University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: **Computer Science**	Spring semester, 2015 Open / Restricted access
Writer: **Jiaqi Ye**	………………………………………… (Writer's signature)
Faculty supervisor: **Dr. Rui Máximo Esteves, Prof. Chunming Rong**	
Title of thesis: **Using Machine Learning for Exploratory Data Analysis and Predictive Modeling**	
Credits (ECTS): **30**	
Key words: Machine learning; Exploratory Data Analysis; Predictive Modeling	Pages: …71………… + enclosure: …76……… Stavanger, ……………….. Date/year: 05/07/2015

Acknowledges

The author would like to express her special gratitude to:

Prof. Rui Maximo Esteves from University of Stavanger;

Prof. Dr. Chunming Rong from University of Stavanger;

ChEng. Flaviu Borlea from University of Stavanger;

Abstract

Exploratory data analysis and predictive analytics can be used to extract hidden patterns from data and are becoming increasingly important tools to transform data into information. Machine learning has become a powerful technique for predictive analytics, it can directly predict the dependent variable without focusing on the complex underlying relationships between predictors.

Oil and gas industries has found these techniques very useful in their business such as oil well production prediction and equipment failure forecasting. Our work intends to build a predictive model based on data which can produce precise predictions and is efficient in practice.

With this work we follow a methodology to build predictive models based on real data. The experiments focus on three machine learning algorithms, which are linear regression, neural network and k-nearest neighbors. Within each category, experiments are carried out on multiple model variants in order to achieve a better performance. The built models have been tested on new data and cross-validation has been performed to validate the models. The predictive performance of each model is evaluated through R-squared and root-mean-squared error (RMSE) parameters and comparison of predicted values and actual values. Experiment results shows that nearest neighbor with k-dimensional tree is the most efficient model with best predictive performance in this case. This model can be a possible solution to help the expert in making prediction relying on the data.

Contents

Index of Figures

Index of Tables

Chapter I – Introduction

The Chapter intends to give a short overview of this work. It starts with the description of the background of the thesis. The basic concepts of the key focus are given. The chapter finishes with a general outline of the work.

Thesis Background

In oil and gas industry, the use of predictive data mining dates back to the early 1990s [1]. Since then, hundreds of technical articles have been contributed to the application of artificial intelligence and data mining in the exploration and production industry. Predictive analytics has found a wide range usage in oil and gas industries, such as oil well production prediction, equipment failure forecasting, etc. One can image that a single equipment failure can cost millions of dollars in damage while unplanned downtime and repair is also something that the oil and gas professionals would try their best to avoid [2]. Through technologies like exploratory data analysis and predictive modelling on the assembled data, one may able to stop problems before they occur.

A manufacturer of mechanical equipment for the Oil & Gas sector, which opted to remain anonymous, wants to find a model which can modelling their data and produce accurate prediction. The given dataset is a typical dataset in their business. However, due to the sensitive information contained in the dataset, the physical meanings of the variables in the dataset are kept unknown by the experimenters. Thus, the experiments are carried out relying only on the data, no prior knowledge can be applied in the analysis.

In this context, this paper studies using machine learning algorithms for exploratory data analysis and predictive modelling on the given data, the key concepts of this topic are explained as following.

Exploratory data analysis (EDA) which was promoted by John Tukey [3] analyzes the data sets to summarize their main characteristics, often with data visualization methods. Applying EDA prior to modelling the data can help checking assumptions required for model fitting and hypothesis testing, and performing necessary data preprocessing, such as handling missing values and data transformation [4]. While predictive modelling can leverage statistics to predict future behavior. A predictive model is consist of a number of predictors and may also contains the response variable (supervised learning). Predictors are the variables which are relevant to the future behavior or results. Once the data is collected, a statistical model can be built to learn the data, predictions are made and the model can be validated by testing the model on new data.

Machine learning algorithms are often applied in predictive models to learn the main pattern from a training dataset in order to make predictions. Machine learning tasks can be classified into three categories, which are supervised learning, unsupervised learning and reinforcement learning [5, 6]. The main focus of our work falls on the study of supervised learning algorithms. In supervise learning, each sample contains an input object and a desired output value, and the algorithm analyzes the training data and produces an inferred function which can be used for mapping new samples [7]. Machine learning algorithms can also be categorized into classification and regression when one considers the desired output of the system [8]. In classification, data (inputs or outputs) is divided into two or more classes, and the algorithm produces a model that assigns inputs to one or more of these classes. In regression, the response variables are continuous rather than discrete. In our case, the desired output is a continuous variable, thus supervised regression learning algorithms are studied to achieve our goal.

Thesis Outline

Our work begins with a study about data preprocessing techniques, which is a key component of exploratory data analysis. The aim was to explore the data and capture the main characteristics, and preprocess the data to remove invalid samples and irrelevant predictors, in order to have valid inputs for the models.

The work follows by a revision about supervised regression learning models. The regression models are classified into two categories: linear regression models and nonlinear regression models. The advantages and limitations of each model are presented. The aim was to find a candidate set of learning models which suit our case. And we stated that linear regression, neural network, and k-nearest neighbor models could be promising methods to solve our problem. We then formulate the aim of our work as following hypothesis:

H: *Given the data, we may use linear regression, or neural network, or K-nearest neighbors for data prediction.*

Experiments to test our hypothesis were conducted following a certain methodology. Data preprocessing methodology was performed as needed. The construction of three models are explained and the performance of each model is presented as well as data visualization as needed. Comparisons of the performance of three models are made and shows that k-nearest neighbor model fits the data best.

Chapter II – Theory

This chapter elaborates the theoretical foundation of our work. The chapter follows the logical process of data exploratory analysis and predictive modelling. It starts with an introduction of various data preprocessing methods. And then the chapter continuous with a review of multiple regression models, and explains the idea of model selection. The chapter ends with a description of over-fitting problems and the approaches to deal with it. The research aims at building a scientific methodology for data analysis.

Data preprocessing

Real-world data is generally incomplete and noisy, and is likely to contain irrelevant and redundant information or errors [9]. Data preprocessing, which is an important step in data mining processes, helps transform the raw data to an understandable format [10]. Besides, some modeling techniques are quite sensitive to the predictors, such as linear regression. Thus, examining and preprocessing data before entering the model is essential. This chapter outlines some important methods in data preprocessing, including data cleaning, data transformation and data reduction.

Data cleaning

Dealing with missing data

Missing data is common in real world dataset, and it has a profound effect on the final analysis result which may make the conclusion unreliable. There are different types of missing data. We should have a good understanding of why the data is missing. If data is missing at random or if the missingness is related to a particular predictor but the predictor has no relationship with the outcome, then the data sample can still represent the population [11]. However, if the data is missing in a pattern that is related to the response, it can lead to a significant bias in the model, making the analysis result unreliable.

Many techniques have been proposed to deal with missing data [12, 13, 14], and generally they can be divided into two strategies. The first and the simplest one would be removing the missing data directly. If the missing data is distributed at random or the missingness is related to a predictor that has zero correlation with the response, and the dataset is large enough, then the removal of missing data has little effect on the performance of analysis. However, if one of the above conditions is not satisfied, then simply removing the missing data is inappropriate.

The second strategy is to fill in or impute the missing data based on the rest of the data. Generally, there are two approaches. One method simply uses the average of the predictor to fill in the missing value. Alternatively, we can use a learning algorithm such as Bayes or decision tree to predict the missing value [15]. It is worth noting that additional uncertainty is added by the imputation.

Dealing with outliers

Outlier is defined as an observation point that is distant from the mainstream data. The presence of outliers can break a model's analysis ability. For example, outliers have a strong impact on data scaling and the regression fits. However, it is hard to identify outliers if the data range is not specified. One of the most efficient ways to identify outliers may be data visualization. By looking at a figure, we can point out some suspected observations and check whether these values are scientifically valid (e.g. positive weight). Removal of outliers can only be taken when there are truly valid reasons.

Instead of removing the outliers, an alternative way is to transform data to minimize the effect caused by outliers. Spatial sign proposed by Serneels in 2006 transform the predictor values onto a new sphere [16]. As figure 1.1 shows, it minimizes the effect of outliers by making all the observations the same distance from the center of the sphere [9].

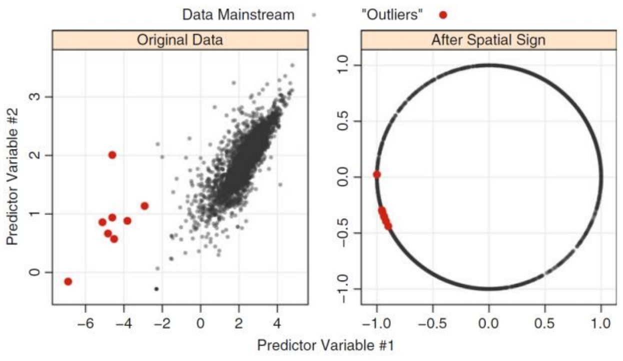


Fig.1 [9]: An example of dealing with outliers by spatial sign.

Data Transformation

Centering and Scaling

Data preprocessing involves transforming data into a suitable form for analysis [17]. The most basic and straightforward data transformation techniques are data centering and scaling. Many data mining techniques, such as Euclidean distance, require data to be centered and scaled before entering the model. These transformations help improve the interpretability of parameter estimates when there is interaction in the model [18].

Data centering subtracts the mean value of the predictors from the data, causing that each predictor has a zero mean. And to scale the data, each value of the predictor is classified by its standard deviation, thus the scaled data has unit deviation. It is worth noting that the presence of outliers has a strong effect on data scaling, identifying and disposing outliers is necessary before data scaling.

Transformations to resolve skewness

Data can be positively or negatively skewed, while many statistical models make assumptions that the data to be examined is normal distributed. Applying transformation to resolve skewness helps to improve the data normality [19].

There are quite a lot of transformation methodologies that helps to fix the skew, such as replacing the data with log, square root, or inverse transformations [9]. Fig.2 shows an example of how a log transformation can fix a positive skew.

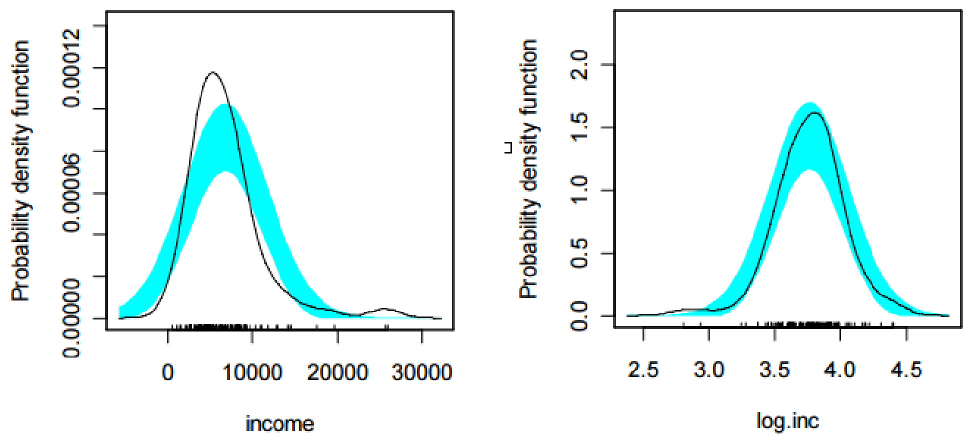


Fig.2 [1]: An example shows transformation can fix skewness. Left: a right skewness data distribution. Right: the same data after a log transformation.

Box and Cox (1964) propose a set of transformations indexed by a parameter λ that can empirically identify an appropriate transformation.

$$x^* = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0 \end{cases} \tag{1}$$

Equation (1) can identify various transformations, such as log transformation, square root (λ=0.5), square (λ=2), inverse (λ=-1), and other in-between transformations.

Data reduction and Feature Extraction

Zero- and Near Zero-Variance Predictors

In real world dataset, it is common that some predictors only have a unique value that is so called zero-variance predictor. It is of very little use in predicting variables, and may cause the model (e.g. linear regression) to crash or fit to be unstable [9]. Thus these uninformative predictors can be discarded.

Similarly, the near zero-variance predictors which have only handful of unique values that have very low frequencies occurrence may also need to be identified and deleted before entering the model. These near zero-variance predictors can be identified through calculating two metrics. One is called frequency ratio, which is the frequency of the most prevalent value over the second most frequent value. The second one is percentage of the unique values of the specified predictors. If the frequency ratio of the predictor is larger than a predefined threshold, and the percentage of unique values is low, then this predictor can be considered as a near zero-variance predictor.

Identifying Highly Correlated Predictors

Real-world data often encounters a common situation where a pair of predictor variables are highly correlated, this is also called collinearity [20]. It is essential to pre-process data to avoid highly correlated predictors pair in the data for the following reasons. First, two highly correlated predictors are likely to contain the same ultimate information and more predictors mean more complexity is added to the model. For the models where training the predictors is costly (such as Neural Network and K-Nearest Neighbors etc.), it is obviously that fewer predictors is preferred. Besides, for some technologies like linear regression, applying highly correlated predictors pair in the model can result in very unstable model, causing numeric errors and worse predictive performance.

An effective approach to dealing with correlated predictors is to remove a minimum number predictors which has top highest pairwise correlations and make sure all pairwise correlations are below a certain level. The basic idea is to remove the highly correlated predictors iteratively as the algorithm shows below [9]:

1. Calculate the predictors’ correlation matrix.

2. Determine the predictor pair (A and B) which has the largest absolute pairwise correlation.

3. Calculate the average correlation between predictor A and the other predictors, same for B.

4. Remove the one (A or B) which has a larger average correlation with other predictors.

5. Repeat Steps 2 – 4 until all absolute correlations are below the threshold.

Principal Components Analysis (PCA)

The previous section discusses about removing the highly correlated predictors to achieve data reduction, while there is another data reduction technique called PCA, which seeks to find linear combination of the predictors that capture the most possible variance. The idea of PCA is that: the first PC is defined as the linear combination of the predictors that captures the most variability of all possible linear combinations [9]. Then, subsequent PCs are derived such that these linear combinations capture the most remaining variability while also being uncorrelated with all previous PCs. Mathematically, the jth PC can be written as:

$$PC_j = (a_{j1} \times \text{Predictor 1}) + (a_{j2} \times \text{Predictor 2}) + \cdots + (a_{jP} \times \text{Predictor P}). \qquad (2)$$

P is the number of predictors. The coefficients $\mathrm{a}_{j1}, \mathrm{a}_{j2}$, ..., a_{jP} are called component weights and help us understand which predictors are most important to each PC.

The primary advantage of PCA lies in that it creates components that are uncorrelated [21]. As is mentioned earlier in this chapter, some predictive models prefer predictors to be uncorrelated (or at least low correlation) in order to find solutions and to improve the model’s numerical stability. PCA preprocessing creates new predictors with desirable characteristics for these kinds of models.

