A Computational Study on Valuation of the Variable Annuity

by

Yinuo Liu

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof. Yuying Li

Waterloo, Ontario, Canada, 2015

© Yinuo Liu 2015

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

Traditional Monte Carlo method for calculating the fair market value of a large portfolio of variable annuity contracts can be slow in practice. Gan [4] has proposed a new approach that combines the data clustering method and the ordinary Kriging method to address this issue.

In this computational study we intent to investigate the possibility of using alternative methods for the data clustering method and the ordinary Kriging method. In our experiment we use random sampling instead of data clustering method, and support vector regression and boosted tree for regression as the alternatives for the ordinary Kriging method. Our experiment results show that the data clustering method does not improve the all over accuracy and the support vector regression with small set of training data points has the best performance.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Yuying Li for the continuous support of my master study and related research, her guidance helped me in all the time of research and writing of this research paper.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A variable annuity (VA) is a contract sold by an insurance company. The contract provides the holder with future payments based on the performance of the contract's underlying securities, usually mutual funds. The VA contracts always guarantee a minimum payment, such as guaranteed minimum death benefit (GMDB) or guaranteed minimum withdraw benefit (GMWB) etc..

In a real world situation, a insurance company usually hold a large portfolio of VA contracts. Therefore how to value this large portfolio of VA contracts in an efficient way becomes a critical problem to the company. In practice, the Monte Carlo method is used to calculate values of VA contracts because often no closed-form formulas are available. When pricing a VA contract the Monte Carlo method requires the generation of a set of risk neutral paths and compute the cashflow of that contract along each path. However, in order to have more accurate results many risk neutral paths are required to price one VA contract, consequently it can be computationally expensive for a large portfolio of VA contracts.

To address this problem, Gan [4] has proposed a novel approach based on data clustering and machine learning. The outline of this approach is to first apply data clustering to select a few representative contracts out of a large portfolio of VA contracts, then use the Monte Carlo method to price those representative VA contracts. Secondly, those VA contracts that have already been valued serve as the training set of a machine learning method called ordinary Kriging method, or Gaussian process regression. Gan [4] shows that this new approach can achieve similar results as the Monte Carlo method does, but in significantly less amount of time.

The objective of this research project is to extend the work of Gan [4]. It aims to explore the possibility of using alternative methods for selecting training samples and regression that together can work better than the data clustering method and ordinary Kriging method in term of accuracy and speed. To achieve this goal, first we replace ordinary Kriging method with support vector regression and boosted tree for regression in the regression step, compare their accuracy and speed; secondly, we use random sampling instead of data clustering method to obtain a small set of contracts that are used as training data points in the regression step. We assess the effectiveness of data clustering method by comparing the accuracy of each regression method using these two types of training.

We organize this computational study in the following way: Chapter 2 provides details of the approach used by Gan [4], which is the main reference of this study. In Chapter 3 we provide the background knowledge of the support vector regression and boosted tree model for regression. In Chapter 4 we present our numerical experiment steps and results on a synthetic VA portfolio and in Chapter 5 we draw conclusions based on the results in Chapter 4.

Chapter 2

Valuation of Large Portfolios of VA Contracts by Data Clustering and Machine Learning

When pricing a VA contract the Monte Carlo method requires the generation of a set of risk neutral paths and compute the cashflow of that contract along each path. Typically one risk neutral path consists of t' time stamps, a VA contract will evolve along this path until it reaches its time of maturity T. Suppose we generate N risk neutral paths, and a contract's value at maturity along the ith path is V_i, then the final value of that contract is computed as $\frac{\sum_{i=1}^{N} V_i}{N}$. The problem arises as we need to generate a large set of risk neutral paths in order to have more accurate result for each VA contract in a large portfolio. In this case, using Monte Carlo simulation to price all contracts in that portfolio can be quite slow.

Gan [4] proposes a new approach based on data clustering and machine learning. It first applies data clustering to select a few representative contracts out of a large portfolio of VA contracts, then uses the Monte Carlo method to price those representative VA contracts. Secondly, those VA contracts that have already been valued serve as the training set of a machine learning method called ordinary Kriging method so that the remaining of VA contracts in the portfolio can be priced. In the rest part of this chapter we will provide a detailed description of the data clustering method and ordinary Kriging method.

2.1 A Data Clustering Method

The k-means algorithm is a commonly used unsupervised machine learning algorithm that clusters similar data points together. However, k-means algorithm assumes numeric values are given to calculate the distances between data points, in our case a VA contract has both numeric and categorical features such that k-means cannot be applied directly. To address this problem Huang [6] proposes a new algorithm called the k-prototype algorithm which extends the k-means algorithm and can be used to cluster objects that have both numeric and categorical features.

To use the k-prototype algorithm we need to define the distance function on two VA contracts. Suppose one VA contracts has d features, and the first d_1 are numeric and the rest $d-d_1$ features are categorical. Then distance between two contracts $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as:

$$D(\boldsymbol{x},\boldsymbol{y},\lambda) = \sqrt{\sum_{h=1}^{d_1}(\boldsymbol{x}_h-\boldsymbol{y}_h)^2 + \lambda \sum_{h=d_1+1}^{d} \delta(\boldsymbol{x}_h,\boldsymbol{y}_h)} \tag{2.1}$$

where $\boldsymbol{x}_h$ and $\boldsymbol{y}_h$ are the hth feature of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. Here λ is a parameter used to maintain the balance between numeric and categorical features in distance computation. $\delta(\cdot,\cdot)$ is defined as:

$$\delta(\boldsymbol{x}_h,\boldsymbol{y}_h) = \begin{cases} 0 & \boldsymbol{x}_h = \boldsymbol{y}_h \\ 1 & \boldsymbol{x}_h \neq \boldsymbol{y}_h \end{cases} \tag{2.2}$$

In order to avoid the situation that some numeric features dominate the distance calculation, we need to normalize every numeric features such they all have mean 0 and standard deviation 1.

