %>%
  fit(as.matrix(XTrainTotal),
      yTrain,
      epochs = 800,
      batch_size = 32,
      validation_split = 0.2)

#check model on test set
model3 %>% evaluate(as.matrix(XTestTotal), yTest)
yPred <- model3 %>% predict(as.matrix(XTestTotal))

#plot the graph
plot(yPred, yTest,
     xlab = "Predicted", ylab = "Real", main = "Predicted vs. Real", col = "blue")
abline(0, 1, col = "red")  # Add a diagonal line for reference
```

3.Code for section 4.3

```
#process Sobol' SA on first DNN model and show the result
d <- dim(XTrainTotal)[2]
N <- 3500
R <- 500
params <- paste(names(XTrainTotal), sep = "") #generate the names of variables
mat <- sobol_matrices(N = N, params = params,
                      type = "LHS") #generate samples based on Sobol Latin hypercube method
Y <- model3 %>% predict(mat)
sensobol_ind <- sobol_indices(Y = Y, N = N, params = params, boot = TRUE, R = R)
cols <- colnames(sensobol_ind$results)[1:5]
```

```
sensobol_ind$results[, (cols):= round(.SD, 3), .SDcols = (cols)]
sensobol_ind_model3$results[sensobol_ind_model3$results$sensitivity == "Si"]
```

4.Code for section 4.4.2

```
#Sobol' indices of dummy parameter in first DNN model
sensobol_ind_dummy <- sobol_dummy(Y = Y, N = N, params = params, boot = TRUE, R = R)
sensobol_ind_dummy

#select factors with larger Sobol' values than dummy's
FilteredSobolIndices <- sensobol_ind$results[sensobol_ind$results$sensitivity == "Si" & sensobol
FilteredFactorsName <- FilteredSobolIndices$parameters
FilteredFactorsName
```

5.Code for section 4.4.3

```
#refit the first improved DNN model for regression
set.seed(228)

model4 <- keras_model_sequential()
model4 %>%
  layer_dense(units = 200, kernel_initializer='normal', activation='relu',
              input_shape = c(length(FilteredFactorsName))) %>%
  layer_dense(units = 100, kernel_initializer='normal', activation='relu') %>%
  layer_dense(units = 50, kernel_initializer='normal', activation='relu') %>%
  layer_dense(units = 1, kernel_initializer='normal')

#compile a NN model:
model4 %>% compile(loss = "mse",
                   optimizer = "adadelta",
                   metrics = "mae")

#fit model:
model4 %>%
  fit(as.matrix(XTrainTotal[FilteredFactorsName]),
      yTrain,
      epochs = 500,
      batch_size = 32,
      validation_split = 0.2)

##check improved model on test set
model4 %>% evaluate(as.matrix(XTestTotal[FilteredFactorsName]), yTest)
yPred <- model4 %>% predict(as.matrix(XTestTotal[FilteredFactorsName]))
```

```r
#plot the graph
plot(yPred, yTest,
     xlab = "Predicted", ylab = "Real", main = "Predicted vs. Real", col = "blue")
abline(0, 1, col = "red")  # Add a diagonal line for reference
```

6.Code for section 4.4.4

```r
#loop until there is no non-influential factors in model
while (NumberOfNonInfluentialFactors > 0) {
  #fit new model until condition satisfied-----------------------------------------
  set.seed(228)

  model5 <- keras_model_sequential()
  model5 %>%
    layer_dense(units = 200, kernel_initializer='normal', activation='relu',
                input_shape = c(length(FilteredFactorsName))) %>%
    layer_dense(units = 100, kernel_initializer='normal', activation='relu') %>%
    layer_dense(units = 50, kernel_initializer='normal', activation='relu') %>%
    layer_dense(units = 1, kernel_initializer='normal')

  #compile a NN model:
  model5 %>% compile(loss = "mse",
                     optimizer = "adadelta",
                     metrics = "mae")

  #fit model:
  model5 %>%
    fit(as.matrix(XTrainTotal[FilteredFactorsName]),
        yTrain,
        epochs = 500,
        batch_size = 32,
        validation_split = 0.2)