However PCA is blind to the response, it is an unsupervised technique. If the predictive relationship between the predictors and response is not connected to the predictors’ variability, then the derived PCs will not provide a suitable relationship with the response. Thus, additional attention should be paid when applying PCA.

Regression Models

Introduction

Regression analysis is a statistical technique for investigating the dependence of a response variable on one or multiple predictors, including prediction of future values of a response, discovering which predictors are more important, and estimating the impact of changing a predictor or a treatment on the value of the response [9]. In this chapter, several regression analysis models such as linear regression model, nonlinear regression model and nonparametric regression model are introduced. Strength and weakness of these methods are also discussed in this section.

Linear Regression

Introduction

Linear regression analysis is one of the most basic regression analysis approaches, actually, it is the foundation of many other modern regression modeling methodologies [22]. Thus, a good understanding of linear regression modeling is necessary to understand other regression modeling methods, such as neural networks or support vector machine.

In general, given a data set $\{y_i, x_{i1}, ..., x_{ik}\}_{i=1}^{n}$ of n statistical units, a linear regression can be written in the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \varepsilon_i \tag{3}$$

It assumes that the relationship between the response y_i of ith sample and the k-vector of predictors x_i is linear, in other words, we says that it is linear in the parameters. The parameter β_j , $j = 0, 1, ..., k$, are called the regression coefficients, and ε_i is the random error or noise. This model describes a hyper-plane in the k-dimensional space of the predictors x_i . If $i = 1$, then Equation (3) involves only one predictor variable, and it is called a simple linear regression. If $i > 1$, Equation (3) involves more than one predictor variables, and it is called multiple linear regression [23].

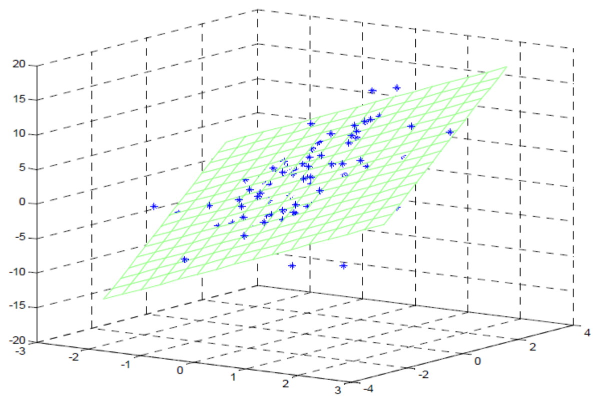


Fig.3 [23]: An example of three-dimensional plot of regression model

There is no obvious answer that says which linear regression model is the best for a training set, however we can estimate the alternative models by estimating the model parameters. To estimate the model parameters, there are many different methods, but the objectives of these methods are the same, that is to minimize the sum of the squared errors.

Ordinary Least Squares Estimation

Ordinary least squares is the most basic and common estimator [22]. It picks parameters $\hat{\beta}$ to minimize the residual sum of squares (RSS), that is, it aims at minimizing the differences between the true responses y_i in the dataset and the predicted value $\hat{y}_i$ from linear regression model. It can be written as the following form.

$$RSS(\beta) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = (y - X\beta)^T (y - X\beta) \tag{4}$$

Where X is the matrix of predictor variables and $\beta = [\beta_0, \beta_1, ..., \beta_k]$ is the matrix of parameters. To minimize RSS, we should have:

$$\frac{\partial RSS(\beta)}{\partial \beta} = 0 \tag{5}$$

Thus, we can get the OLS estimator of β :

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{6}$$

Regression coefficients vector is directly interpreted by Equation (4), and it can be calculated only under the condition of $n > k$ observations are available. Equation (5) can be computed easily and very efficiently by computer. It can be quickly applied even to problems with hundreds of features and tens of thousands of observations [24]. These properties makes OLS linear regression very popular as a modeling tool.

However, there is a list of problems with using OLS linear regression in practice. The first problem would be outliers. OLS regression can perform very badly when there is outliers exists in the training dataset. Since OLS regression concerns about the sum of the squared residuals, any observations that differs a lot from the rest of the data will have an extremely large squared errors, thus it has a significant effects on the sum of the squared residuals.

OLS regression model makes the assumption that each predictors are uncorrelated, thus problem will arise if the variables fed to it are strongly correlated to each other. Under this circumstance, OLS regression method may lead to poor predictions. This also can be explained from Equation (6), if the linearly independent property of predictors X is not fulfilled, then matrix $(X^T X)^{-1}$ may not exist.

Another problem of OLS regression and other linear regression methods is that they do not fit nonlinear relationship between predictors and responses. This would be discussed later in this chapter.

Partial Least Square Analysis (PLS)

Like PCA, partial least square (PLS) methodology finds linear combination of predictors. However, as described previously, PCA does not consider the response when choosing components, and it has difficulty making accurate predictions when the variability of the predictors and the response is not correlated. In contrast, partial least square methodology is considered as a supervised dimension reduction procedure, the PLS linear combinations of predictors are chosen to maximally summarize the covariance with the response [23].

As is illustrated in Fig.4 [9], the PLS finds components that maximally summarize the variability of predictors while being optimally correlated with the outcome at the same time.

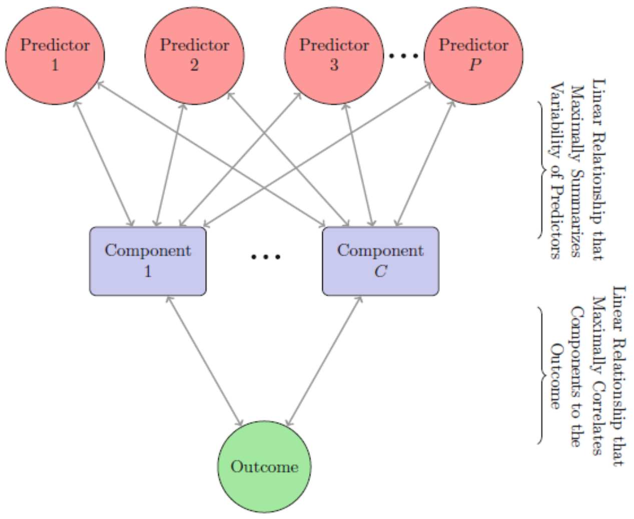

Predictor 1
Predictor 2
Predictor 3
Predictor P
Component 1
Component C
Outcome
Linear Relationship that Maximally Summarizes Variability of Predictors
Linear Relationship that Maximally Correlates Components to the Outcome

Fig.4 [9] A diagram displaying the structure of a PLS model

Fig.5 [9] illustrates the differences between PCA and PLS by applying them on the same dataset. As the left-hand panel shows, the first PLS component direction is nearly perpendicular to the first PCA direction. And the right-hand panel illustrates that the PLS direction contains highly predictive information for the outcome, which can hardly be achieved by PCA in this case.

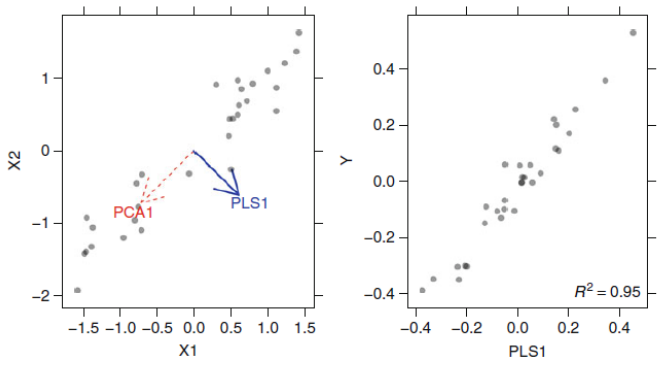


Fig.5 [9] Comparison of PCA and PLS approaches when applying on the same dataset.

Penalized Regression Models

Mean squared error (MSE) is a combination of variance and bias, and the Least-squares estimators are said to be the best linear unbiased estimators, in which “best” signifies minimum variance [25]. However, it is possible to find a model with lower MSE than an unbiased model, and this is called the “bias-variance trade off”, as illustrated in Fig. 6.

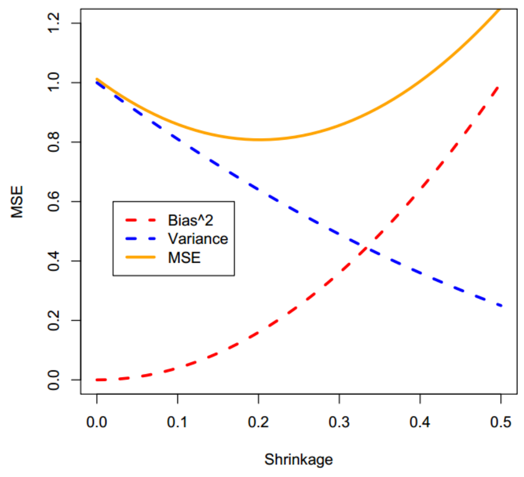


Fig.6 [25] Bias-Variance tradeoff

It solves the multicollinearity issue faced by least-squares linear regression and improves the prediction performance. Penalized regression add a penalty to the residual sum of squared (RSS) to build such a biased linear regression model.

Specifically, the ridge regression add a second-order penalty (“L2”) to the parameter estimates, as Equation (7) shows. And the ridge regression estimate $\hat{\beta}_R$ is given by Equation (8).

$$RSS_{L2}(\beta) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{7}$$

$$\hat{\beta}_R = (X^T X + \lambda I)^{-1} X^T y \tag{8}$$

According to Equation (8), we can infer that:

$$\lambda \to 0 : \hat{\beta}_{Ridge} \to \hat{\beta}_{OLS} \, .$$

$$\lambda \to \infty : \hat{\beta}_{Ridge} \to 0 . \tag{9}$$

This illustrates the shrinkage feature of ridge regression. By applying the ridge regression penalty, the estimates can be shrink toward zero, in this way the variance of the estimate is reduced [25]. However, as ridge regression can never sets the coefficients to zero exactly, and therefore cannot perform variable selection, this property does not suit for the situation where some of the parameter estimates become negligibly small.

Another popular penalized regression model is the least absolute shrinkage and selection operator model (lasso). The lasso estimate $\hat{\beta}_L$ is defined as the argument that minimize RSS:

$$RSS_{L1}(\beta) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{p} \left|\beta_j\right| \tag{10}$$

The only difference between the lasso regression and ridge regression is that the lasso uses a ℓ_1 penalty while ridge regression uses a (squared) ℓ_2 penalty. The problem may look similar, however, their solutions behave very differently. The tuning parameter λ controls the strength of the penalty, and Lasso has the same shrinkage property as we discussed in ridge regression Equation (8). But unlike ridge regression, due to the nature of the ℓ_1 penalty, some parameters are actually shrunken to zero for some value of λ between 0 to infinity. This makes lasso to be able to perform variable selection in the linear model. With the increase of λ , less variables are selected, and among the nonzero parameters, more shrinkage is employed [25].

The shrinkage and variable selection properties make lasso a very attractive model. However it is worth noting that if there are high correlations between variables, lasso tends to select only one of them. And there is a severe restriction in lasso model, if there are more variables than observations ($\mathrm{n} < p$), lasso can only select at most n out of the p variables, this is of course not what we expect.

Nonlinear Regression

Introduction

Linear regression models offer a rich and flexible framework that meets many analysts' needs. However, it doesn't mean that linear regression models are suitable under all circumstances. Many problems lie in both engineering and the sciences where the response variable and the predictor variables are related through a known nonlinear function. A nonlinear regression model emerged as required [Introduction].

Many of linear regression models can be adapted to nonlinear models by manually adding model terms, such as squared terms [26]. Nevertheless, this requires prior knowing the specific nature of the nonlinearity in the data.

Neural network (NN), support vector machines (SVMs), and K-nearest neighbors (KNNs) are discussed in this section due to their popularity in application. These regression models are inherently nonlinear in nature, and they does not require knowing specified nonlinearity data trend prior to model training.

Neural Network

Neural network (NN) proposed by Bishop et al. (1995) is a powerful nonlinear regression technique inspired by the way biological nervous system (i.e. brain, process information) [27]. Similar to partial least squares, neural network is composed of a number of interconnected processing elements (called neurons or hidden units).

As is illustrated in Fig.7, the hidden units are linear combinations of the predictor variables. However, the linear combination is usually transformed by a nonlinear function such as a sigmoidal function $\mathrm{g}(\square)$:

$$h_k(x) = g\left(\beta_{0k} + \sum_{i=1}^{P} x_j \beta_{jk}\right) \tag{11}$$

Where $\mathrm{g}(u) = \frac{1}{1+e^{-u}}$

Once the hidden units are determined, another linear combination is applied in connecting the hidden units to the outcome.

$$\mathrm{f}\left(x\right)=\gamma_0+\sum_{k=1}^{H}\gamma_k h_k \tag{12}$$

It is not hard to calculate that, for this kind of NN model with N predictors, the total number of parameters estimated is $H(N+1)+H+1$. It is obvious that with the increase of N, the number of parameters will become quite large, thus pre-processing data and removing irrelevant predictors is an essential step to reduce computation time.

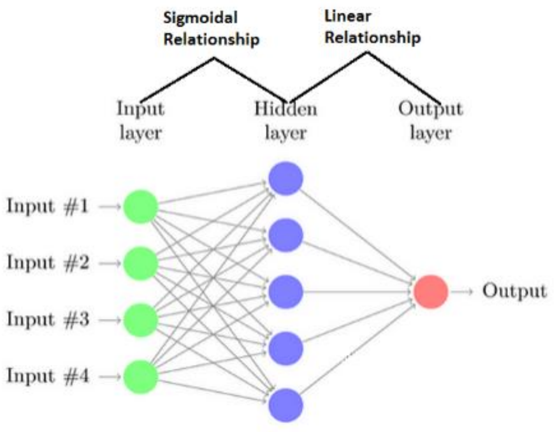


Fig.7 A diagram of a feed forward neural network with a single hidden layer.

In neural network, the parameters are initialized to random values and then specific learning algorithms such as Bayesian or gradient descent algorithms are applied to minimize the sum of the squared residuals [28]. However, neural network model cannot guarantee a global solution but a local optimal solution. A better approach is creating several NN models with different initial values and averaging the results of these models to get a more reliable prediction.

As has been discussed above, NN uses gradients to tune the model parameter, thus highly correlated predictors often have a negative impact on NN model. Removal of highly correlated predictors or pre-processing data using PLS like techniques would both help get a stable model and improve computation time.

Over-fitting is also a negligible issue in neural network due to the large number of regression coefficients in NN. To avoid over-fitting issue, one can apply a penalization method similar to ridge regression discussed in linear regression section. A penalty λ (often called weight decay) is added to the regression coefficients so that any large values can be penalized and the number of free parameters is limited. λ is usually between 0 and 0.1.