With distance function defined in Eq.(2.1), the k-prototype algorithm seeks to partition the n data points into k ($k \leq n$) clusters $\mathbf{C} = \{C_1, C_2, \ldots, C_k\}$ so as to minimize the within-cluster sum of squared distances. It is equivalent to find:

$$\arg\min_{\mathbf{C}} \sum_{j=1}^{k} \sum_{\boldsymbol{x}\in C_j} D^2(\boldsymbol{x},\boldsymbol{\mu_j},\lambda) \tag{2.3}$$

where $\boldsymbol{\mu_j}$ is the center or prototype of cluster C_j.

Like the k-means algorithm, the k-prototype algorithm works in an iterative fashion to solve the minimization problem (2.3). It mainly consists of two steps: the assignment step and the update step, and the algorithm proceeds by alternating these two steps. We illustrate the details of the k-prototype algorithm used in our study in Algorithm 1.

Algorithm 1: k-prototype algorithm

```
1 Initialize the $k$ cluster centers $\boldsymbol{\mu_1}^{(0)}, \boldsymbol{\mu_2}^{(0)}, \ldots, \boldsymbol{\mu_k}^{(0)}$ by randomly selecting $k$ distinct
  instances for the dataset $\mathbf{X}$;
2 Assignment Step: Assign each observation to the cluster whose center is the
  closest to that observation. Mathematically, it is equivalent to partition the dataset
  $\mathbf{X}$ in following way:
      $C_j^{(t)} = \{\boldsymbol{x} : D^2(\boldsymbol{x}, \boldsymbol{\mu_j}, \lambda) \leq D^2(\boldsymbol{x}, \boldsymbol{\mu_l}, \lambda) \ \forall l, 1 \leq l \leq k\}$   (2.4)
  where $j = 1, 2, \ldots, k$ and $D(\cdot, \cdot, \lambda)$ is defined in Eq.(2.1);
3 Update Step: Update the cluster centers in following way:
      $\mu_{jh}^{(t+1)} = \frac{1}{|C_j^{(t)}|} \sum_{\boldsymbol{x} \in C_j^{(t)}} \boldsymbol{x_h}, \quad h = 1, 2, \ldots, d_1$
      $\mu_{jh}^{(t+1)} = \text{mode}_h(C_j^{(t)}) \quad h = d_1 + 1, \ldots, d$   (2.5)
  where $\text{mode}_h(C_j^{(t)})$ is the most frequent categorical value of the $h^{th}$ feature in cluster
  $C_j$ at the $t^{th}$ iteration;
4 Repeat the assignment step and the update step until the assignments no longer
  change.
```

Note that the k-prototype algorithm does not necessarily solve (2.3), it only guarantees to find a local minimum as the objective function may have multiple local minima and different initializations of cluster centers could lead to different clustering results.

2.2 Ordinary Kriging Method

In this section, we illustrate mathematically how the ordinary Kriging method is applied in estimating the FMV of our synthetic portfolio of VA contracts, based on the representative contracts selected by the data clustering method.

In [4] Kriging estimate of the FMV of a VA contract $\boldsymbol{x}_i$ can be modelled as a linear combination of the FMV of representative contracts. More formally, let $\hat{y}_i$ be the Kriging estimate of the FMV of VA contract $\boldsymbol{x}_i$, and y_j denote the FMV (computed by the Monte Carlo method) of representative contract $\boldsymbol{z_j}, j = 1, \ldots, k$. Then $\hat{y}_i$ is computed by

$$\hat{y}_i = \sum_{j=1}^{k} w_{ij} y_j \tag{2.6}$$

where $w_{i1}, w_{i2}, \ldots, w_{ik}$ are Kriging weights which can be obtained, as shown in [4], by solving the following linear equation system:

$$\begin{bmatrix} V_{11} & \cdots & V_{1k} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ V_{k1} & \cdots & V_{kk} & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_{i1} \\ \vdots \\ w_{ik} \\ \mu_i \end{bmatrix} = \begin{bmatrix} \gamma_{i1} \\ \vdots \\ \gamma_{ik} \\ 1 \end{bmatrix} \tag{2.7}$$

where μ_i is used to guarantee $\sum_{l=1}^{k} w_{il} = 1$,

$$V_{rs} = \alpha + \exp(-\frac{3}{\beta} D(\boldsymbol{z_r}, \boldsymbol{z_s}, \lambda)), \;\; r, s = 1, 2, \ldots, k, \tag{2.8}$$

and

$$\gamma_{ij} = \alpha + \exp(-\frac{3}{\beta} D(\boldsymbol{x}_i, \boldsymbol{z_j}, \lambda)), \;\; j = 1, 2, \ldots, k. \tag{2.9}$$

Here $D(\cdot, \cdot, \lambda)$ is defined in Eq.(2.1) and α and β are another two parameters which need to be selected a prior. Note that numeric features of VA contracts are normalized using the z-score method [1].

It seems if we need to estimate a portfolio containing n VA contracts, n linear equation systems are required to be solved, which can be very time-consuming. To avoid this, we combine Eq.(2.6) and (2.7) together and reformulate our approach.