  #process Sobol' SA---------------------------------------------------------------
  d <- length(FilteredFactorsName)
  N <- 3500
  R <- 500
  params <- paste(FilteredFactorsName, sep = "") #generate the names of variables
  mat <- sobol_matrices(N = N, params = params,
                        type = "LHS") #generate samples based on Sobol Latin hypercube method
  Y <- model5 %>% predict(mat)
  sensobol_ind <- sobol_indices(Y = Y, N = N, params = params, boot = TRUE, R = R)
  cols <- colnames(sensobol_ind$results)[1:5]
```

```
  sensobol_ind$results[, (cols):= round(.SD, 3), .SDcols = (cols)]

  #check if there is non-influential factors in model--------------------------------
  sensobol_ind_dummy <- sobol_dummy(Y = Y, N = N, params = params, boot = TRUE, R = R)
  FilteredSobolIndices <- sensobol_ind$results[sensobol_ind$results$sensitivity == "Si"
                                               & sensobol_ind$results$original >
                                                 sensobol_ind_dummy$original[1], ]
  FilteredFactorsName <- FilteredSobolIndices$parameters
  NumberOfNonInfluentialFactors <- dim(
    sensobol_ind$results[sensobol_ind$results$sensitivity == "Si" &
                           sensobol_ind$results$original <=
                           sensobol_ind_dummy$original[1], ])[1]
}


#check the performance of final model
model5 %>% evaluate(as.matrix(XTestTotal[FilteredFactorsName]), yTest)
yPred <- model5 %>% predict(as.matrix(XTestTotal[FilteredFactorsName]))

#plot the graph
plot(yPred, yTest,
     xlab = "Predicted", ylab = "Real", main = "Predicted vs. Real", col = "blue")
abline(0, 1, col = "red")  # Add a diagonal line for reference

#show the Sobol' indices of factors in final model
sensobol_ind$results
```

7.Code for section 5.4.2

```
#function for palynomial model
PolyFunc <- function(a, x){
  y <- a[1]*x[,1]^2 + a[2]*x[,2]*x[,3] + a[3]*x[,4]
  return(y)
}

#samples of model parameters by LHS method
ParamSample <- lhsDesign(10000, 3)$design
ParamSample[,1] <- ParamSample[,1] / 2 + 2.75
ParamSample[,2] <- ParamSample[,2] / 2 + 0.75
ParamSample[,3] <- ParamSample[,3] / 2 - 2.25

#calculate the summary statistics for first step ABC
TotalSobolX4RealSample <- rbeta(200, 300, 700)
```

```r
SufficientStat1 <- sum(log(TotalSobolX4RealSample))
SufficientStat2 <- sum(log(1 - TotalSobolX4RealSample))

#function for generating bootstrap samples of sobol
sobol_boot <- function(d, i, N, params, matrices, R, first, total, order, boot) {

  # Stopping rule to check concordance between estimators and sample matrix
  # ------------------------------------------------------------------

  ms <- "Revise the correspondence between the matrices and the estimators"

  if (isTRUE(all.equal(matrices, c("A", "B", "AB")))) {
    if (!first == "saltelli" & !first == "jansen" |
        !total == "jansen" & !total == "sobol" & !total == "homma" &
        !total == "janon" & !total == "glen") {
      stop(ms)
    }

  } else if (isTRUE(all.equal(matrices, c("A", "B", "BA")))) {
    if (!first == "sobol"| !total == "saltelli") {
      stop(ms)
    }

  } else if (isTRUE(all.equal(matrices, c("A", "B", "AB", "BA")))) {

    if (!first == "azzini" | !total == "azzini" &
        !total == "jansen" & !total == "sobol" & !total == "homma" &
        !total == "janon" & !total == "glen" & !total == "saltelli") {

      if (!total == "azzini" | !first == "saltelli" & !first == "jansen" &
          !first == "azzini" & !first == "sobol") {

        stop(ms)
      }
    }
  }

  # ------------------------------------

  k <- length(params)

  if (boot == TRUE) {
    m <- d[i, ]
```

```
} else if (boot == FALSE) {
  m <- d
}
if (order == "second") {
  k <- length(params) + length(utils::combn(params, 2, simplify = FALSE))

} else if (order == "third") {
  k <- length(params) +
    length(utils::combn(params, 2, simplify = FALSE)) +
    length(utils::combn(params, 3, simplify = FALSE))