Multilayer Perceptron

Multilayer perceptron is a common structure type of neural network, it is a feedforward network which has two or more hidden layers [29]. Fig.8 depicts a diagram of multilayer perceptron network with 2 hidden layers. As is shown in Fig.8, there is no connection between neurons in the same layer, and connections usually begin in a hidden unit on a layer and end to a hidden unit on the next layer. Different from single layer network, multilayer perceptron can create internal representations and extract different features in each layer [30].

Multilayer perceptron is considered as one of the preferred techniques for gesture recognition.

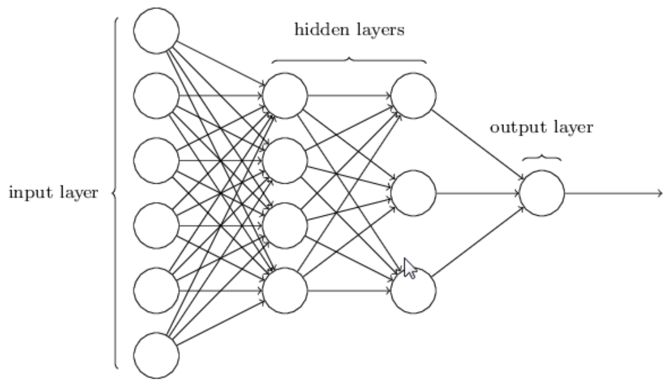


Fig.8 [30] A diagram of multilayer perceptron with 2 hidden layers

Backpropagation algorithm

Previously, we discussed feed forward neural network in which the information always move one direction. In this section, we look at backpropagation algorithm proposed by Rumelhart, Hinton and Williams [31]. Neural network with backpropagation algorithm is considered as a supervised learning algorithm, since it requires the actual outcome of each observation to calculate the loss function gradient.

Fig.9 illustrates how the backpropagation neural network works, the propagation of information in the backpropagation neural network generally involves two steps. The first step is a forward propagation to training the input through the network and generate the output activations. The second step is a backward propagation of the error (delta) of prediction and actual response (prediction-response) to update the weights of the network.

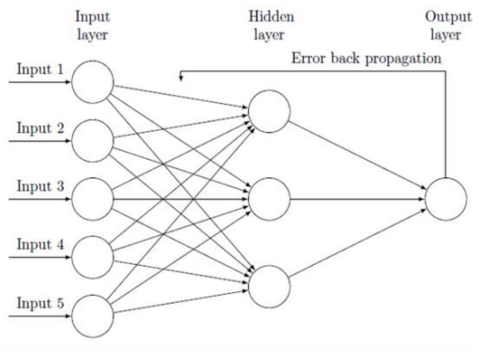


Fig.9 [31] A diagram of back propagation neural network

Deep Belief Neural Network with Restricted Boltzmann Machine

Restricted Boltzmann machine (RBM) is a generative stochastic neural network that can train input data based on a probabilistic model. RBM was first proposed under the name Harmonium by Paul Smolensky (1986) [32], but it becomes popular after Geoffrey Hinton et al. invented the fast learning algorithms for them, and now it is widely used in feature reduction, classification, collaborative filtering and so on [37,38].

Fig.10 shows a diagram of a RBM with 10 visible variables and 3 hidden units. As we can see from Fig.10, the RBM graphical model is a fully-connected bipartite graph in which may have a symmetric connection between the visible and hidden units and no connection between neurons of the same layer. The value of nodes depends on the value of other nodes that they are connected to. Given the state of the visible variables, hidden units are independent from each other, vice versa. Thus, the conditional distributions $\mathrm{p}(h|v)$ and $\mathrm{p}(v|h)$ factorize nicely as Equation (13).

$$\mathrm{p}(h|v) = \prod_{i=1}^{n} p(h_i|v) \quad \textbf{and} \quad \mathrm{p}(v|h) = \prod_{j=1}^{m} p(v_j|h) \tag{13}$$

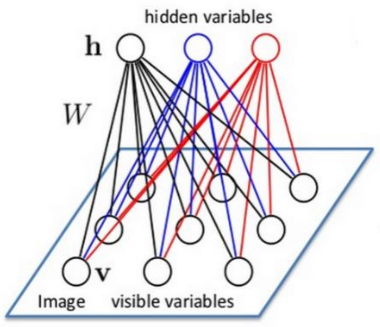


Fig.10 [36]: A diagram of a restricted Boltzmann machine.

Deep belief network (DBN), which is a multi-layer learning architectures, can be built with stacked RBMs as building blocks. This idea is proposed by Geoffrey Hinton et.al in [33], which have received a lot of attention in many areas. Fig.11 shows the training process of deep belief network with stacked RBMs as building blocks.

The basic idea of using stacked RBMs to train as deep belief networks is that the hidden units of a trained RBM extract relevant features of the observations which can be served as input for another RBM [35]. As Fig.11 shows, the first step of DBN is training the visible layer that models the input x, obtaining a representation of the input. And then the extracted feature of input can be served as input for the second layer. Train the second layer as a RBM, iterate these steps until a desired number of layers is obtained. Equation (14) presents the theoretical justification of the learning procedure, the probability of generating a visible vector v with l hidden layers can be expressed as:

$$P(\mathrm{v}, h^1, h^2, \ldots, h^l) = P(\mathrm{v}|h^1) P(h^1|h^2) \ldots P(h^{l-2}|h^{l-1}) P(h^{l-1}, h^l) \tag{14}$$

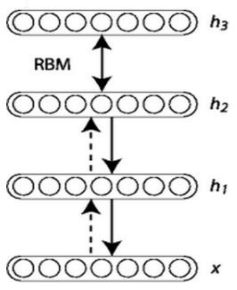


Fig.11 [34] A diagram of training process of DBN with RBMs.

An important property of DBN is that it is representationally efficient, for a same function, it needs fewer computational units than a single layer neural network. Besides, DBN with stacked RBMs can be viewed as an unsupervised pre-training of the feed-forward neural network [36], and it has been argued that it helps to overcome some problems that have been observed in multilayer perceptron. DBN can also combines some supervised fine-tuning algorithms, such as backpropagation algorithm, to better model the tuning the model.

Support Vector Machine

Supposing vector machines (SVMs) are a class of powerful supervised learning models that can be both used for classification and regression analysis [39]. Here, we just discuss how to use SVMs to build a robust regression model, in the hope of minimizing the effect of outliers on the regression equations.

In linear regression section, we have discussed one of drawbacks of ordinary linear regression is it would be easily influenced by outliers, because it seeks to find parameter estimates that minimize RSS (Equation (2)) which represents the sum of the squared residuals. One alternative to reduce the sensitivity of equation against outliers is using Huber function. As illustrated in Fig.12 left panel, this function utilizes squared residuals when they are “small” and uses absolute residuals when they are “large” [40].

In SVMs regression, it uses a function similar to but with an important difference from Huber function [9]. As shown in Fig.12 right panel, given a user defined threshold ε observations with residuals smaller than the threshold do not contribute to the regression fit while observations which has an absolute difference greater than the threshold contribute a linear-scale amount. Equation (14) presents the ε loss function with a penalty Cost defined by user which penalizes large residuals, and $L_{\varepsilon}(\cdot)$ is the ε insensitive function.

$$\mathrm{Cost}\sum_{i=1}^{n} L_{\varepsilon}\left(y_i - \hat{y}_i\right) + \sum_{j=1}^{P} \beta_j^{\,2} \tag{14}$$

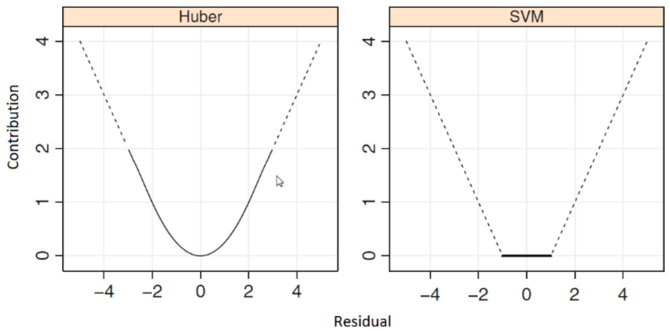


Fig.12 [9]: The relationship of model residual and its contribution to the regression line.

Since squared residuals are not used in the cost function, there is limited effect of large outliers on the regression equation. However, if a relatively large value is used for threshold, then only the outlier points contribute to the regression line. However, this approach has shown its effectiveness in defining the model.

Fig.13 shows the comparison between robustness qualities of SVM model and ordinary linear regression model in the existence of outliers. It is obvious that the OLS model (red line) is pulled towards the outlying points, while the SVM model (blue line) better describes the overall structure of the data.

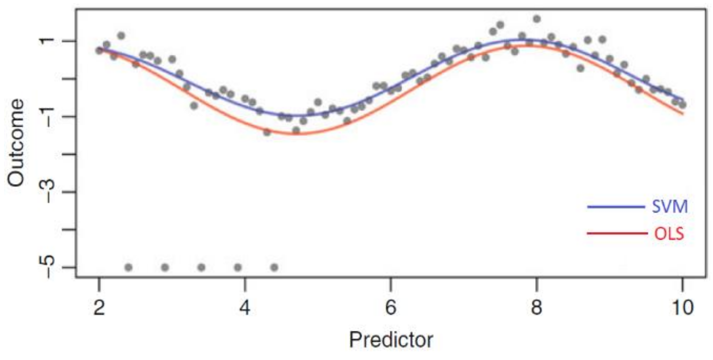


Fig.13 [9] Comparison of robustness qualities of SVM model and ordinary regression model.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) algorithm is a non-parametric method which simply predicts a new observation using the K-nearest observations from the training set [41]. KNN can be both applied for classification and regression. In regression prediction, KNN identifies the desired sample's k nearest neighbors in the feature space. The predicted value of the desired sample is generally the average of the k neighbor's responses.

KNN defines neighbors based on the distance between samples. There are many distance measurements, such as Tanimoto, Hamming, and cosine [42]. To decide which type of distance measurement to use, one should make the decision under specific context. Among various kind of distance measurements, Euclidean distance of the most commonly used and is defend as follows:

$$\left(\sum_{\mathrm{j}=1}^{P}\left(\mathrm{y}_{aj}-y_{bj}\right)^{2}\right)^{\frac{1}{2}} \tag{14}$$

Where y_a and y_b are two individual samples. It is worth noticing that all the predictors must be centered and scaled prior to performing KNN since distance measurement is used.

The number of neighbors should be chosen carefully for too few neighbors for it may result in over-fitting individual observations while too many neighbors may result in poor predictive performance. To find the optimal number of neighbors, K can be determined by resampling. Fig.14 illustrates the parameter tuning process of KNN. After the candidate set of K parameter is defined, the data would be resampled multiple times to estimate model performance for each K value.

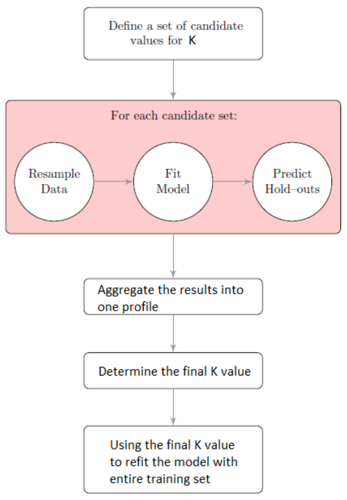

Define a set of candidate values for K
For each candidate set:
Resample Data
Fit Model
Predict Hold-outs
Aggregate the results into one profile
Determine the final K value
Using the final K value to refit the model with entire training set

Fig.14 A diagram of parameter tuning process of KNN.

K-Nearest Neighbors algorithm is intuitive and straightforward and have a remarkable predictive ability, especially when the response is related to the local predictor structure. However, the computational time is a noticeable problem. To predict a sample, distances between the observation and all other observations must be computed, thus the computation will increase dramatically with K and data size.

K-dimensional tree (k-d tree) proposed by Bentley (1975) overcomes the time-costly problem by replacing the original data with a less memory intensive representation of the data [43]. Instead of loading the data, it only loads the description of locations of the original data. Observations are placed through a tree structure. When predicting a new sample, k-d tree only computes the distances of those observations in the tree that are close to the new sample. It has been proven that when the number of training observations is much larger than the number of predictors, k-d tree provides significant computational improvements.

Model Selection

Among various machine learning algorithms, linear regression models (OLS and PLSR), neural network and K-nearest neighbor models are chosen to learning the data. The reasons behind the choices are presented below.

Linear regression algorithm is simple and efficient, the model is highly interpretable and it is considered as the most basic machine learning algorithm. If linear regression model can produce a decent predictive performance, there is no sense to seek for more complicated and time consuming algorithms. [44] applies linear regression to forecast electricity consumption in Italy, and achieved high predictive accuracy with adjusted regression coefficients equal to 0.981 for total consumption. Preliminary observation of data shows that the target value has a strong correlation with some of the predictors, which indicates a linear regression model is worth a try.

In contrast, neural network is a powerful data-driven and self-adaptive tool, which has the capability of capturing nonlinear and complex underlying characteristics of any physical process with a high degree of accuracy. It can handle large amount of data and has the ability to detect all possible interactions between predictor variables. [45] uses neural network to develop an investment system to predict takeover targets. The model incorporates various predictors and exhibits highly successful prediction rate. [46] proposes a neural networks model for wind power forecasting, results show that the proposed model outperforms high effectiveness.

K-Nearest Neighbors algorithm is a nonparametric method and it is one of the simplest machine learning algorithm. The algorithm is interpretable and straightforward with excellent predictive performance. [47] compares the performance of k-nearest neighbor(k-nn) model and methods based on parametric distribution for predicting the basal area diameter distribution. The experiment results shows that the k-nn regression models give a more accurate description of the basal area diameter distribution than the parametric methods. [48] applies k-nearest neighbor algorithm to predict stock prices for a sample of six major companies. The results demonstrate that the k-nearest neighbor algorithm is robust with small error ratio, and the prediction results were close and nearly parallel to actual stock prices.

The selected models have achieved success in many applications. Besides, the three models have different strengths towards different data structures. Thus, we believe that at least one of these models will produce accurate prediction in our case.

Over-Fitting problem and Model Tuning

The Problem of Over-Fitting

In the field of forecasting, there is a situation where the applied model can learn the structure of the dataset very well while performing poorly when predicting new data, in this case, the model is most likely to be over fit. This occurs when the model not only learns the general patterns in the data but also learns the characteristics of noise.

To illustrate the over-fitting concept, consider the following classification example in Fig 15. A company tries to find the customers that are susceptible to buy their product and send them a brochure. They try to distinct the target group from others based on their age and education. As is shown in Fig 15, model 1 shows a complex boundary and attempts to make absolutely no error on the training dataset. The estimated error rate in this panel is over optimistic and this model is not likely to generalize to new data, this would result in poor predictive performance. The right-hand figure shows an alternative model to fit the training data, the boundary is fairly smooth and just learn the main pattern of the data. The second model performs much better in predicting new data [49].