Let

$$\mathbf{A} = \begin{bmatrix} V_{11} & \cdots & V_{1k} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ V_{k1} & \cdots & V_{kk} & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix}, \; \mathbf{w_i} = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{ik} \end{bmatrix}, \; \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}, \; \Gamma_i = \begin{bmatrix} \gamma_{i1} \\ \vdots \\ \gamma_{ik} \\ 1 \end{bmatrix}$$

. Then Eq.(2.6) can be rewritten as

$$\hat{y}_i = \begin{bmatrix} \mathbf{w_i} \\ \mu_i \end{bmatrix}^T \begin{bmatrix} \mathbf{Y} \\ 0 \end{bmatrix} \tag{2.10}$$

[1] standard normalization

and Eq.(2.7) is equivalent to

$$\mathbf{A}\begin{bmatrix}\mathbf{w_i}\\ \mu_i\end{bmatrix} = \Gamma_i \tag{2.11}$$

which implies

$$\begin{bmatrix}\mathbf{w_i}\\ \mu_i\end{bmatrix} = \mathbf{A}^{-1}\Gamma_i \tag{2.12}$$

. Plugging Eq.(2.12) into Eq.(2.10) we have

$$\hat{y}_i = \Gamma_i^T(\mathbf{A}^{-1}\begin{bmatrix}\mathbf{Y}\\ 0\end{bmatrix}) \tag{2.13}$$

since $\mathbf{A}$ is a symmetric matrix. Because the set of representative contracts $\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k}$ and their respective FMV $y_1, y_2, \ldots, y_k$ are fixed, $\mathbf{F} = \mathbf{A}^{-1}\begin{bmatrix}\mathbf{Y}\\ 0\end{bmatrix}$ is a constant matrix of for all VA contracts whose FMV we are about to estimate. Let $\boldsymbol{\Gamma} = (\Gamma_1, \ldots, \Gamma_i, \ldots, \Gamma_n)$, then we have the following equation:

$$\begin{bmatrix}\hat{y}_1\\ \vdots\\ \hat{y}_i\\ \vdots\\ \hat{y}_n\end{bmatrix} = \Gamma^T\mathbf{F} \tag{2.14}$$

This allows us to only to solve one linear equation system to obtain the Kriging estimates of FMV of VA contracts $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$.

Chapter 3

Regression Methods

In this Chapter, we will provide an overview of theoretical details of two main regression methods involved in our computational study, one is support vector regression (SVR) and the other is boosted tree for regression. We organize this chapter in the following way: in Section 3.1 we first discuss the formulation of SVR and the kernel method that makes SVR non-linear; in Section 3.2 we present the detailed description of the boosted tree regression method.

3.1 SVR

3.1.1 Linear SVR

To present the detailed formulation of linear SVR, we mainly follow [5] and [8]. In general, suppose we have n training data points $\boldsymbol{x}_i, i = 1, 2, \ldots, n$, each training instance is of dimension d, $\boldsymbol{x}_i \in \mathbf{R^d}$, and has a quantitative response y_i , $y_i \in \mathbf{R}$. Given $(\boldsymbol{x}_i, y_i)$ $i = 1, 2, \ldots, n$, SVR aims to find a function $f(\boldsymbol{x})$ whose value at $\boldsymbol{x}_i$ is at most ϵ deviation from the response y_i. In a linear SVR, function $f(\boldsymbol{x})$ takes the following linear form:

$$f(\boldsymbol{x}) = \omega^T \boldsymbol{x} + b \tag{3.1}$$

where $\omega \in \mathbf{R^d}$ and $b \in \mathbf{R}$. Then the SVR is formulated as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||\omega||^2 \\ \text{subject to} \quad & y_i - \omega^T \boldsymbol{x}_i - b \leq \epsilon \\ & \omega^T \boldsymbol{x}_i + b - y_i \leq \epsilon, \quad i = 1, 2, \ldots, n \end{aligned} \tag{3.2}$$

where ϵ is the deviation, see e.g. [8]. Similar to the idea from the soft-margin support vector machine for classification which incorporate slack variables to deal with inseparable cases, variables ξ_i and ξ_i^* can be introduced to fix the infeasible constrains of the minimization problem (3.2), see e.g. [8]. After adding slack variables to (3.2) we obtain the primal problem (3.3):

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - \omega^T \boldsymbol{x}_i - b \leq \epsilon + \xi_i \\ & \omega^T \boldsymbol{x}_i + b - y_i \leq \epsilon + \xi_i^* \\ & \xi_i \geq 0, \xi_i^* \geq 0 \end{aligned} \tag{3.3}$$

where $C > 0$ is a cost parameter which determines the trade-off between the flatness of function f and the amount of deviations that are larger than ϵ.

Then Lagrangian $\mathbf{L}(\omega, b, \xi_i, \xi_i^*)$ for (3.3) is

$$\begin{aligned} \mathbf{L}(\omega, b, \xi_i, \xi_i^*) = & \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{n}\alpha_i(\epsilon + \xi_i - y_i + \omega^T \boldsymbol{x}_i + b) \\ & - \sum_{i=1}^{n}\alpha_i^*(\epsilon + \xi_i^* + y_i - \omega^T \boldsymbol{x}_i - b) \end{aligned} \tag{3.4}$$

where $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ are Lagrangian multipliers and these dual variables have to be non-negative, that is

$$\eta_i \geq 0, \eta_i^* \geq 0, \alpha_i \geq 0, \alpha_i^* \geq 0 \tag{3.5}$$

For the optimal solution, the following Karush-Kuhn-Tucker (KKT) conditions are satisfied:

$$\frac{\partial \mathbf{L}}{\partial b} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0 \tag{3.6}$$

$$\frac{\partial \mathbf{L}}{\partial \omega} = \omega - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x}_i = 0 \tag{3.7}$$

$$\frac{\partial \mathbf{L}}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \tag{3.8}$$

$$\frac{\partial \mathbf{L}}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \tag{3.9}$$

$$\alpha_i(\epsilon + \xi_i - y_i + \omega^T \boldsymbol{x}_i + b) = 0 \tag{3.10}$$

$$\alpha_i^*(\epsilon + \xi_i^* + y_i - \omega^T \boldsymbol{x}_i - b) = 0 \tag{3.11}$$

$$\eta_i \xi_i = 0 \tag{3.12}$$

$$\eta_i^* \xi_i^* = 0 \tag{3.13}$$

Substituting (3.6)-(3.9) into (3.4) yields the dual optimization problem (3.14),

$$\begin{aligned} \text{maximize} \quad & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\boldsymbol{x}_i^T \boldsymbol{x_j} \\ & -\epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) = 0 \\ & 0 \le \alpha_i \le C \\ & 0 \le \alpha_i^* \le C \end{aligned} \tag{3.14}$$