} else if (order == "fourth") {
  k <- length(params) +
    length(utils::combn(params, 2, simplify = FALSE)) +
    length(utils::combn(params, 3, simplify = FALSE)) +
    length(utils::combn(params, 4, simplify = FALSE))
}

# Define vectors based on sample design
# ------------------------------------------------------------------

if (isTRUE(all.equal(matrices, c("A", "B", "AB")))) {
  Y_A <- m[, 1]
  Y_B <- m[, 2]
  Y_AB <- m[, -c(1, 2)]

} else if (isTRUE(all.equal(matrices, c("A", "B", "BA")))) {
  Y_A <- m[, 1]
  Y_B <- m[, 2]
  Y_BA <- m[, -c(1, 2)]

} else if (isTRUE(all.equal(matrices, c("A", "B", "AB", "BA")))) {
  Y_A <- m[, 1]
  Y_B <- m[, 2]
  Y_AB <- m[, 3:(k + 2)]
  Y_BA <- m[, (k + 3):ncol(m)]

} # A warning might be needed here
if (isTRUE(all.equal(matrices, c("A", "B", "AB"))) |
    isTRUE(all.equal(matrices, c("A", "B", "BA")))) {
  f0 <- 1 / (2 * N) * sum(Y_A + Y_B)
  VY <- 1 / (2 * N - 1) * sum((Y_A - f0)^2 + (Y_B - f0)^2)
```

```
}

# Define first-order estimators
# ------------------------------------------------------------------

# Define variance for estimators with A, B, AB; or A, B, BA matrices
if (first == "saltelli" | first == "jansen" | first == "sobol") {
  f0 <- 1 / (2 * N) * sum(Y_A + Y_B)
  VY <- 1 / (2 * N - 1) * sum((Y_A - f0)^2 + (Y_B - f0)^2)
}

# ---------------------------------
if (first == "sobol") {
  Vi <- 1 / N * Rfast::colsums(Y_A * Y_BA - f0^2)

} else if (first == "saltelli") {
  Vi <- 1 / N * Rfast::colsums(Y_B * (Y_AB - Y_A))

} else if (first == "jansen") {
  Vi <- VY - 1 / (2 * N) * Rfast::colsums((Y_B - Y_AB)^2)

} else if (first == "azzini") {
  VY <- Rfast::colsums((Y_A - Y_B)^2 + (Y_BA - Y_AB)^2)
  Vi <- (2 * Rfast::colsums((Y_BA - Y_B) * (Y_A - Y_AB)))

} else {
  stop("first should be sobol, saltelli, jansen or azzini")
}

if (first == "azzini") {
  Si <- Vi[1:length(params)] / VY[1:length(params)]

} else {
  Si <- Vi[1:length(params)] / VY
}

# Define total-order estimators
# ------------------------------------------------------------------

# Define variance for estimators with A, B, AB; or A, B, BA matrices
if (total == "azzini" | total == "jansen" | total == "sobol" |
    total == "homma" | total == "janon" | total == "glen" | total == "saltelli") {
```

```
  f0 <- 1 / (2 * N) * sum(Y_A + Y_B)
  VY <- 1 / (2 * N - 1) * sum((Y_A - f0)^2 + (Y_B - f0)^2)
}

# ---------------------------------
if (total == "jansen") {
  Ti <- (1 / (2 * N) * Rfast::colsums((Y_A - Y_AB)^2)) / VY

} else if (total == "sobol") {
  Ti <- ((1 / N) * Rfast::colsums(Y_A * (Y_A - Y_AB))) / VY

} else if (total == "homma") {
  Ti <- (VY - (1 / N) * Rfast::colsums(Y_A * Y_AB) + f0^2) / VY

} else if (total == "saltelli") {
  Ti <- 1 - ((1 / N * Rfast::colsums(Y_B * Y_BA - f0^2)) / VY)

} else if (total == "janon") {
  Ti <- 1 - (1 / N * Rfast::colsums(Y_A * Y_AB) -
               (1/ N * Rfast::colsums((Y_A + Y_AB) / 2))^2) /
    (1 / N * Rfast::colsums((Y_A ^ 2 + Y_AB^2) / 2) -
       (1/ N * Rfast::colsums((Y_A + Y_AB) / 2))^2)