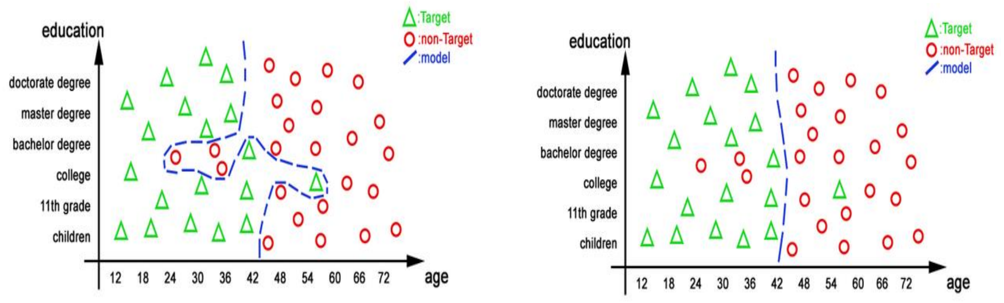


Fig.15 [48]: An example of classification data that illustrates the over-fitting problem.

(Left: Classification model 1. Right: Classification model 2)

Methods

There are several approaches to help avoid over-fitting problem, these methods can be roughly classified into two categories: data splitting and resampling techniques.

Data Splitting

As has been discussed previously, models may produce overly optimistic performance estimates during learning processes, a good approach to inspect the estimate is to test the model on samples that were not used in training process. For example, one can split the data set into “training” data set and “test” or “validation” data set, using the “training” data set to create the model while using the “test” or “validation” data set to qualify performance.

A simple way to split the data is to take a simple random sample. However, data attributes are not controlled in this way and it is risky because the distribution of the outcomes may be different between the training and test data sets. A better approach would be applying random sampling within subgroups.

Another data splitting approach proposed by Willett (1999) and Clark (1997) is based on maximum dissimilarity sampling [50]. There are several ways to measure dissimilarity, the simplest one is to use the distance between two sample values. Longer distance between two sample points indicates larger dissimilarity. Martin et al. (2012) compares different data splitting methods, including random sampling, dissimilarity sampling and other methods [9].

Resampling Techniques

Resampling techniques as an alternative approach for estimating model performance is to resample the training set [51]. Similarly, the process uses a subset of samples to fit a model and uses remaining samples to evaluate the performance of the model. This process is repeated multiple times and then applies statistical methods to provide honest estimates of model performance.

K-Fold Cross-Validation

In k-fold cross-validation, the samples are randomly partitioned into k roughly equal sized subsamples. Among the k sample sets, the first subset is retained as the validation data for testing the model, and the held-out samples are used as training data to fit the model. The first subset is then return to the training data and the process repeats k times while each of the k subsets used exactly once as the validation data. The k resampled estimates of performance are aggregated and summarized to produce a single estimation.

As is illustrated in Fig 16, the training set are partitioned into five groups, these groups are left out in turn as validation data. Performance estimates are calculated from each set of remaining subsamples. The cross-validation estimate of model performance would be the average of the five performance estimates.

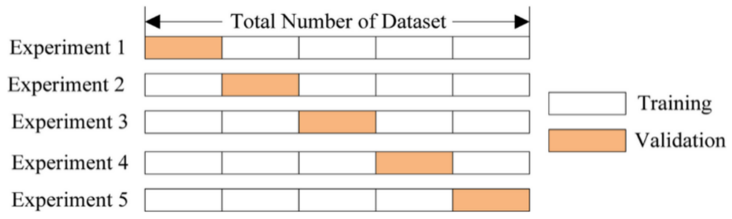


Fig.16 [9] A schematic of fivefold cross-validation.

Resampling techniques operates quite similarly, other resampling methods such as generalized cross-validation, repeated training/tests splits, bootstrap can be refer to paper [9].

Chapter III – The Model

The chapter starts with the hypothesis which formulates the aim of our work. An outline of the methodology to verify the hypothesis is followed. Then a description of preliminary data exploratory result is given. The explanation of experiments are presented by giving the methodologies of building each model and important implementation of to realize the experiments. The chapter ends with the test results and necessary analysis.

Hypothesis

This work is intended to use machine learning to build predictive models for given data. Among various machine learning algorithms, linear regression, neural network and K-nearest neighbors are chosen.

These models have been successfully applied in various applications as described in the last chapter. The challenge is the background of the dataset and the physical meaning of the predictors and response are kept unknown, thus, there is no prior knowledge that can be used for reference and no previous predictive model can be straightforwardly applied in this work. The only thing we can rely on is the data. This work tries to verify that whether the three chosen models can be applied in the given data and have a satisfying predictive performance.

The following hypothesis formulates the aim of our work:

H: *Given the data, we may use linear regression, or neural network, or K-nearest neighbors for data prediction.*

The authenticity of the hypothesis is verified using the methodology demonstrated in the following section.

Methodology

To verify our hypothesis, the following methodology was performed:

1. Data preprocessing: remove irrelevant predictors and highly correlated predictors, data transformation.
2. Creation, training and estimation of linear regression models, including ordinary linear regression model and partial least square model.
3. Creation, training and estimation of neural network models, including feed-forward neural network with single hidden layer, deep belief neural network with restricted Boltzmann machine, neural network with backpropagation algorithm.
4. Creation, training and estimation of K-Nearest neighbor model and K-dimensional tree model.
5. Comparison of predictive performance and computational performance between linear regression, neural networks, k-nearest neighbors and k-dimensional tree models.

If the results were in line with our expectations, then the hypothesis would be considered as true.

Data Description

The given dataset consists of 144 predictors (named X1, X2…X145) and a response variable to be predicted (named X62). There are 783679 observations in total which are split into 6 csv files.

No missing data is found in the dataset, while there are some predictors just containing NULL values or unique value which should be removed.

Data Preprocessing

Methodology

Data preprocessing follows the steps:

1. Remove zero-variance and near zero-variance predictors.

2. Split the dataset into training dataset and test dataset.

3. Filter the highly correlated predictors using algorithm 1.1.

4. Data transformation: Box-Cox transformation, centering and scaling.

Remove Zero- and Near Zero-Variance Predictors

The raw data contains quite a few predictors that only contain null value or unique value, these predictors have no contribution to predicting the response, thus they are considered as irrelevant predictors which should be removed.

Code:

```
## Remove near zero-variance predictors
    library(caret);
    near_zero <- nearZeroVar(data);
    length(near_zero);
    data_pre <- data[,-near_zero];
```

Result: The number of zero-variance and near zero-variance predictors is 98, after removing them, the number of remaining predictors in the dataset is 46.

Data splitting

To examine the predictive performance of regression models, one needs a clean dataset to be tested on. Thus the original dataset is split into training dataset and test dataset, and the training dataset contains 80% of total observations in the original dataset while the test dataset contains 20% of total observations. The training dataset is used for training the model and fine tuning parameters, while the test dataset is kept clean to perform model prediction on it. Besides, this paper split the predictors and response into two datasets.

```
Code:

## Data Splitting

    set.seed(1)

    trainingRows <- createDataPartition(data_pre$X62,p=0.80,list=FALSE)

    training.set <- data_pre[trainingRows,]

    testing.set <- data_pre[-trainingRows,]

    rm(trainingRows)

    trainPredictors <- training.set[,-match("X62",names(training.set))]

    trainResponse <- training.set[,match("X62",names(training.set))]

    trainResponse <- data.frame(trainResponse)
```

Filter highly correlated predictors using algorithm 1.1

To identify the highly correlated predictors in the data, we first plot the correlation matrix to visualize the correlation between predictors.

```
Code:

## visulaization of the correlation matrix of predictors

    corr_data <- cor(data_pre);

    library(corrplot);

    corrplot(corr_data,order="hclust");
```

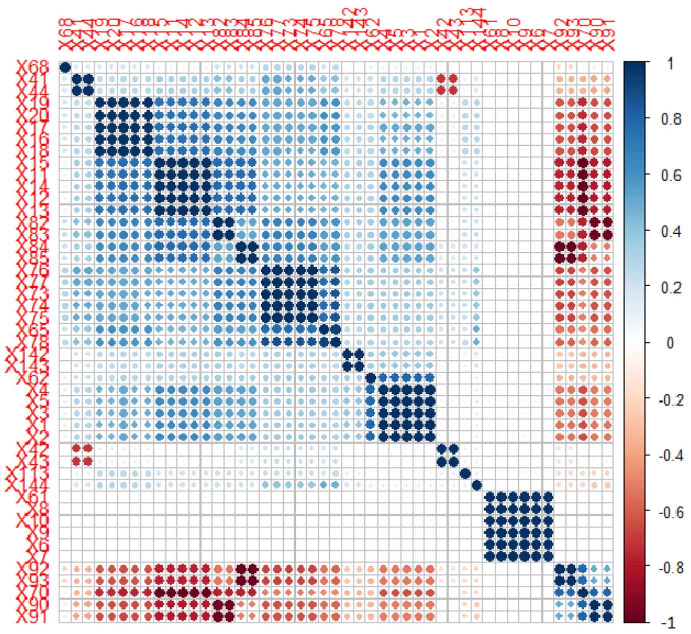


Fig.17 Correlation matrix of predictors

Fig.17 shows the plot of the correlation matrix of predictors, the dark blue and dark red points in the figure represents that highly correlated predictor pairs. And it is not hard to see that they are gathering in groups, e.g. x73, x74, x75. Thus it is possible for us to remove some of them to get fewer features.

What’s more, if we take a deep look into the figure, we can find that the variable X62, which is the predicted value, has high correlation coefficient with predictors x1, x2, x3, x4, x5. This implies that there may be a strong linear relationship between the predicted value x62 and the predictors.

To filter based on correlations, this paper follows algorithm 1.1 and applies the findCorrelation function in R as shown in the code below. For a given threshold of pairwise correlations, the function returns column numbers denoting the predictors that are recommended for deletion. Here the threshold is set to 0.85.

```
Code:

##Function: high correlated predictors filter

    filter1 <- function(x){

    library(caret)

    corr_data <- cor(x);

    highCorr_data <- findCorrelation(corr_data, cutoff = 0.85);

    x$X3 <- (x$X1+x$X2+x$X3+x$X4+x$X5)/5

    x$X19 <- (x$X16+x$X17+x$X18+x$X19+x$X20)/5

    x$X85 <- (x$X84+x$X85)/2

    x$X65 <- (x$X65+x$X78)/2

    x$X61 <- (x$X6+x$X7+x$X8+x$X9+x$X10+x$X61)/6

    trainPredictor_filter1 <- x[,-highCorr_data];

    return(trainPredictor_filter1)

    }
```

Fig.18 shows the correlation matrix of predicors after filtering the highly correlated predictors. The correlation coefficients between the remaining 13 predictors are less than 0.85.

It is worth mentioning that this paper does not use the remaining predictors directly to build the model, instead, we use the average of the highly correlated predictor pairs to better summarize the information provided by predictors. For example, instead of using predictor X3, we use the average highly correlated predictor group (X1, X2, X3, X4, X5).

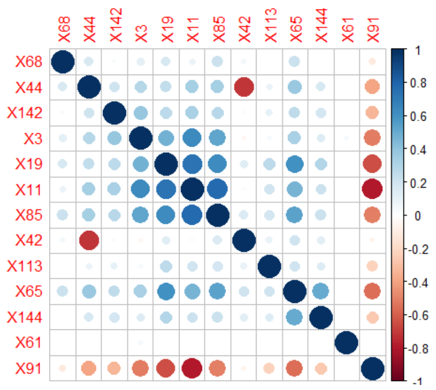


Fig.18 Correlation matrix of predictors after filtering the highly correlated predictors.

Data Transformation for predictors

As last step of data preprocessed, this paper performs basic transformation technoligies on predictors. Box-Cox transformation is applied to help better normalizing the data distribution of individual predictors, while data centering and scaling are common basic requirement for regression models.

```
Code:

## Function: datascale(x)

   datascale <- function(x){

   library(caret);library(e1071)

   tran <- preProcess(x,method=c("BoxCox","center","scale"))

   x_scale <- predict(tran, x)

   return(x_scale)

   }
```

Experiments

Experiments are carried out using software R and Rstudio.

For linear regression models, ordinary linear regression (OLS) and partial least squares regression (PLSR) are implemented. And for neural network, feed-forward neural network with a single hidden layer model, deep belief network with restricted Boltzmann machine, and multiple perceptron with back propagation algorithm have been implemented. And for nearest neighbor algorithm, both k nearest neighbor (knn) and nearest neighbors with k-dimensional tree have been implemented.

To estimate the predictive performance of the model, we mainly use R-squared and residual standard error (RMSE) to estimate the performance of the model. R-squared is statistical measure that indicates how well the data is fit to a statistical model. And RMSE is used as a measure of the differences between predicted values of a model and the observed values. In addition to R-squared and RMSE, the visualization plot of predicted values vs observed values is also an important reference to estimate the predictive performance of a model.

The experiments are explained by demonstrated the methodology followed by test results and analysis.

Linear Regression

Methodology

(1) Ordinary Linear Regression

a. Utilize data preprocessing results as input for the model

b. Train the model with training dataset

c. Test the model on new data (test dataset)

d. Estimate the model predictive performance (R-Squared, RMSE, visualization of predicted values over observed values)

The model is built on R, using R package “caret” [54] to train the model.

(2) Partial Least Squares Regression

a. Data preprocessing (without removing highly correlated predictors)

b. Train the PLSR model with a set of components

c. Select the optimal number of components, fine tuning the model.

d. Test the model on new data (test dataset)

e. Estimate the model predictive performance (R-Squared, RMSE, visualization of predicted values over observed values)

The methodology for partial least squares regression experiment is similar to the methodology of ordinary linear regression model. Unlike OLS, which is built on PLSR model, we leave out the step of removing highly correlated predictors. Thus, there are in total 46 predictors input to PLS model.

Result and Analysis

(1) Ordinary Linear Regression model

The text box below shows the R-squared and RMSE estimates of OLS model. The R-squared of OLS as shown is 0.7365 which is not very high but still indicates that there is linear correlation between predictors and response. Test the model with new data, the R-squared of predicted values and actual values is 0.7386, which is quite close to the R-squared estimated by model training.

```
Residual standard error: 64450 on 626930 degrees of freedom
Multiple R-squared:   0.7365,     Adjusted R-squared:   0.7365
F-statistic: 1.348e+05 on 13 and 626930 DF,   p-value: < 2.2e-16

> postResample(OLS_Pred, testResponse$testResponse)
          RMSE       Rsquared
6.419733e+04 7.386356e-01
```

Fig.19 visualizes the relationship between values predicted by OLS model and the actual observed values. As we can see from the figure, the majority of the points fall into a line while there are still quite a few points that fall outside the line. Especially the points which fall in the vertical line and horizontal line as shown in the figure implies that there is certain pattern in the data that the OLS does not learn.