From (3.6) we have $\omega = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x}_i$, this leads to

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x}_i^T \boldsymbol{x} + b \tag{3.15}$$

and the constant offset b can be obtained from the interior point optimization process, see details in [8]. (3.15) shows $f(\boldsymbol{x})$ depends on the training data points through the inner products $\langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$. Equations (3.10)-(3.13) are the complementary slackness conditions.

From (3.10)-(3.11) we know that $\alpha_i \alpha_i^* = 0$, which means dual variables α_i and α_i^* cannot be both positive simultaneously. Since $\eta_i = C - \alpha_i$ and $\eta_i^* = C - \alpha_i^*$, then (3.12) and (3.13) are equivalent to:

$$(C - \alpha_i)\xi_i = 0 \tag{3.16}$$

$$(C - \alpha_i^*)\xi_i^* = 0 \tag{3.17}$$

(3.16) and (3.17) imply that only samples $\boldsymbol{x}_i$ whose values $f(\boldsymbol{x}_i)$ are more than ϵ from their responses y_i will have the corresponding $\alpha_i = C$ or $\alpha_i^* = C$.

3.1.2 Nonlinear SVR

To make the SVR nonlinear, we follow the formulation in [8]. First we introduce a function $\Phi : \mathbf{R^d} \rightarrow \mathcal{F}$ that maps training data points $\boldsymbol{x}_i$ $\boldsymbol{x}_i \in \mathbf{R^d}$ into some feature space $\mathcal{F}$. With the help of the kernel trick, in $\mathcal{F}$ the inner product between two instances $\boldsymbol{x}_i$ and $\boldsymbol{x_j}$ can be computed by

$$K(\boldsymbol{x}_i, \boldsymbol{x_j}) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x_j}) \rangle \tag{3.18}$$

where $K(\cdot, \cdot)$ can be a known kernel function. This means without knowing the mapping function $\Phi(\cdot)$ explicitly, we can directly use $K(\boldsymbol{x}_i, \boldsymbol{x'_i})$ to compute the inner product of $\boldsymbol{x}_i$ and $\boldsymbol{x'_i}$ in the feature space $\mathcal{F}$.

With (3.18) the dual problem for nonlinear SVR can be formulated as following:

$$\begin{aligned} \text{maximize} \quad & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\boldsymbol{x}_i, \boldsymbol{x_j}) \\ & -\epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C \\ & 0 \leq \alpha_i^* \leq C \end{aligned} \tag{3.19}$$

and (3.15) becomes

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{3.20}$$

3.2 Boosted Tree for Regression

In this section, we introduce the algorithm for the gradient boosted regression tree in [5] and stochastic gradient boosted regression tree in [2], which extends the work of [5].

3.2.1 Gradient Boosted Tree for Regression

Suppose we are given n training data points $\boldsymbol{x}_i, i = 1, 2, \ldots, n$, each training date point is of dimension d, $\boldsymbol{x}_i \in \mathbf{R}^{\mathbf{d}}$ and has a quantitative response y_i, $y_i \in \mathbf{R}$. Our target is to find an estimate or approximation $\hat{F}(\boldsymbol{x})$ of regression function $F^*(\boldsymbol{x})$, which minimizes the expected value of some loss function $\Psi(y, F(\boldsymbol{x}))$ for all possible $(\boldsymbol{x}, y)$ values. Mathematically, $F^*(\boldsymbol{x})$ is defined as

$$F^*(\boldsymbol{x}) = \arg\min_{F(\boldsymbol{x})} \mathbf{E}_{y,\boldsymbol{x}} \Psi(y, F(\boldsymbol{x})) \tag{3.21}$$

For the boosted tree regression model, $F(\boldsymbol{x})$ is restricted to be a sum of regression trees,

$$F_M(\boldsymbol{x}) = \sum_{m=0}^{M} T(\boldsymbol{x}, \Theta_m) \tag{3.22}$$

where M is the number of trees and $T(\boldsymbol{x}, \Theta_m)$, with parameters $\Theta_m = \{\gamma_{lm}, R_{lm}\}_1^L$, is a regression tree that partitions the data into L disjoint regions $\{R_{lm}\}_1^L$, and $\boldsymbol{x} \in R_{lm}$ implies $T(\boldsymbol{x}, \Theta_m) = \gamma_{lm}$. More formally,

$$T(\boldsymbol{x}, \Theta_m) = \sum_{l=l}^{L} \gamma_{lm} I(\boldsymbol{x} \in R_{lm}) \tag{3.23}$$

where $I(\cdot)$ is the indicator function. Based on the model F_{m-1} and the training data points $(\boldsymbol{x}, y_i)$, $i = 1, \ldots, n$, the parameters Θ_m for the m^{th} tree are obtained by solving the following minimization problem

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{n} \Psi(y_i, F_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i, \Theta_m)) \tag{3.24}$$

Finding the regions R_{lm} are generally difficult, however, if we are given the regions R_{lm}, the optimal values in each region can be obtained by solving (3.25).

$$\hat{\gamma}_{lm} = \arg\min_{\gamma_{lm}} \sum_{\boldsymbol{x}_i \in R_{lm}} \Psi(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma_{lm}) \tag{3.25}$$

To approximately find the regions R_{lm}, Friedman [2] borrows ideas from a numerical optimization method called steepest descent. The empirical loss at training data points $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, is

$$\Psi(F) = \sum_{i=1}^{n} \Psi(y_i, F(\boldsymbol{x}_i)). \tag{3.26}$$

Ignoring the constraint that F is a sum of trees (3.22), we now seek an approximating function $\hat{F}$ that minimizes the empirical loss at all training data points. The corresponding minimization problem is

$$\hat{F} = \arg\min_{F} \Psi(F) \tag{3.27}$$

where $F = \{F(\boldsymbol{x_1}), F(\boldsymbol{x_2}), \ldots, F(\boldsymbol{x_n})\}$, $F \in \mathbf{R^n}$.