} else if (total == "glen") {
  Ti <- 1 - (1 / (N - 1) *
               Rfast::colsums(((Y_A - mean(Y_A)) * (Y_AB - Rfast::colmeans(Y_AB))) /
                                sqrt(stats::var(Y_A) * Rfast::colVars(Y_AB))))

} else if (total == "azzini") {
  Ti <- Rfast::colsums((Y_B - Y_BA)^2 + (Y_A - Y_AB)^2) /
    Rfast::colsums((Y_A - Y_B)^2 + (Y_BA - Y_AB)^2)

} else {
  stop("total should be jansen, sobol, homma saltelli, janon, glen or azzini")
}
Ti <- Ti[1:length(params)]

# Define computation of second-order indices
# ------------------------------------------------------------------

if (order == "second" | order == "third" | order == "fourth") {
  com2 <- utils::combn(1:length(params), 2, simplify = FALSE)
  tmp2 <- do.call(rbind, com2)
```

```r
  Vi2 <- Vi[(length(params) + 1):(length(params) + length(com2))] # Second order
  Vi1 <- lapply(com2, function(x) Vi[x])
  final.pairwise <- t(mapply(c, Vi2, Vi1))
  Vij <- unname(apply(final.pairwise, 1, function(x) Reduce("-", x)))

  if (first == "azzini") {
    VY <- Rfast::colsums((Y_A - Y_B)^2 + (Y_BA - Y_AB)^2)
    Sij <- Vij / VY[(length(params) + 1):(length(params) + ncol(utils::combn(1:length(params),

  } else {
    Sij <- Vij / VY
  }

} else {
  Sij <- NULL
}

# Define computation of third-order indices
# ------------------------------------------------------------------

if (order == "third" | order == "fourth") {
  com3 <- utils::combn(1:length(params), 3, simplify = FALSE)
  tmp3 <- do.call(rbind, com3)
  Vi3 <- Vi[(length(params) + length(com2) + 1):(length(params) + length(com2) + length(com3))
  Vi1 <- lapply(com3, function(x) Vi[x])

  # Pairs
  tmp <- do.call(rbind, com2)
  Vij.vec <- as.numeric(paste(tmp[, 1], tmp[, 2], sep = ""))
  names(Vij) <- Vij.vec
  Vi2.1 <- unname(Vij[paste(tmp3[, 1], tmp3[, 2], sep  = "")])
  Vi2.2 <- unname(Vij[paste(tmp3[, 1], tmp3[, 3], sep  = "")])
  Vi2.3 <- unname(Vij[paste(tmp3[, 2], tmp3[, 3], sep  = "")])

  mat3 <- cbind(Vi3, Vi2.1, Vi2.2, Vi2.3, do.call(rbind, Vi1))
  Vijk <- unname(apply(mat3, 1, function(x) Reduce("-", x)))

  if (first == "azzini") {
    Sijl <- Vijk / VY[(length(params) + length(com2) + 1):(length(params) + length(com2) + len

  } else {
    Sijl <- Vijk / VY
  }
```

```
} else {
  Sijl <- NULL
}

# Define computation of fourth-order indices
# ------------------------------------------------------------------

if(order == "fourth") {
  com4 <- utils::combn(1:length(params), 4, simplify = FALSE)
  tmp4 <- do.call(rbind, com4)
  Vi4 <- Vi[(length(params) + length(com2) + length(com3) + 1):
              (length(params) + length(com2) + length(com3) + length(com4))] # Fourth order
  Vi1 <- lapply(com4, function(x) Vi[x])

  # triplets
  tmp <- do.call(rbind, com3)
  Vijk.vec <- as.numeric(paste(tmp[, 1], tmp[, 2], tmp[, 3], sep = ""))
  names(Vijk) <- Vijk.vec
  Vi3.1 <- unname(Vijk[paste(tmp4[, 1], tmp4[, 2], tmp4[, 3], sep  = "")])
  Vi3.2 <- unname(Vijk[paste(tmp4[, 1], tmp4[, 2], tmp4[, 4], sep  = "")])
  Vi3.3 <- unname(Vijk[paste(tmp4[, 1], tmp4[, 3], tmp4[, 4], sep  = "")])
  Vi3.4 <- unname(Vijk[paste(tmp4[, 2], tmp4[, 3], tmp4[, 4], sep  = "")])