Fig.20 shows the relationship between the predicted values and the residuals of OLS model. If the data well fits the model, then the residuals should appear to be around 0 with respect to the predicted values. However, the residuals which are shown in Fig.17 have a tendency to grow with respect to certain predicted values. This again reveals the fact that the OLS model does not learn all the features from the data.

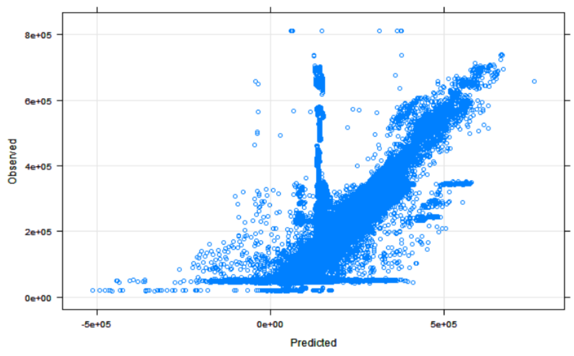


Fig.19 A plot of predicted values vs observed values in OLS.

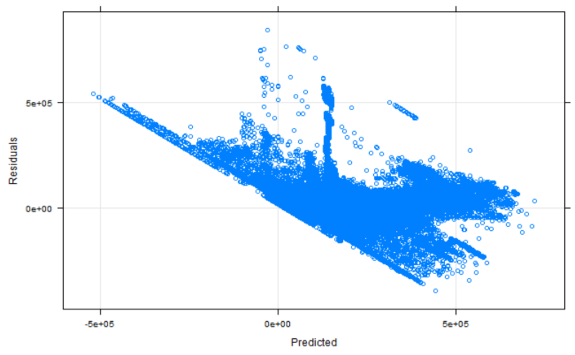


Fig.20: A plot of predicted values vs residuals values in OLS.

Fig.21 and Fig.22 shows the actual observed values from data and the predicted values from OLS model respectively. Fig.18 is exactly the result we try to reproduce. From Fig.19, we can conclude that OLS model presents some main features of the data, but it is not precious enough for the application.

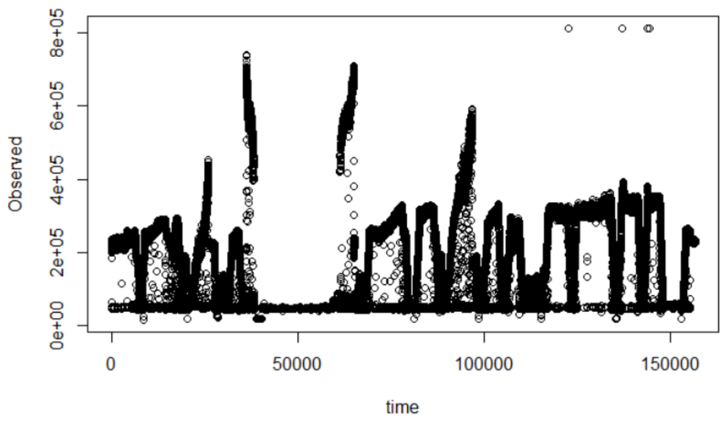


Fig.21 The actual observed values of response variable

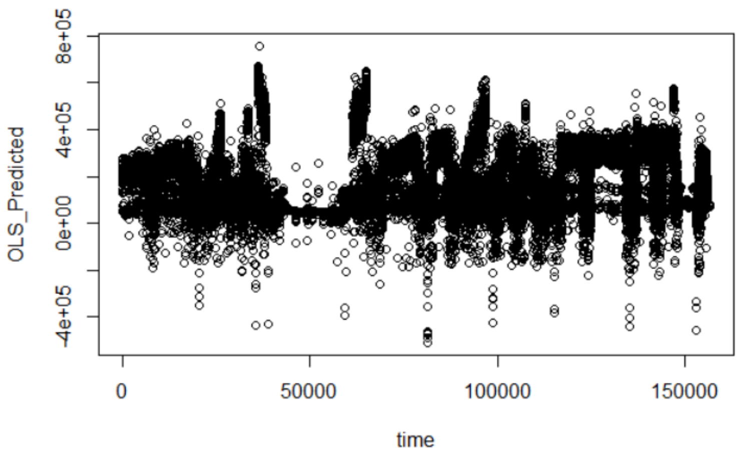


Fig.22 The predicted response values of OLS model

(2) Partial Least Squares Regression (PLSR)

We first train PLSR model with a set of components whose number of components ranges from 1 to 15, and get the result as shown in Fig.23. The figure shows a dramatic drop of RMSE when the number of components increases from 1 to 2, and decreases gradually with the increasing number of components. Details are shown in Fig.24.

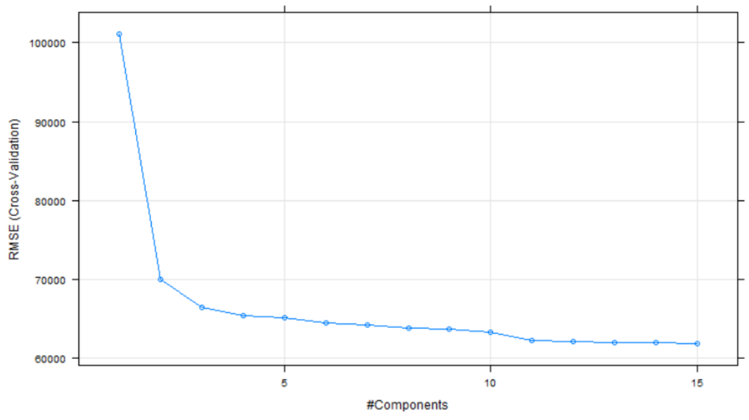
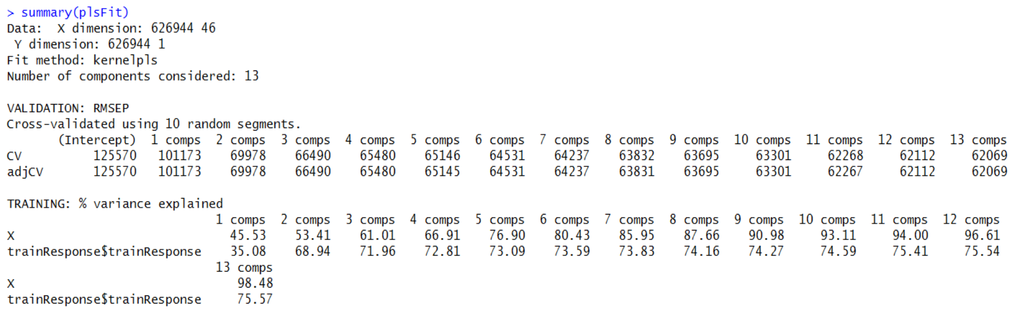


Fig.23 RMSE of PLSR models with different number of components

The following figure shows the detail of PLS regression fit. From the RMSE aspect, the RMSE changes very little after the number of components is greater than 3. However, as we seek to capture a majority of the information in original predictors, 13 components which explain 98.48% information of original predictors is considered appropriate. Thus, 13 PLS components are selected to build the model, and 75.57% variance of the response variable is explained.



```
> summary(plsFit)
Data:  X dimension: 626944 46
 Y dimension: 626944 1
Fit method: kernelpls
Number of components considered: 13

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV          125570   101173    69978    66490    65480    65146    64531    64237    63832    63695     63301     62268     62112     62069
adjCV       125570   101173    69978    66490    65480    65145    64531    64237    63831    63695     63301     62267     62112     62069

TRAINING: % variance explained
                            1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
X                             45.53    53.41    61.01    66.91    76.90    80.43    85.95    87.66    90.98     93.11     94.00     96.61
trainResponse$trainResponse   35.08    68.94    71.96    72.81    73.09    73.59    73.83    74.16    74.27     74.59     75.41     75.54
                            13 comps
X                              98.48
trainResponse$trainResponse    75.57
```

Fig.24 Summary of PLSR regression fit

The importance of each individual predictors in related to the response which is shown in Fig.25. The top 5 most important predictors are X3, X2, X5, X1, X4, which have the highest correlation coefficients with response X62. This figure gives us a clear recipe that shows which predictors are the most important and which are of little use and even can be removed.

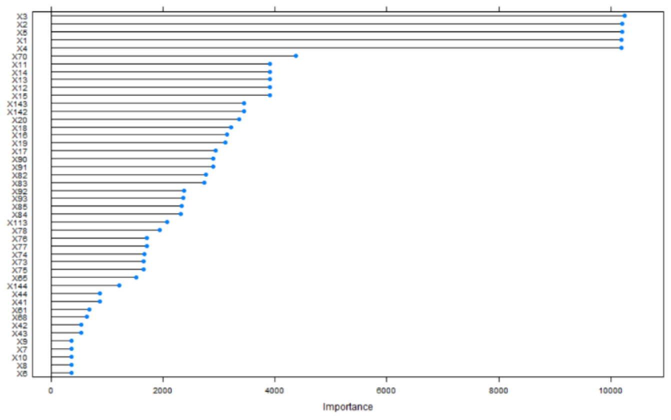
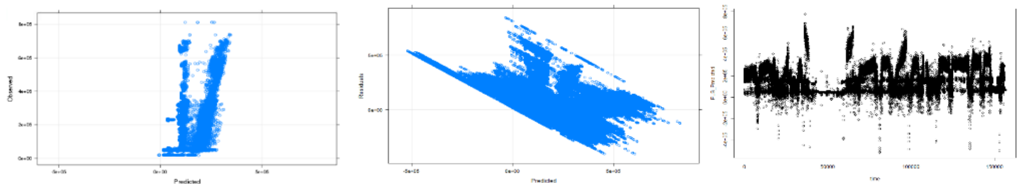


Fig.25 PLS predictor importance scores for the data

We then test the trained model on new data, the Rsquared estimate of predict values of PLSR model and the actual observations is 0.757041, and the Root mean squared error is approximately 61894. Compared with OLS, the predictive performance of PLSR model has improved, but not that much. Predictive performance of PLSR model is visualized in Fig.26.



(a) (b) (c)

Fig.26 Visualization of predictive performance of PLSR model. (a) Predicted values VS observed values. (b) Residuals vs Predicted values (c) Predicted values distribution.

The results of the experiment show nonlinear relationship between the response and the predictors in which linear regression model cannot fit. More advanced model is needed for modeling the data.

Neural Network

Methodology

(1) Neural network with single hidden layer

a. Data preprocessing following the methodology introduced previously

b. Create candidate models with single hidden layer and different number of hidden units.

c. Train the models with resampling techniques and select the model which fits the data best.

d. Test the selected model on new data

e. Estimate the model predictive performance (R-Squared, RMSE, visualization of predicted values over observed values)

f. Estimate the computational performance of the model

It is worth to mention that in neural network models, the input variables including predictor variables and target variable have been linearly scaled to range 0~1. As demonstrated in [28, 52], standardization of target variable is a convenience for getting good initial weights and assures convergence. Since it is a linear transformation, it is easy to recover to original value.

Besides, neural network algorithm is a nondeterministic algorithm, each experiment run will produce different results. Thus, in order to get an honest performance result, each model should be run several times and take the average of the estimates to evaluate the performance. R package “caret” offers method “avNNet” which stands for average neural network, it runs the model several times and take the average as performance estimates.

(2) Deep Belief Network With Restricted Boltzmann Machine (RBM)

a. Data preprocessing, input data is scaled to 0~1
b. Create DBN models with stacked RBMs of different number of hidden layers
c. Train candidate DBN models and select the optimal model
d. Estimate the predictive performance of the model.

RBM are often used as a modeling technique over binary vectors. In order to model continuous data, the input data needs to be scaled to the interval [0, 1] and modeled by the probability of visible variables to be one [9].

DBN is also a nondeterministic algorithm, thus experiment repetition is required to get a reliable performance estimate. In our experiment, each model has been run 10 times and average estimates have been taken to evaluate the performance of the model.

Implementation of deep belief network with stacked RBMs uses R package "deepnet" [55].

(3) Neural network with backpropagation algorithm

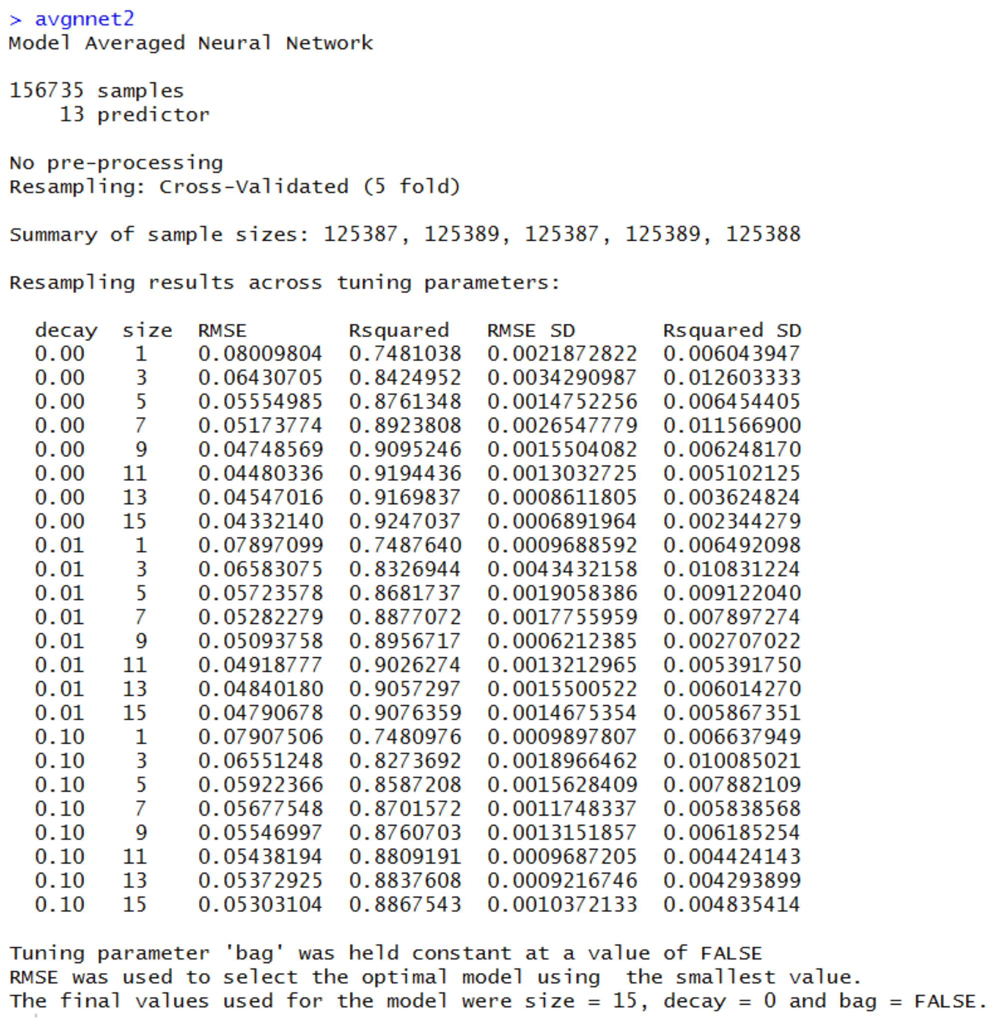
   a. Select the optimal NN model from previous models (NN with single hidden layer and DBN with RBMs)
   b. Train the selected model with backpropagation algorithm.
   c. Estimate the predictive performance of the new model
   d. Estimate the computation performance

Implementation of neural network with backpropagation uses R package "RSNNs" [56].