Problem (3.27) can be then solved by the steepest descent method. Let $\mathbf{g_m}$ be the negative gradient of loss function $\Psi(F)$ at m^{th} iteration, the i^{th} component g_{im} of $\mathbf{g_m}$ is

$$g_{im} = -\left[\frac{\partial \Psi(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x}_i)=F_{m-1}(\boldsymbol{x}_i)} \tag{3.28}$$

Then F in the next iteration is computed by

$$F_m = F_{m-1} + \rho_m \mathbf{g_m} \tag{3.29}$$

where ρ_m is the step size obtained using a line search.

From (3.24) we see that given the current boosted tree model F_{m-1} and its fits $F_{m-1}(\boldsymbol{x}_i)$, the solution tree $T(\boldsymbol{x}, \hat{\Theta}_m)$ maximally reduces (3.24). On the other hand, the negative gradient $\mathbf{g_m}$ computed by (3.28) gives the direction for which (3.27) is most rapidly decreasing at $\{F_{m-1}(\boldsymbol{x_1}), F_{m-1}(\boldsymbol{x_2}), \ldots, F_{m-1}(\boldsymbol{x_n})\}$. Therefore, we can use the negative gradient values (3.28) to construct an approximate tree solution to (3.24). This can be done by fitting a regression tree $T(\boldsymbol{x}, \Theta)$ to the negative gradient values (3.27). The resulting regions $\tilde{R}_{lm}$ approximate the regions R_{lm} that solve (3.24). Once $\tilde{R}_{lm}$ are known, the optimal value $\hat{\gamma}_{lm}$ in each region can be computed by (3.25), and the current boosted tree model $F_{m-1}(\boldsymbol{x})$ is separately updated in each region

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu \sum_{l=1}^{L} \hat{\gamma}_{lm} I(\boldsymbol{x} \in \tilde{R}_{lm}). \tag{3.30}$$

where $0 \leq \nu \leq 1$ is the learning rate of this algorithm.

To summarize, we present the details of the algorithm for gradient boosted tree for regression in Algorithm 2.

Algorithm 2: Gradient Boosted Tree for Regression [5]

```
1  Initialize model with a constant value: $F_0(x) = \arg\min_\gamma \sum_{i=1}^n \Psi(y_i, \gamma)$ ;
2  for $m = 1$ to $M$ do
3      for $i = 1$ to $n$ do
4          $g_{im} = -\left[\frac{\partial \Psi(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x}_i)=F_{m-1}(\boldsymbol{x}_i)}$ ;
5      end
6      Fit a regression tree with the training set $\{(\boldsymbol{x}_i, g_{im})\}_{i=1}^n$ giving regions $\tilde{R}_{lm}$,
       $l = 1, 2, \dots, L$, in the leaf nodes;
7      for $l = 1, 2, \dots, L$ do
8          $\hat{\gamma}_{lm} = \arg\min_{\gamma_{lm}} \sum_{\boldsymbol{x}_i \in \tilde{R}_{lm}} \Psi(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma_{lm})$ ;
9      end
10     Update $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu \sum_{l=1}^L \hat{\gamma}_{lm} I(\boldsymbol{x} \in \tilde{R}_{lm})$ ;
11 end
```

In Chapter 4, the boosted tree model we use is the gradient boosted tree model. For simplicity, boosted tree model refers to the gradient boosted tree model in later chapters.

3.2.2 Stochastic Gradient Boosted Tree for Regression

In [2] Friedman proposed a modification to the gradient boosted tree algorithm by incorporating randomness into it. The new algorithm is called stochastic boosted tree algorithm because at each iteration, instead of using the full training set, it randomly selects a subsample out of the full training set to fit a regression tree (line 6) that gives leaf node regions $\tilde{R}_{lm}$, $l = 1, 2, \dots, L$, and computes the optimal value in each of these regions (line 7-9).

Specifically, let $\{\pi(i)\}_1^n$ be a random permutation of the set of integers $\{1, 2, \dots, n\}$, then a random subsample of size $\tilde{n} < n$ of training data points $\{\boldsymbol{x}_i, y_i\}_1^n$ is given by $\{\boldsymbol{x}_{\pi(i)}, y_{\pi(i)}\}_1^{\tilde{n}}$. Algorithm 3 shows the details of the algorithm for the stochastic gradient boosted tree for regression .

Friedman [2] suggests that by only using a random subsample of the training data points at each iteration can reduce the overall computation cost. While this approach can cause the variance of individual regression tree estimates to increase, it can reduce the variance of the final model (3.22) which averages the estimates of individual regression tree since using random subsample at each iteration can effectively reduce the correlation between the estimates at different iterations.