  # Pairs
  Vi2.1 <- unname(Vij[paste(tmp4[, 1], tmp4[, 2], sep  = "")])
  Vi2.2 <- unname(Vij[paste(tmp4[, 1], tmp4[, 3], sep  = "")])
  Vi2.3 <- unname(Vij[paste(tmp4[, 1], tmp4[, 4], sep  = "")])
  Vi2.4 <- unname(Vij[paste(tmp4[, 2], tmp4[, 3], sep  = "")])
  Vi2.5 <- unname(Vij[paste(tmp4[, 2], tmp4[, 4], sep  = "")])
  Vi2.6 <- unname(Vij[paste(tmp4[, 3], tmp4[, 4], sep  = "")])

  mat4 <- cbind(Vi4, Vi3.1, Vi3.2, Vi3.3,
                Vi2.1, Vi2.2, Vi2.3, Vi2.4, Vi2.5,
                Vi2.6, do.call(rbind, Vi1))

  Vijlm <- unname(apply(mat4, 1, function(x) Reduce("-", x)))

  if (first == "azzini") {
    Sijlm <- Vijlm / VY[(length(params) + length(com2) + length(com3) + 1):
                          (length(params) + length(com2) + length(com3) + length(com4))]

  } else {
```

```
      Sijlm <- Vijlm / VY
    }

  } else {
    Sijlm <- NULL
  }
  return(c(Si, Ti, Sij, Sijl, Sijlm))
}

#generate boostrap samples of Sobol' index for each parameter
d <- 4 #dimension of input
N <- 1000 #sample number of xi
params <- paste("$X_", 1:d, "$", sep = "")
mat <- sobol_matrices(N = N, params = params)
R <- 200 #bootstrap sample number of sobol
SampleNum1 <- 0
ParamSampleSelectedOnce <- matrix(NA, nrow = dim(ParamSample)[1],
                                  ncol = dim(ParamSample)[2])
TotalSobolX4PriorSample <- matrix(NA, nrow = dim(ParamSample)[1],
                                  ncol = 1)
TotalSobolX4PriorSample1 <- matrix(NA, nrow = dim(ParamSample)[1],
                                  ncol = 1)

#select the parameter samples that satisfy the condition of prior of Sobol'
for(i in 1:dim(ParamSample)[1]){
  Y <- PolyFunc(ParamSample[i,], mat) #generate sobol matrices for estimate sobol

  #generate bootstrap samples of sobol indices
  SobolBootstrap <- boot(data = matrix(Y, nrow = N), statistic = sobol_boot, R = R,
              N = N, params = params, first = "saltelli", total = "jansen" ,
              order = "first", matrices = c("A", "B", "AB"), boot = TRUE)

  #extract bootstrap samples of total sobol for X4
  #since there are 4 Xi, the 8-th column is samples for total of X4
  TotalSobolX4GenerateSample <- SobolBootstrap$t[, 8]

  #calculate summary statistic
  SummaryStat1 <- sum(log(TotalSobolX4GenerateSample))
  SummaryStat2 <- sum(log(1 - TotalSobolX4GenerateSample))

  #select by condition
  StatDist <- sqrt((SummaryStat1 - SufficientStat1)^2 + (SummaryStat2 - SufficientStat2)^2)
  if((abs(SummaryStat1 - SufficientStat1)<15) &
```

```
     (abs(SummaryStat2 - SufficientStat2)<15) &
     (is.na(StatDist) == FALSE)){
    SampleNum1 <- SampleNum1 + 1
    ParamSampleSelectedOnce[i, ] <- ParamSample[i,]
    TotalSobolX4PriorSample1[i] <- mean(TotalSobolX4GenerateSample)
  }
  TotalSobolX4PriorSample[i] <- mean(TotalSobolX4GenerateSample)
}

#process the samples
ParamSampleSelectedOnce <- data.frame(na.omit(ParamSampleSelectedOnce))
colnames(ParamSampleSelectedOnce) <- c("a1", "a2", "a3")
TotalSobolX4PriorSample1 <- na.omit(TotalSobolX4PriorSample1) %>% unlist() %>% unname()