Result and Analysis

(1) Neural network with single hidden layer

Fig.27 shows the summary of neural network model with single hidden layer. Because of the non-deterministic property of neural network, thus each model is run several times and we use the average parameter estimates to evaluate the performance of the model. The inputs of the model is made up of 13 predictors with 156736 observations. Data preprocessing has been done beforehand, 5-fold cross validation has been applied.



```
> avgnnet2
Model Averaged Neural Network

156735 samples
    13 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)

Summary of sample sizes: 125387, 125389, 125387, 125389, 125388

Resampling results across tuning parameters:

  decay  size  RMSE        Rsquared   RMSE SD       Rsquared SD
  0.00    1    0.08009804  0.7481038  0.0021872822  0.006043947
  0.00    3    0.06430705  0.8424952  0.0034290987  0.012603333
  0.00    5    0.05554985  0.8761348  0.0014752256  0.006454405
  0.00    7    0.05173774  0.8923808  0.0026547779  0.011566900
  0.00    9    0.04748569  0.9095246  0.0015504082  0.006248170
  0.00   11    0.04480336  0.9194436  0.0013032725  0.005102125
  0.00   13    0.04547016  0.9169837  0.0008611805  0.003624824
  0.00   15    0.04332140  0.9247037  0.0006891964  0.002344279
  0.01    1    0.07897099  0.7487640  0.0009688592  0.006492098
  0.01    3    0.06583075  0.8326944  0.0043432158  0.010831224
  0.01    5    0.05723578  0.8681737  0.0019058386  0.009122040
  0.01    7    0.05282279  0.8877072  0.0017755959  0.007897274
  0.01    9    0.05093758  0.8956717  0.0006212385  0.002707022
  0.01   11    0.04918777  0.9026274  0.0013212965  0.005391750
  0.01   13    0.04840180  0.9057297  0.0015500522  0.006014270
  0.01   15    0.04790678  0.9076359  0.0014675354  0.005867351
  0.10    1    0.07907506  0.7480976  0.0009897807  0.006637949
  0.10    3    0.06551248  0.8273692  0.0018966462  0.010085021
  0.10    5    0.05922366  0.8587208  0.0015628409  0.007882109
  0.10    7    0.05677548  0.8701572  0.0011748337  0.005838568
  0.10    9    0.05546997  0.8760703  0.0013151857  0.006185254
  0.10   11    0.05438194  0.8809191  0.0009687205  0.004424143
  0.10   13    0.05372925  0.8837608  0.0009216746  0.004293899
  0.10   15    0.05303104  0.8867543  0.0010372133  0.004835414

Tuning parameter 'bag' was held constant at a value of FALSE
RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were size = 15, decay = 0 and bag = FALSE.
```

Fig.27 Summary of averaged neural network model with single hidden layer.

Fig.28 illustrates the RMSE profile of averaged neural network. It is obvious that RMSE decreases with increase of the number of hidden units, and when the number of neurons is greater than 10, RMSE gradually decreases with increasing number of neurons. However, computational time increases significantly with the increase in the number of hidden units. Thus, there is a tradeoff between predictive performance and computation time. Therefore, the optimal model we choose in this case is made up of 15 hidden units with weight decay equals to 0.1.

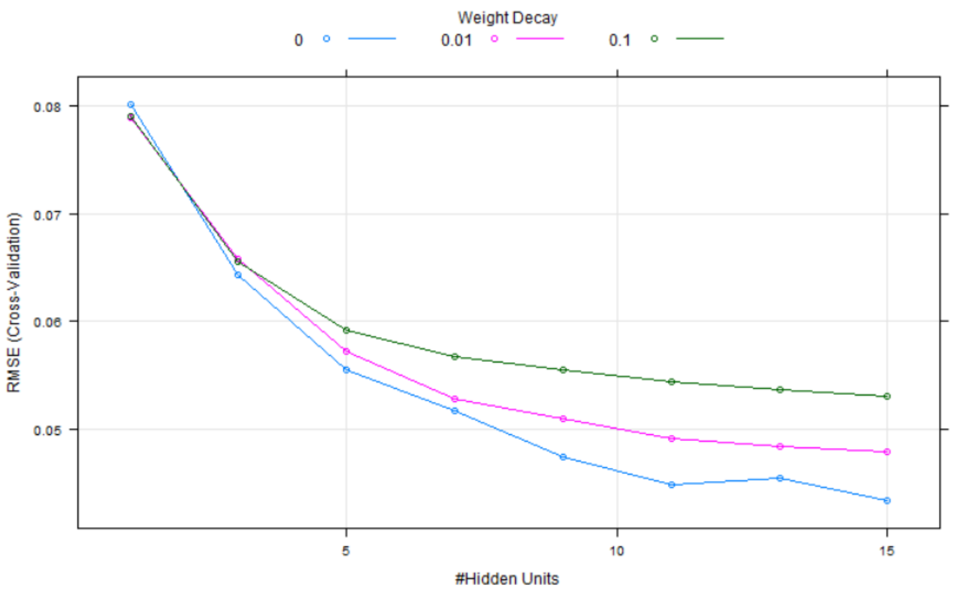


Fig.28: The RMSE cross-validation profile for averaged neural network model

We build the selected model and train it on the full training dataset which has 626944 observations. And we test the model on the test dataset which has 156735 observations. Computational time is recorded. The performance estimates of the model is listed in Table 1. And Fig.29 illustrates the relationship of predicted values of this model and the actual observed values.

Table 1 Performance estimates of neural network with a single layer of 15 neurons

Model	R-squared	RMSE	Computation time (seconds)
Averaged Neural network (size = 15)	0.9215101	37602.73	2793.106

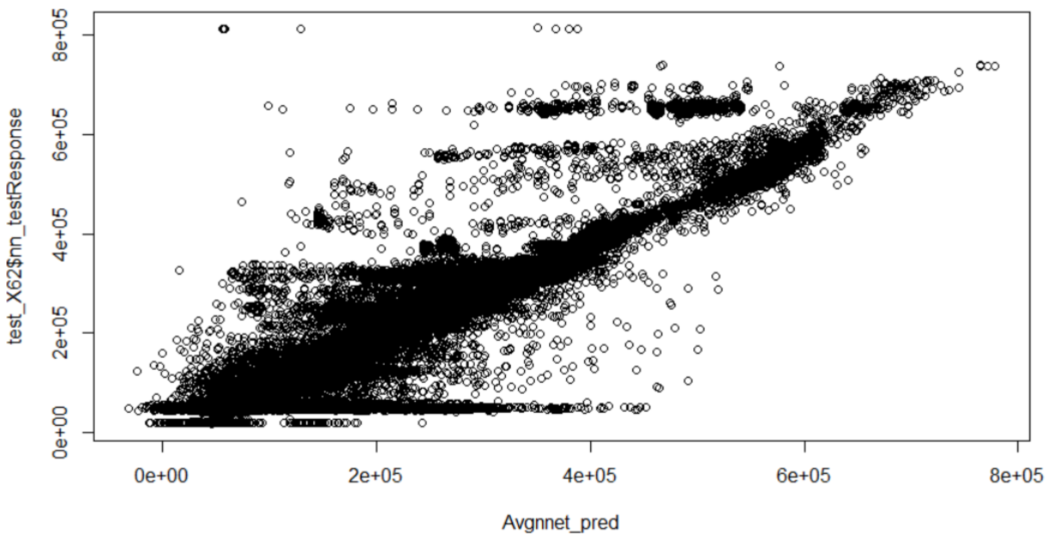


Fig.29 Predicted values of single hidden layer neural network model vs observed values

(2) DBN with stacked RBMs

To find a best structure of DBN, we built several models to learn the same data. They are DBN with single layer RBM with 15 neurons, DBN with 2 layers RBM and 3 layers RBM, each layer is consist of 15 hidden units. The experiment results are summarized in Table 2.

Table 2. Performance estimates of DBNs with different structures

Model	R-squared	RMSE	Averaged computation time
DBN with 1 layer RBM	0.858203	48395.04	5233.42
DBN with 2 layer RBM	0.859157	47989.46	6028.19
DBN with 3 layer RBM	0.856225	48765.38	8707.55

The predictive performance of three DBN models are no much difference with R-squared approximately equals to 0.858, which is worse than the neural network model with single hidden layer. Besides, the computation time of DBN models increases with the increase of RBM layers. Thus, we can conclude that deep belief network with restricted Boltzmann machine algorithm is not suitable for our data.

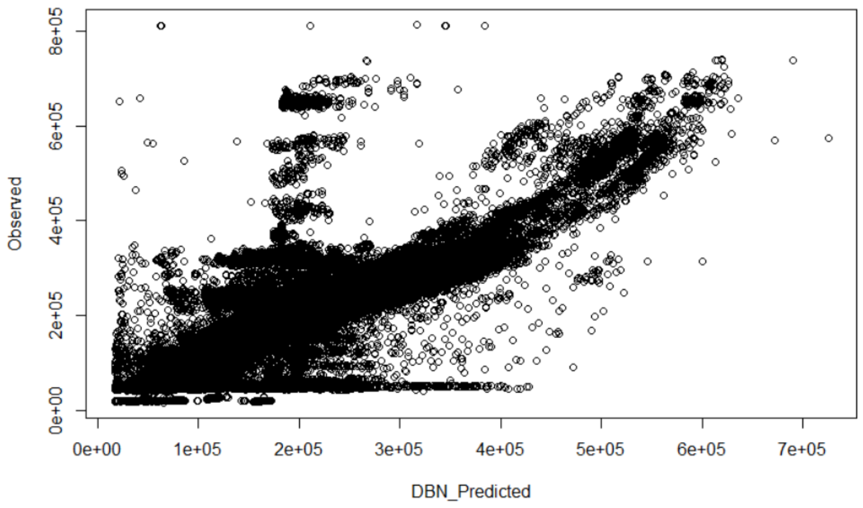


Fig.30 Predicted values of DBN with 2 layers RBMs vs actual observed values

(3) Neural network with backpropagation algorithm

Based on the experience gained from previous experiments on neural network and the research on backpropagation algorithm, we then build a backpropagation neural network with 2 hidden layers of 15 neurons. The structure of the model is shown in Fig.31.

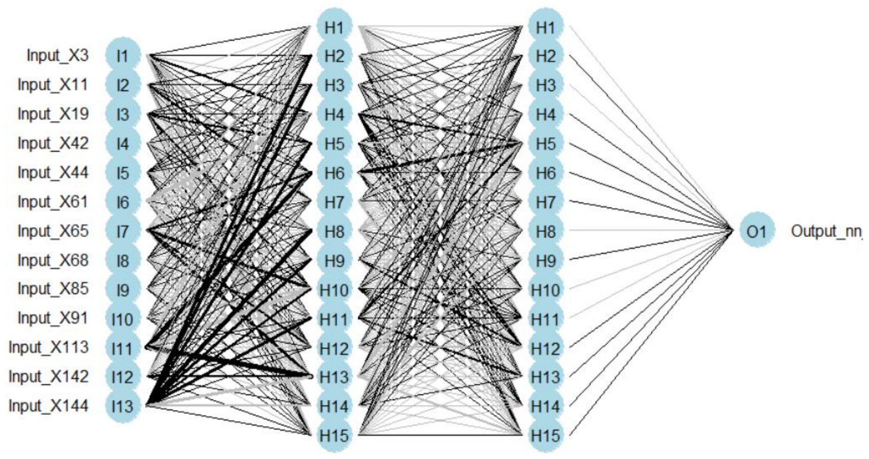


Fig.31 The structure diagram of backpropagation neural network model with 2 hidden layers.

Table 3 summarize the performance estimates of our backpropagation neural network model. The R-squared estimate of this model is 0.9586 which is quite high and shows excellent predictive performance of our backpropagation neural network model. The computation time of training this model is 3731.42s, which approximately equals to 62 minutes.

Table 3 Performance estimates of backpropagation neural network model

Model	R-squared	RMSE	Computation time (seconds)
Backpropagation neural network (size = 15)	0.9585961	28077.00	3731.42

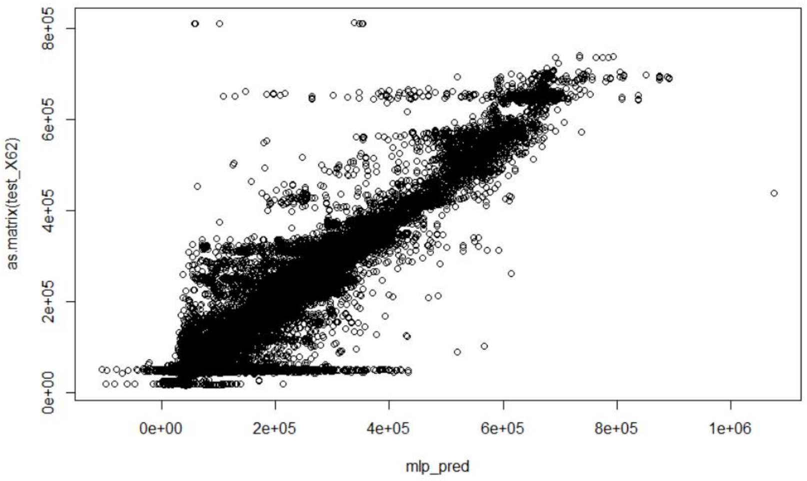


Fig.32 Predicted values vs actual observed values in backpropagation neural network model

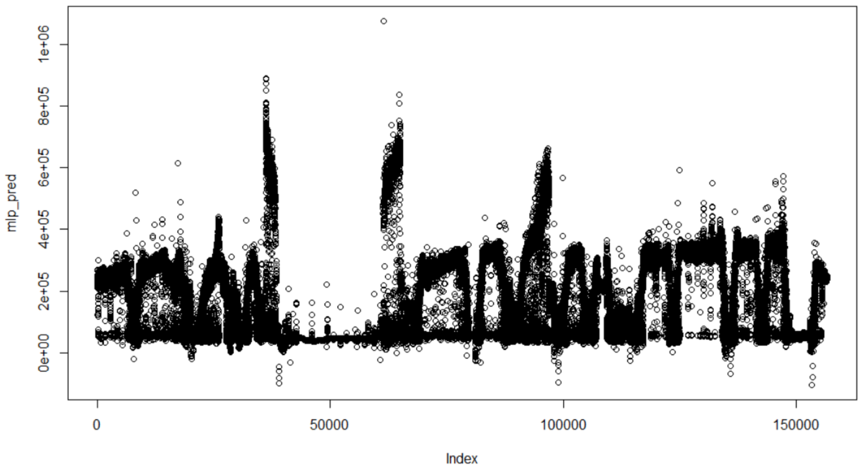


Fig.33 The predicted values of backpropagation neural network model.