Algorithm 3: Stochastic Gradient Boosted Tree for Regression [2]

```
1  Initialize model with a constant value: $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} \Psi(y_i, \gamma)$ ;
2  for $m = 1$ to $M$ do
3      Generate a random sample $\{\boldsymbol{x}_{\pi(i)}, y_{\pi(i)}\}_1^{\tilde{n}}$ of size $\tilde{n}$ of the training data;
4      for $i = 1$ to $\tilde{n}$ do
5          $g_{\pi(i)m} = -\left[\frac{\partial \Psi(y_{\pi(i)}, F(\boldsymbol{x}_{\pi(i)}))}{\partial F(\boldsymbol{x}_{\pi(i)})}\right]_{F(\boldsymbol{x}_{\pi(i)}) = F_{m-1}(\boldsymbol{x}_{\pi(i)})}$ ;
6      end
7      Fit a regression tree with the training set $\{(\boldsymbol{x}_{\pi(i)}, g_{\pi(i)m})\}_{i=1}^{\tilde{n}}$ giving regions $\tilde{R}_{lm}$,
       $l = 1, 2, \ldots, L$, of the leaf nodes;
8      for $l = 1, 2, \ldots, L$ do
9          $\hat{\gamma}_{lm} = \arg\min_{\gamma_{lm}} \sum_{\boldsymbol{x}_{\pi(i)} \in \tilde{R}_{lm}} \Psi(y_{\pi(i)}, F_{m-1}(\boldsymbol{x}_{\pi(i)}) + \gamma_{lm})$ ;
10     end
11     Update $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu \sum_{l=1}^{L} \hat{\gamma}_{lm} I(\boldsymbol{x} \in \tilde{R}_{lm})$ ;
12 end
```

Chapter 4

Computational Investigation and Discussion of Results

In this chapter, we present the experiment results of applying three regression methods discussed in the previous chapter on a synthetic dataset. For our experiment, we generate a portfolio of 100,000 variable annuity contracts according to the specification in Table 4.1, using the synthetic VA contracts generation code from Gan [4]. Each VA contract has two categorical features (Guarantee type and Gender) and four numeric features, values of each feature are selected randomly from the ranges specified in Table 1.1 with equal probability.

Table 4.1: Variable annuity contract specification

Attribute	Values
Guarantee type	GMDB only, GMDB+GMWB
Gender	Male, Female
Age	$\mathbf{N} \cap [20, 60]$
Premium	$\mathbf{R} \cap [10000, 500000]$
Withdraw rate	0.04, 0.05, 0.06, 0.07, 0.08
Maturity	$\mathbf{N} \cap [10, 25]$

N and **R** represent the set of natural number and the set of real numbers, respectively.

After obtaining a synthetic dataset, we focus on estimating the fair market value (FMV) of each VA contract, which is the main task of this computational study. The steps of one run of our experiment can be described as follows:

(1) Generate a scenario file which contains a fixed number (e.g. 1000) of risk neural paths, then calculate the corresponding FMV of each VA contract by the Monte Carlo method based on these paths. The resulting FMV values will be used to benchmark subsequent comparisons.

(2) Select k representative VA contracts out of the portfolio using the data clustering method proposed by Gan [4], which implements the k-prototype algorithm [6].

(3) For the set of representative VA contracts, calculate their FMV by the Monte Carlo method under the same set of risk neutral paths in Step (1), the resulting set of VA contracts with FMV will be treated as training data points for our regression methods for model training, and the rest of the VA contracts in the portfolio form a testing set.

(4) Train a regression model on the training set obtained from step (3), perform parameter tuning when necessary, then apply the resulting model to the testing set.

(5) Calculate the accuracy and record the CPU time of the regression method in Step (4). In our experiment, we use four types of accuracy measurements, which are absolute portfolio difference (APD), relative portfolio difference (RPD), root mean squared error (RMSE) and mean absolute difference (MAD). Let $\hat{y_i}$ be the estimated value of contract i obtained from Step (4) and y_i be the benchmark value from Step (1), n denote the total number of contracts in the portfolio, then these four measurements can be defined in the following way:

$$\text{APD} = \sum_{i=1}^{n} |\hat{y_i} - y_i| \tag{4.1}$$

$$\text{RPD} = \frac{\sum_{i=1}^{n} |\hat{y_i} - y_i|}{\sum_{i=1}^{n} y_i} \times 100\% \tag{4.2}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y_i} - y_i)^2}{n}} \tag{4.3}$$

$$\text{MAD} = \frac{\sum_{i=1}^{n} |\hat{y_i} - y_i|}{n} \tag{4.4}$$

We are particularly interested in how k, the number of representative contracts affects the accuracy of each regression method, so we have tried four different values of k (100, 500, 1000, 2000). Moreover, due to randomness in the Monte Carlo method, multiple runs of our method are performed in order to see the variation in our regression estimates of the portfolio value. To achieve this goal, we repeat Step (1) to Step (5) 30 times. In generally, model training and testing for large k can be quite slow because model training always involve cross-validation which increases the computation a lot, this limits the number of repetitions.

4.1 Data Preparation

4.1.1 Monte Carlo Valuation

We follow the Monte Carlo simulation implemented by Gan [4] to price all 100,000 VA contracts in the portfolio so we can have the benchmark portfolio value. When pricing a single VA contract, Gan [4] uses 1000 risk neutral paths so we use the same number of paths as well. After every VA contract has been valued, we calculate the portfolio FMV and record the CPU time. Since the Monte Carlo method discussed here is not of major interest of this computational study, we choose to omit its details. A Full description of this computation can be found in Gan [4].

4.1.2 Representative Contracts Selection and Valuation

In this section, we focus on using the data clustering method in Gan [4] to construct a set of representative contracts, which will be used as training set for our regression methods.

Our approach consists of two steps. We first use the k-prototypes algorithm to select a set of k representative contracts out of the portfolio. Suppose that the resulting cluster centers are $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}$, then the representative VA contracts $\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k}$ are chosen as follows:

$$\boldsymbol{z_j} = \arg\min_{\boldsymbol{x} \in C_j} D(\boldsymbol{x}, \boldsymbol{\mu_j}, \lambda)$$

where the distance function $D(\cdot, \cdot, \lambda)$ is defined in Eq.(2.1) and C_j is the cluster of which $\boldsymbol{\mu_j}$ is the center. This means for a particular cluster of VA contracts, we choose the one that is closest to the cluster center as the representative of that cluster because $\boldsymbol{\mu_j}$ itself may not correspond to a specific contract in that cluster.