#draw the plot of density of model parameters
p1 <- ggplot() +
  geom_density(aes(x = ParamSample[, 1], colour = "a1 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a1, colour = "a1 density after selected"),
               fill = "yellow", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a1 density before selected" = "purple",
               "a1 density after selected" = "yellow")) +
  xlab("a1") +
  ylab("density p")

p2 <- ggplot() +
  geom_density(aes(x = ParamSample[, 2], colour = "a2 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a2, colour = "a2 density after selected"),
               fill = "yellow", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a2 density before selected" = "purple",
               "a2 density after selected" = "yellow")) +
  xlab("a2") +
  ylab("density p")

p3 <- ggplot() +
  geom_density(aes(x = ParamSample[, 3], colour = "a3 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a3, colour = "a3 density after selected"),
```

```
                fill = "yellow", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a3 density before selected" = "purple",
               "a3 density after selected" = "yellow")) +
  xlab("a3") +
  ylab("density p")
print(p1 / p2 / p3)

#draw the density plot of Sobol' indices
ggplot() +
  geom_density(aes(x = as.numeric(TotalSobolX4PriorSample),
                   colour = "Sobol' index density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = TotalSobolX4PriorSample1,
                   colour = "Sobol' index density after selected"),
               fill = "yellow", alpha = 0.1) +
  geom_density(aes(x = rbeta(2000, 300, 700),
                   colour = "prior of Sobol' index"),
               fill = "red", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("Sobol' index density before selected" = "purple",
               "Sobol' index density after selected" = "yellow",
               "prior of Sobol' index" = "red")) +
  xlab("Sobol' index of X4") +
  ylab("density p")
```

8.Code for section 5.4.3

```
#density plot of posterior and prior of model parameters
p1 <- ggplot() +
  geom_density(aes(x = ParamSample[, 1], colour = "a1 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a1, colour = "a1 density after selected"),
               fill = "yellow", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedTwice$a1, colour = "posterior of a1"),
               fill= "red", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a1 density before selected" = "purple",
               "a1 density after selected" = "yellow",
               "posterior of a1" = "red")) +
```

```
  xlab("a1") +
  ylab("density p")

p2 <- ggplot() +
  geom_density(aes(x = ParamSample[, 2], colour = "a2 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a2, colour = "a2 density after selected"),
               fill = "yellow", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedTwice$a2, colour = "posterior of a2"),
               fill= "red", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a2 density before selected" = "purple",
               "a2 density after selected" = "yellow",
               "posterior of a2" = "red")) +
  xlab("a2") +
  ylab("density p")

p3 <- ggplot() +
  geom_density(aes(x = ParamSample[, 3], colour = "a3 density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedOnce$a3, colour = "a3 density after selected"),
               fill = "yellow", alpha = 0.1) +
  geom_density(aes(x = ParamSampleSelectedTwice$a3, colour = "posterior of a3"),
               fill= "red", alpha = 0.1) +
  scale_colour_manual(
    name = "Density Plot",
    values = c("a3 density before selected" = "purple",
               "a3 density after selected" = "yellow",
               "posterior of a3" = "red")) +
  xlab("a3") +
  ylab("density p")
print(p1 / p2 / p3)

#density plot of posterior and prior of Sobol' indices
ggplot()+
  geom_density(aes(x = as.numeric(TotalSobolX4PriorSample),
                   colour = "Sobol' index density before selected"),
               fill = "purple", alpha = 0.1) +
  geom_density(aes(x = TotalSobolX4PriorSample1,
                   colour = "Sobol' index density after selected"),
               fill = "yellow", alpha = 0.1) +
  geom_density(aes(x = rbeta(2000, 300, 700),
```

```
                colour = "prior of Sobol' index"),
            fill = "red", alpha = 0.1) +
geom_density(aes(x = TotalSobolX4PosteriorSample,
                colour = "posterior of Sobol' index"),
            fill= "green", alpha = 0.1)+
scale_colour_manual(
  name = "Density Plot",
  values = c("Sobol' index density before selected" = "purple",
            "Sobol' index density after selected" = "yellow",
            "prior of Sobol' index" = "red",
            "posterior of Sobol' index" = "green")) +
geom_vline(xintercept = 0.282)+
xlim(0.2, 0.4)+
xlab("Sobol' index of X4")
```