Fig.32 and Fig.33 visualizes the predicted values of the backpropagation neural network model and the comparison between predicted values and actual values. The figures again show that our backpropagation neural network model produce predictions which are very close to actual values. Thus among the three neural network models we have built, the backpropagation neural network model is considered the most suitable model for our data.

K-Nearest Neighbor and K-dimensional tree (k-d tree)

Methodology

a. Data preprocessing following the methodology introduced previously

b. Create models with a candidate set of k value (e.g. 1, 3, 5, 7, 9)

c. Train the models with resampling techniques and select the model which fits the data best.

d. Test the selected model on new data

e. Estimate the model predictive performance (R-Squared, RMSE, visualization of predicted values over observed values)

f. Train k-d tree model with optimal k value selected in K-Nearest neighbor model, compare the predictive performance of k-d tree and k-nearest neighbor model.

g. Compare the computational performance of K-Nearest neighbor and k-d tree model.

The experiment shows that the K-nearest neighbor model is very time consuming. Thus, instead of using all the dataset, we use a small sample of 30000 observations for tuning parameters (step c). Once the optimal k value is selected, we then apply the whole dataset to train the selected model.

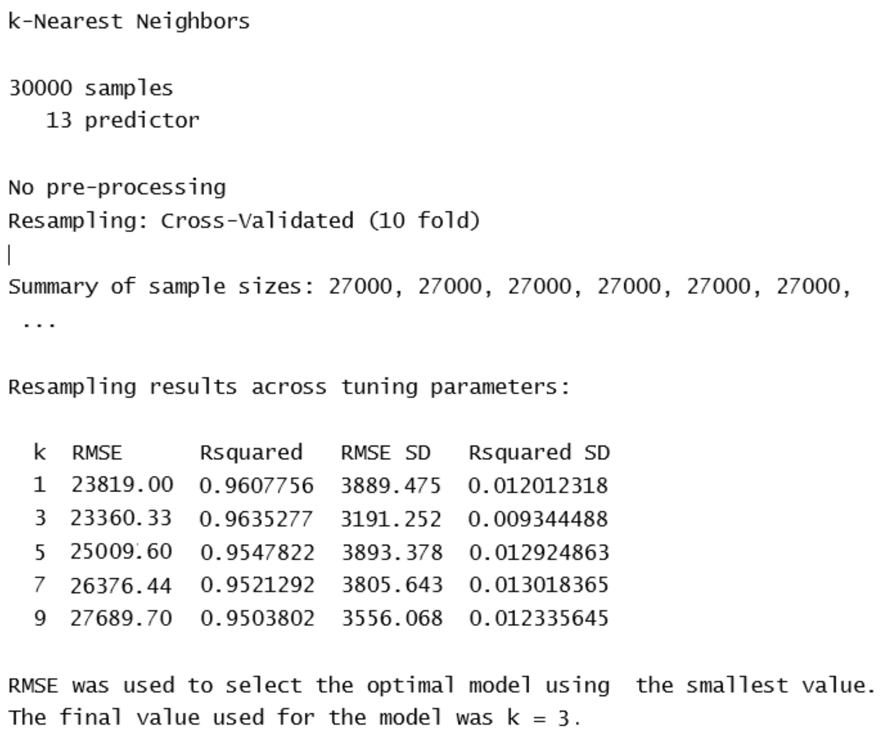
It is also worth to mention that K-nearest neighbor and k-d tree is a deterministic algorithm. Once the k value is selected, there is no need to train the model multiple times with same parameters. However, resampling techniques is applied to avoiding the over fitting problem.

Implementations of nearest neighbor algorithms use R package “RANN” [57].

Result and Analysis

Fig.33 shows the key information of k-nearest neighbors training model. The selected candidate set of k values is k = 1, 3, 5, 7, 9, the inputs of the model is made up of 13 predictors with 30000 observations. Data preprocessing has been done beforehand, 10-fold cross validation is applied.

As shown in Fig.34, the R-squared estimates are above 0.95 for all k candidate values, which are pretty high. The optimal k value selected for the model is 3, which has the smallest RMSE, as shown in Fig. 35.



```
k-Nearest Neighbors

30000 samples
   13 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 27000, 27000, 27000, 27000, 27000, 27000,
 ...

Resampling results across tuning parameters:

  k  RMSE      Rsquared   RMSE SD   Rsquared SD
  1  23819.00  0.9607756  3889.475  0.012012318
  3  23360.33  0.9635277  3191.252  0.009344488
  5  25009.60  0.9547822  3893.378  0.012924863
  7  26376.44  0.9521292  3805.643  0.013018365
  9  27689.70  0.9503802  3556.068  0.012335645

RMSE was used to select the optimal model using  the smallest value.
The final value used for the model was k = 3.
```

Fig.34 Summary of k-nearest neighbor training model

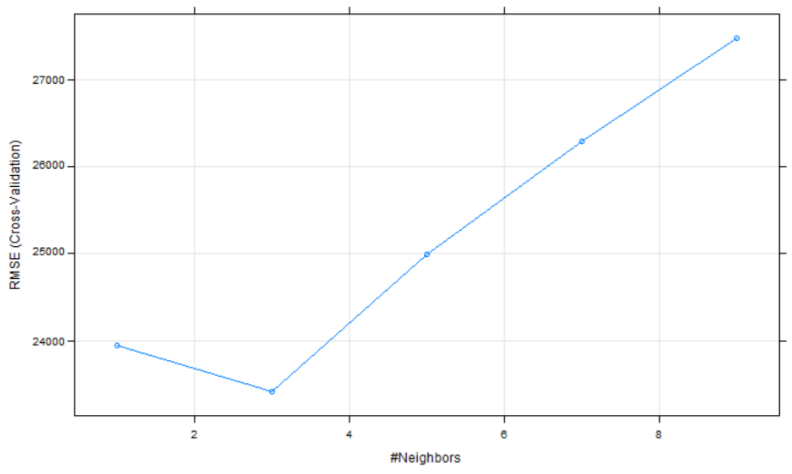


Fig.35: The RMSE cross-validation profile for KNN model

Then we build the KNN model with k =3, and we train the model on the full training dataset which has 626944 observations. And we test the model on the test dataset which has 156735 observations. Computational time is recorded.

```
## K Nearest neighbors regression with k=3

      ptm <- proc.time()

      library(caret)

      knn <- knnreg(trainPredictors_filter1, y=trainResponse$trainResponse,k=3)

      knn.pred <- predict (knn, testPredictors_filter)

      postResample (knn.pred, testResponse$testResponse)

      knn.time <- proc.time()-ptm
```

As mentioned in chapter 2, KNN is time-costly when the data size is large, while k-d tree overcomes the problem. Thus we also conduct an experiment on k-d tree model with same inputs and same k value (k=3).

```
## K-d tree regression with k=3

        library(FNN)

        ptm <- proc.time()

        kd_tree <- knn.reg(trainPredictors_filter1,test=testPredictors_filter,

                                  y=trainResponse$trainResponse,k=3,algorithm=c("kd_tree"))

        postResample(kd_tree$pred,testResponse$testResponse)

        kd_tree.time <- proc.time()-ptm
```

The performance estimates of KNN model and k-d tree model are listed in Table 4. The R-squared and RMSE parameters estimates of two models are quite close, which shows that the predictive performance of KNN model and k-d tree model have no much difference. However, the computation time of two models differs a lot. The total computation time of KNN model with given inputs is 1522.34 seconds, while the computation time of k-d tree model with same inputs is just 3.38 seconds. The experiments show that k-dimensional tree produces accurate predictions as well as time efficient.

Table 4 Performance estimates of KNN and k-d tree models with k=3

Model	R-squared	RMSE	Computation time (seconds)
k-nearest neighbor (k=3)	0.992687	10742.87	1522.34
k-d tree (k=3)	0.992784	10671.18	3.38

Fig.36 visualizes the relationship between values predicted by k-dimensional tree model and the actual observed values. The majority of the points in the figure fall into a line which shows the model fits the data very well. Fig.37 visualizes the predicted data distribution of k-d tree model. Compared with Fig.21 which shows the actual observed data distribution, the predicted data distribution of k-d tree model is very similar to the observed data distribution. The visualizations of predicted values once again shows that the k-d tree model produces accurate predictions.

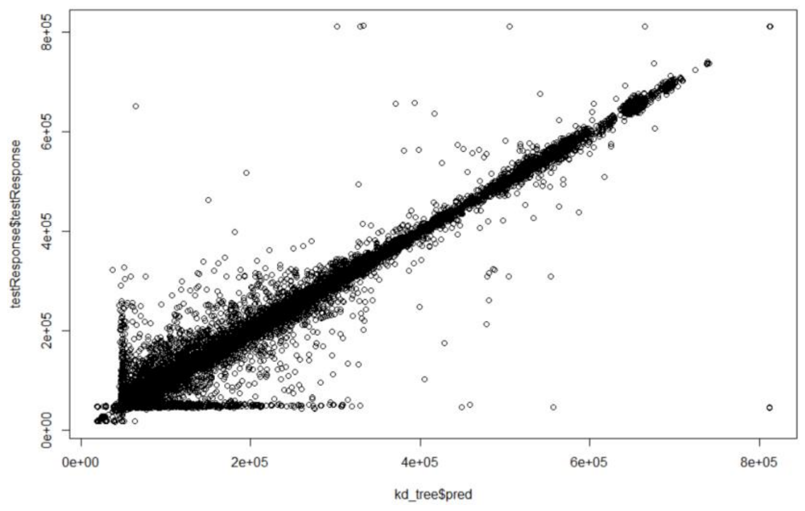


Fig.36 Predicted values of k-d tree model vs actual values.

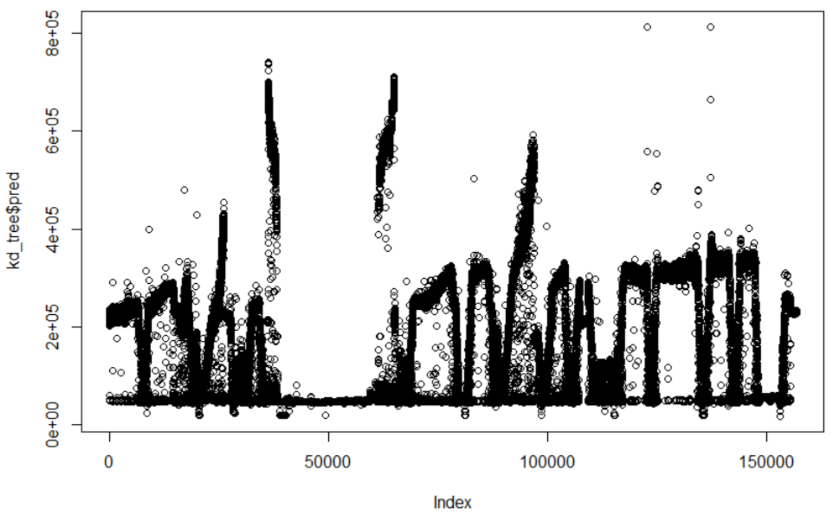


Fig.37 Predicted data distribution of k-d tree model.

Experiment results summary

Table 5 lists the summary of experiment results, from which we can get the performance estimates and computation time of each model. From the table, we can see that neural network with backpropagation algorithm, k-nearest neighbors and k-dimensional tree models produce decent predictive performance with R-squared estimates large than 0.95. Considering the computation time, k-dimensional tree with 3.38 seconds computation time proves to be the most efficient model. Thus we can conclude that nearest neighbors with k-dimensional tree algorithm is the model which best fits our data.

Table 5. Summary of experiment results

Model	**R-squared**	**RMSE**	**Computation time (seconds)**
Ordinary linear regression	0.7365	64197.33	22.43
Partial least squares regression	0.7570	61894.00	67.91
Averaged Neural network (size = 15)	0.9215	37602.73	2793.106
DBN with stacked RBMs (2 RBMs)	0.8592	47989.46	6028.19
NN with BP algorithm	0.9586	28077.00	3731.42
K nearest neighbors	0.9927	10742.87	1522.34
K-dimensional tree	0.9928	10671.18	3.38

Chapter IV Conclusion

Our work intends to apply machine learning algorithm for data exploratory analysis and building predictive models on real data. To achieve this goal, we first conducted a research on various machine learning algorithms, analyzed the advantages and limitations of each algorithm. According to the survey, we selected linear regression, neural network, and k-nearest neighbor algorithms which were considered as promising models to build the models. Data preprocessing techniques and methods to avoid over-fitting problems were taken into account to help build the models correctly. Following a certain methodology, we conducted experiments on the selected models and their variants. The methodology we performed to build our models can be easily extended to build other predictive models.

To evaluate the predictive performance of model, this work applied a combination of parameter estimates and data visualization. Besides, the computation time of each model is also presented in this paper in order to select an efficient and practical method.

The experiment results shows that our initial hypothesis is true and nearest neighbors model with k-dimensional tree algorithm is considered as the most efficient model that produces accurate predictions in our case. This paper explains the whole process to building the model to perform data analysis on raw data, thus, it can be easily put into practice.

It would be interesting to apply this model on new and bigger dataset to examine the performance of this model on big data. It would be also interesting to extend this model in a way to apply to the data with different structure.

Besides, further developments can be done to build neural network model with more different structures, such as combine deep belief network with backpropagation algorithm. And how to reduce computation time of neural network is also a topic worthy of study.