Secondly we use the Monte Carlo method to calculate the FMV of each representative contract under the same set of risk neutral scenarios described in Section 4.1.1. This is to guarantee a representative contract's FMV obtained here is the same as the one obtained from Section 4.1.1. Thus one representative contract consists of following fields:

(FMV, Guarantee Type, Gender, Age, Premium, Withdraw rate, Maturity)

Now this set of representative contracts with FMV forms a training set and will be used to train regression models in later steps.

4.1.3 Numerical Results

To account for randomness of the Monte Carlo simulation, we generate 30 different scenario files, each contains 1000 risk neutral paths. Then we repeat the processes described in both Section 4.1.1 and Section 4.1.2 using the these scenario files individually. As a result we have 30 training sets and their corresponding testing sets for each k. Table 4.2 shows the mean (μ), standard deviation (σ), as well as coefficient of variation (CV)[1] of the portfolio FMV. It also includes average CPU time used to price the portfolio. Table 4.3 displays the average CPU time of running the k-prototypes algorithm and Monte Carlo method for different k. Figure 4.1 shows the distribution of the portfolio FMV, the distribution is roughly symmetric.

Table 4.2: Portfolio FMV from Monte Carlo valuation

Measurement	Values
μ	1,440,356,432
σ	57,494,865
CV	4%
Average CPU time (seconds)	241.37

[1]ratio of the standard deviation σ to the mean μ

Table 4.3: Average CPU time of representative contracts selection and valuation

	k			
	100	500	1000	2000
k-prototypes	8.12	6.19	5.25	4.04
Monte Carlo	0.48	2.74	4.39	8.13
Total	8.50	8.93	9.64	12.17

The number are in seconds, average over 30 scenarios.

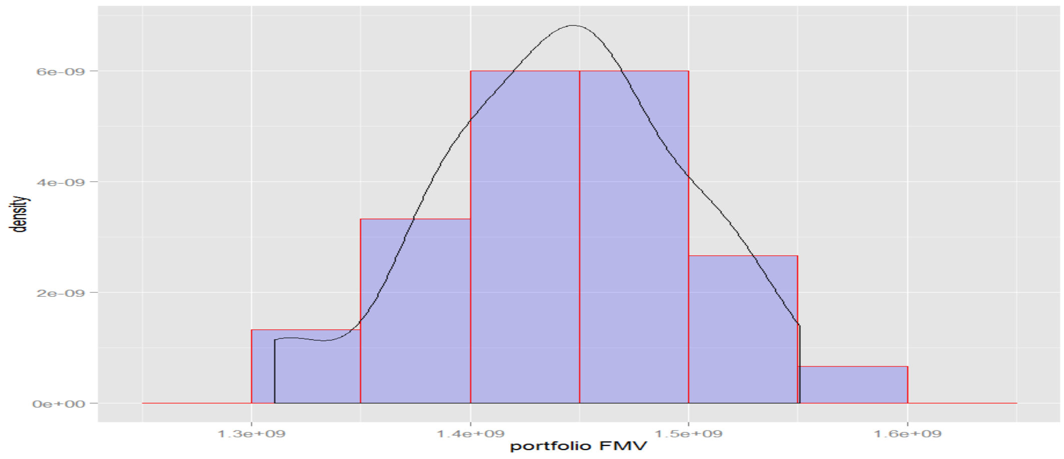

Figure 4.1: Histogram of portfolio FMV

4.2 SVR

In this section, we illustrate the details of how we apply SVR to estimate the FMV of our synthetic VA portfolio. We observe that in our dataset the number of features is small, as suggested in [9] it may be useful to map data to high dimensional spaces. Radial basis function kernel (RBF kernel) is a frequently-used non-linear kernel that works well in practice, so we choose RBF kernel in our experiment. Generally, RBF kernel takes the following form:

$$K(\boldsymbol{y_i}, \boldsymbol{y_j}) = \exp(-\gamma||\boldsymbol{y_i} - \boldsymbol{y_j}||^2) \tag{4.5}$$

where $\boldsymbol{y_i}$ and $\boldsymbol{y_j}$ are two sample vectors and $||\boldsymbol{y_i} - \boldsymbol{y_j}||^2$ is the squared Euclidean distance. In our case, we need to slightly modify the distance measure such that it is consistent with the one we use in k-prototypes algorithm. The modified RBF that serves our purpose is defined as:

$$K(\boldsymbol{x}_i, \boldsymbol{x_j}) = \exp(-\gamma D(\boldsymbol{x}_i, \boldsymbol{x_j}, \lambda)^2) \tag{4.6}$$

here $\boldsymbol{x}_i$ and $\boldsymbol{x_j}$ are two VA contracts, and distance function $D(\cdot, \cdot, \lambda)$ is defined in Eq.(2.1). Since we will normalize all numeric features by the z-score method before doing any distance computations, as suggested in Gan [4], we set $\lambda = 1$ to keep a balance between categorical and numeric features when evaluating Eq.(4.6) in Section 4.2.

4.2.1 Training SVR Model

When training a SVR model under our experiment setting, there are two tuning parameters involved, cost parameter C and scale parameter γ. We perform a grid-search on C and γ and find out the optimal pair of (C, γ) based on 5-fold cross-validation error. Table 4.4 shows the values of C and γ used in the grid search. The final model that will be applied on the testing set is the one using the optimal pair of (C, γ). More formally, the training process is described in Algorithm 4.