Reference

1. Intelligent Solutions, Inc. “Oilfield data mining”, 2011. Retrieved from http://www.intelligentsolutionsinc.com/PDFs/OilFieldDataMining.pdf.
2. Sondalini, Mike, and Howard Witt. "What is equipment reliability and how do you get it?" (2004), *http://www.lifetime-reliability.com/*
3. McCullagh, Peter. "John Wilder Tukey 16 June 1915--26 July 2000."Biographical Memoirs of Fellows of the Royal Society (2003): 537-555.
4. Exploratory data analysis. Available from: https://en.wikipedia.org/wiki/Exploratory_data_analysis.
5. Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
6. Domingos, Pedro. "A few useful things to know about machine learning."*Communications of the ACM* 55.10 (2012): 78-87.
7. Jordan, Michael I., and D. E. Rumelhart. "Internal world models and supervised learning." *Machine Learning: Proceedings of Eighth Internatinoal Workshop*. 2014.
8. Alpaydin, Ethem. *Introduction to machine learning*. MIT press, 2014.
9. Kuhn, Max, and Kjell Johnson. *Applied predictive modeling*. New York: Springer, 2013.
10. Quilumba, Franklin L., et al. "An overview of AMI data preprocessing to enhance the performance of load forecasting." *Industry Applications Society Annual Meeting, 2014 IEEE*. IEEE, 2014.
11. Little, Roderick JA, and Donald B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 2014.
12. Van Buuren, Stef. *Flexible imputation of missing data*. CRC press, 2012.
13. Goldstein, H. "A general procedure for handling missing data." (2013).
14. Kabacoff, Robert. *R in Action*. Manning Publications Co., 2011.
15. Enders, Craig K., Amanda J. Fairchild, and David P. MacKinnon. "A Bayesian approach for estimating mediation effects with missing data." *Multivariate behavioral research* 48.3 (2013): 340-369.
16. Serneels, Sven, Evert De Nolf, and Pierre J. Van Espen. "Spatial sign preprocessing: a simple way to impart moderate robustness to multivariate estimators." *Journal of Chemical Information and Modeling* 46.3 (2006): 1402-1409.
17. Bro, Rasmus, and Age K. Smilde. "Centering and scaling in component analysis." *Journal of Chemometrics* 17.1 (2003): 16-33.

18. Draper, Norman Richard, Harry Smith, and Elizabeth Pownell. *Applied regression analysis*. Vol. 3. New York: Wiley, 1966.
19. Osbourne, Jason W. "Notes on the Use of Data Transformation." *Practical Assessment, Research & Evaluation* 8.6 (2002): n6.
20. Belsley, David A., Edwin Kuh, and Roy E. Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*. Vol. 571. John Wiley & Sons, 2005.
21. Abdi, Hervé, and Lynne J. Williams. "Principal component analysis." *Wiley Interdisciplinary Reviews: Computational Statistics* 2.4 (2010): 433-459.
22. Seber, George AF, and Alan J. Lee. *Linear regression analysis*. Vol. 936. John Wiley & Sons, 2012.
23. Montgomery, Douglas C., Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to linear regression analysis*. Vol. 821. John Wiley & Sons, 2012.
24. Harrell, Frank E. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Springer Science & Business Media, 2013.
25. Tibshirani, Robert. "Regression shrinkage and selection via the lasso: a retrospective." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.3 (2011): 273-282.
26. Gallant. "Nonlinear regression." *The American Statistician* 29.2 (1975): 73-81.
27. Bishop, Christopher M. *Neural networks for pattern recognition*. Oxford university press, 1995.
28. Linggard, Robert, D. J. Myers, and C. Nightingale, eds. *Neural networks for vision, speech and natural language*. Vol. 1. Springer Science & Business Media, 2012.
29. Kruse, Rudolf, et al. "Multi-Layer Perceptrons." *Computational Intelligence*. Springer London, 2013. 47-81.
30. Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *Cognitive modeling* 5 (1988).
31. Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. No. ICS-8506. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, 1985.
32. Smolensky, Paul. "Information processing in dynamical systems: Foundations of harmony theory." (1986): 194.
33. Hinton, Geoffrey, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.
34. Hinton, Geoffrey E. "A practical guide to training restricted boltzmann machines." *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg, 2012. 599-619.

35. Fischer, Asja, and Christian Igel. "Training restricted Boltzmann machines: an introduction." *Pattern Recognition* 47.1 (2014): 25-39.
36. Cueto, María Angélica, Jason Morton, and Bernd Sturmfels. "Geometry of the restricted Boltzmann machine." *Algebraic Methods in Statistics and Probability,(eds. M. Viana and H. Wynn), AMS, Contemporary Mathematics* 516 (2010): 135-153.
37. Cai, Xianggao, Su Hu, and Xiaola Lin. "Feature extraction using restricted Boltzmann machine for stock price prediction." *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*. Vol. 3. IEEE, 2012.
38. Salakhutdinov, Ruslan. *Learning deep generative models*. Diss. University of Toronto, 2009.
39. Weston, Jason. "Support Vector Machine." *Tutorial, http://www. cs. columbia. edu/~ kathy/cs4701/documents/jason_svm_tutorial. pdf, accessed* 10 (2014).
40. Pradhan, Biswajeet. "A comparative study on the predictive ability of the decision tree, support vector machine and neuro-fuzzy models in landslide susceptibility mapping using GIS." *Computers & Geosciences* 51 (2013): 350-365.
41. Jiang, Shengyi, et al. "An improved K-nearest-neighbor algorithm for text categorization." *Expert Systems with Applications* 39.1 (2012): 1503-1509.
42. Potashov, M., et al. "Direct distance measurements to SN 2009ip." *Monthly Notices of the Royal Astronomical Society: Letters* (2013): slt016.
43. Shekhar, Aishwarya, Trapti Sharma, and Devesh Kumar Srivastava. "Effectiveness of Kd-Tree in Ray Tracing of Dynamic Point Clouds."*Networking and Communication Engineering* 7.4 (2015): 133-137.
44. Bianco, Vincenzo, Oronzio Manca, and Sergio Nardini. "Electricity consumption forecasting in Italy using linear regression models." *Energy* 34.9 (2009): 1413-1421.
45. Cheh, John J., Randy S. Weinberg, and Ken C. Yook. "An application of an artificial neural network investment system to predict takeover targets." *Journal of Applied Business Research (JABR)* 15.4 (2013): 33-46.
46. Foley, Aoife M., et al. "Current methods and advances in forecasting of wind power generation." *Renewable Energy* 37.1 (2012): 1-8.
47. Xu, Qing, et al. "Calibration of area based diameter distribution with individual tree based diameter estimates using airborne laser scanning." *ISPRS Journal of Photogrammetry and Remote Sensing* 93 (2014): 65-75.
48. Alkhatib, Khalid, et al. "Stock Price Prediction Using K-Nearest Neighbor (kNN) Algorithm." *International Journal of Business, Humanities and Technology* 3.3 (2013): 32-44.

49. Business Intelligence, Not all lifts are born equal (2010). Retrieved from http://www.business-insight.com/html/intelligence/bi_overfitting.html
50. Willett, Peter. "Similarity-based data mining in files of two-dimensional chemical structures using fingerprint measures of molecular resemblance." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.3 (2011): 241-251.
51. Weber, N. C. "On resampling techniques for regression models." *Statistics & probability letters* 2.5 (1984): 275-278.
52. Priddy, Kevin L., and Paul E. Keller. *Artificial neural networks: an introduction*. Vol. 68. SPIE Press, 2005.
53. Smith, Steven W. "The scientist and engineer's guide to digital signal processing." (1997).
54. Kuhn, Max. "Building predictive models in R using the caret package." *Journal of Statistical Software* 28.5 (2008): 1-26.
55. Xiao Rong, Package "deepnet" (2015), http://cran.r-project.org/web/packages/deepnet/deepnet.pdf
56. Christoph Bergmeir, José M. Benítez, "Package RSNNS" (2015), http://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf
57. Sunil Arya, David Mount (for ANN), Samuel E. Kemp, Gregory Jefferis, "Package RANN" (2015), http://cran.r-project.org/web/packages/RANN/RANN.pdf

APPENDIX- Code

```
## Ordinary linear regression

        library(caret)

        OLS_Fit <- train(x=trainPredictors_filter1,y=as.matrix(trainResponse),

                              method="lm", trControl=ctrl)

## Make prediction on new data

        OLS_Pred <- predict(OLS_Fit, testPredictors_filter)

        postResample(OLS_Pred, testResponse$testResponse)

        plot(OLS_Pred,as.matrix(testResponse), xlab = "Predicted", ylab = "Observed")

## Visulization of Predicted vs Observed

        xyplot(as.matrix(testResponse) ~ predict(OLS_Fit,testPredictors_filter),type =
        c("p", "g"), xlab = "Predicted", ylab = "Observed")

## Visulization of Predicted vs Residuals

        xyplot(resid(OLS_Fit) ~ predict(OLS_Fit),type = c("p", "g"), xlab = "Predicted",
        ylab = "Residuals")

## Visulization of the Predicted values

        plot(OLS_Pred,xlab = "time", ylab= "OLS_Predicted")

## Visulization of Observed values

        plot(testResponse$testResponse,xlab = "time", ylab= "Observed")
```

```
Code:

## Partial Least Squares Regression

        library(caret)

        plsTune <- train(trainPredictor_scale, trainResponse$trainResponse,

        method="pls", tuneLength = 15, trControl = ctrl)

## Visualization of number of components vs RMES

        plot(plsTune)

## Visualization of importance of predictors in predicting response

        plsImpGrant <- varImp(plsTune,scale=F)

        plot(plsImpGrant)

## Create a PLSR model with 13 components, test the model on new data,

        library(pls)

        plsFit <- plsr(trainResponse$trainResponse~.,
        ncomp=13,data=trainPredictor_scale,validation="CV")

        PLS_Pred <- pls:::predict.mvr(plsFit, newdata = testPredictors_scale,

        ncomp=13,type="response")

        postResample(PLS_Pred,as.matrix(testResponse))

##Visulization of Predicted vs Observed

        xyplot(as.matrix(testResponse) ~ predict(plsFit,newdata= testPredictors_scale),
        type = c("p", "g"), xlab = "Predicted", ylab = "Observed")

##Visulization of Predicted vs Residuals

        xyplot(resid(plsFit) ~ predict(plsFit), type = c("p", "g"),

        xlab = "Predicted", ylab = "Residuals")

## Visulization of the Predicted values

        plot(PLS_Pred,xlab = "time", ylab= "PLS_Predicted")
```

```
Code:

## Train neural network with method "avgnet"

        library(doParallel);

        cl <- makeCluster(detectCores())

        registerDoParallel(cl)

        ptm <- proc.time()

        library(caret)

        nnetGrid_2 <- expand.grid(.decay = c(0,0.01,0.1), .size = c(1,3,5,7,9,11,13,15), .bag = F)

        avgnnet_2 <- train(nn_trainPredictors_filter,nn_trainResponse$nn_trainResponse,

                        method="avNNet", tuneGrid=nnetGrid_2, trControl = trainControl(method =
                        "cv",number = 10), linout = T, trace= F, MaxNWts = 1000,maxit = 500)

        stopCluster(cl)

        avgnnet.time2 <- proc.time()-ptm

## Test the model on new data

        avgnnet_Pred <- predict(avgnnet_2, nn_testPredictors_filter)

        postResample(avgnnet_Pred, nn_testResponse$nn_testResponse)

        plot(avgnnet_Pred, nn_testResponse$nn_testResponse, xlab=
        "Predicted",ylab="Observed")
```

Code:

```
## Model: Deep Belief Network with Restricted Boltzmann Machine

## Scale the input data to interval [0, 1].

        training_normalized <- apply(nn_trainPredictors, MARGIN = 2, FUN = function(x)(x -
        min(x))/(max(x)- min(x)))

        testset_normalized <- apply(nn_testPredictors,MARGIN = 2, FUN = function(x)(x -
        min(x))/(max(x)- min(x)))

        trainX62_normalized <- apply(train_X62, MARGIN = 2, FUN = function(x)(x -
        min(x))/(max(x)- min(x)))

        testX62_normalized <- apply(test_X62, MARGIN = 2, FUN = function(x)(x -
        min(x))/(max(x)- min(x)))

## Remove the highly correlated predictors

        nn_train_filter <- filter1(x = training_normalized);

        nn_test_filter <- filter1(x = testset_normalized);

## Train DBN with RBM with 1 layer of 15 hidden units.

        library(deepnet);library(caret)

        nn_train <- foreach(i = 1:5, .packages = c('deepnet','caret')) %dopar%

        {

        dbnFit <- dbn.dnn.train(as.matrix(nn_train_filter),as.matrix(trainX62_normalized), hidden =
c(15), numepochs = 500)

        dbn_Pred<- nn.predict(dbnFit,nn_test_filter)

        dbn_Pred<- dbn_Pred*(max(train_X62)-min(train_X62))+min(train_X62)

        list <- postResample(dbn_Pred,test_X62$nn_testResponse)

        RMSE[i] <- list[1];

        Rsquared[i] <- list[2];

        }

        plot(dbn_Pred)

        plot(dbn_Pred, test_X62$nn_testResponse)
```

Code:

```
### Neural network with Backpropagation algorithm

    library(RSNNS)

    library(doParallel);library(caret);

    cl <- makeCluster(detectCores())

    registerDoParallel(1)

    ptm <- proc.time()

## Train the model with 1 hidden layer of 15 neurons

    mlpTune_1 <- mlp(nn_train_filter,as.matrix(trainX62_normalized),size = c(15),

                        linOut = T,maxit = 40)

## Test the regression fit on new data

    mlp_pred <- predict(mlpTune_1, nn_test_filter)

    mlp_pred <- mlp_pred*(max(train_X62)-min(train_X62))+min(train_X62)

    postResample(mlp_pred,test_X62)

    plot(mlp_pred, test_X62$nn_testResponse,xlab="NN_BP_Predicted", ylab="Observed")

    stopCluster(cl)

    nn_mlp.time <-proc.time()-ptm

## Visualization of the predicted values

    plot(mlp_pred)

## Visualization of RMSE of each iteration

    plotIterativeError(mlpTune_1)

## Visualization of the model structure

    Library(devtools)

    source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff0
    44412703516c34f1a4684a5/nnet_plot_update.r')

    plot.nnet(mlpTune_1)
```

```
## K Nearest Neighbor Regression

    library(doParallel);library(caret)

    cl <- makeCluster(detectCores()-1)

    registerDoParallel(cl)

    ptm <- proc.time()

    knnFit1 <- train(sample_Predictors,sample_Response$Response,method="knn",

                tuneGrid=data.frame(.k=c(1,3,5,7,9)),trControl=trainControl(method="cv"))

    predicted_1 <- predict(knnFit1,newdata = testPredictors_filter)

    postResample(pred = predicted_1, obs = testResponse$testResponse)

    knnFit1.time <- (proc.time()-ptm)/60

    stopCluster(cl)

## K Nearest neighbors regression with k=3

    ptm <- proc.time()

    library(caret)

    knn <- knnreg(trainPredictors_filter1, y=trainResponse$trainResponse,k=3)

    knn.pred <- predict(knn,testPredictors_filter)

    postResample(knn.pred,testResponse$testResponse)

    knn.time <- proc.time()-ptm

## K-d tree regression with k=3

    library(FNN)

    ptm <- proc.time()

    kd_tree <- knn.reg(trainPredictors_filter1,test=testPredictors_filter,

                  y=trainResponse$trainResponse,k=3,algorithm=c("kd_tree"))

    postResample(kd_tree$pred,testResponse$testResponse)

    kd_tree.time <- proc.time()-ptm

    plot(kd_tree$pred,testResponse$testResponse)
```