Table 4.4: Values of SVR model parameters used in grid search

C	$10, 10^{1.5}, 10^2, 10^{2.5}, 10^3, 10^{3.5}, 10^4, 10^{4.5}, 10^5$
γ	$1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}$

Algorithm 4: Training the optimal SVR model

```
1  Normalize numeric features using z-score method ;
2  Define a set $\Gamma$ containing several possible $\gamma$ values ;
3  Define a set $\zeta$ of several possible $C$ values ;
4  for $\gamma \in \Gamma$ do
5      Construct a training kernel matrix $\mathbf{T}$ using $\gamma$, where $\mathbf{T}_{i,j} = K(\boldsymbol{z_i}, \boldsymbol{z_j})$ [a] ;
6      for $C \in \zeta$ do
7          Use $\mathbf{T}$ as input of LIBSVM [1], perform 5-fold cross-validation with cost
           parameter $C$ and obtain the cross-validation error;
8      end
9  end
10 Find the pair of $(C', \gamma')$ with the smallest cross-validation error ;
11 Train a final model the all training data with optimal $(C', \gamma')$ ;
```

[a] $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$ are two representative contracts, $K(\cdot, \cdot)$ is defined in Eq.(1.2)

Table 4.5 displays a subset of the cross-validation error with different γ and C combination. The error we compute here is the RMSE. These results are obtained from running

Table 4.5: A subset of cross-validation errors (SVR)

C	γ	$k = 100$	$k = 500$	$k = 1000$	$k = 2000$
$10^{4.5}$	10^{-1}	1060	1083	964	914
10^{5}	10^{-1}	1074	1032	866	848
10^{4}	10^{-1}	1122	1192	1077	1053
10^{5}	10^{-2}	1444	1544	1554	1751
$10^{3.5}$	10^{-1}	1641	1375	1278	1245
$10^{4.5}$	10^{-2}	2133	1705	1664	1781
10^{4}	10^{-2}	2680	2182	1999	1946
10^{5}	10^{-3}	2844	2621	2793	2920
10^{3}	10^{-1}	3183	1798	1587	1499

Numbers in red are the smallest cross-validation error for each k.

Algorithm 4 on a randomly chosen training set out of the 30 training sets. For the rest training sets, we also apply Algorithm 4 to select the optimal parameters.

We observe that the cross-validation error is more sensitive to γ than to C. For example, when k is 100, we keep $\gamma = 0.1$ and increase C from 10^3 to 10^5 (C is increased by 10 times), the RMSE only decreases by 4.3%. However, if we fix $C = 100,000$ and increase γ from 0.1 to 1 (γ is increased by 10 times), the RMSE increases by more than 200%.

4.2.2 Testing Results of SVR Model

The SVR model $\mathbf{SVR_{opt}}$ trained with the optimal parameters (C', γ') in Section 4.2.1 is then applied on its corresponding testing set. Our testing process is described in Algorithm

5.

Algorithm 5: Applying $\mathbf{SVR_{opt}}$ on testing instances $\boldsymbol{x}_i, i = 1, \dots, n-k$

```
1 Initialize an empty testing kernel matrix $\mathbb{T}'$ of dimension $(n-k) \times k$ [a];
2 Normalize numeric features according to the same mean and standard deviation in
  training set;
3 for $i \leftarrow 1$ to $n-k$ do
4     for $j \leftarrow 1$ to $k$ do
5         $\mathbb{T}'_{i,j} \leftarrow K(\boldsymbol{x}_i, \boldsymbol{z_j})$ [b] ;
6     end
7 end
8 Input $\mathbb{T}'$ and $\mathbf{SVR_{opt}}$ to LIBSVM [4] to obtain the FMV of $\boldsymbol{x}_i, i = 1, \dots, n-k$ ;
9 Compare the estimated FMV with the benchmark FMV of each VA contract,
  calculate the values of APD, RPD, RMSE and MAD;
```

[a] n is total number of VA contacts in the portfolio and k is the number of representative contracts

[b] $\boldsymbol{z_j}$ is a representative VA contract, $\boldsymbol{x}_i$ is an arbitrary VA contract in the portfolio whose FMV needs to be estimated, $K(\cdot,\cdot)$ is defined in Eq.(4.2)

Next we present the testing results of applying the trained SVR model on its corresponding testing set. APD, RPD, MAD and RMSE are calculated for each testing set. Table 4.6 and 4.7 show the average values as well as standard deviations of these four measurements on 30 independent testing sets. Table 4.8 displays the average CPU time of SVR using different size of training and testing set.

Table 4.6: Accuracy of SVR estimates

k	APD	RPD	MAD	RMSE
100	42,702,802	2.96%	4071	5234
500	17,289,638	1.20%	1890	2890
1000	14,447,890	1.01%	1440	2274
2000	2,649,731	0.18%	1080	1792

Table 4.6 to 4.8 show the accuracy of SVR with respect to the benchmark, as well as the CPU time needed to complete all required computation. We can see that the accuracy improves as k increases (APD, RPD, MAD and RMSE all decrease as k increases). Although SVR estimate is quite close to the benchmark value when $k = 2000$, but the time used is increased by more than 200% comparing with the benchmark method. One possibility is that when $k = 2000$ the 5-fold cross-validation and testing kernel matrix construction are both slow.

Table 4.7: Variation of SVR estimates

k	APD	RPD	MAD	RMSE
100	5,747,094	0.35%	170	207
500	2,207,429	0.15%	62	96
1000	3,034,228	0.23%	53	85
2000	2,145,686	0.14%	45	79

Table 4.8: Average CPU time used by SVR

	k			
	100	500	1000	2000
SVR	10.90	62.04	159.13	515.06

The numbers (in seconds) shown above are the total time of training plus testing.

4.3 Boosted Tree Model for Regression

In this section, we illustrate the details of how we use the boosted tree regression model to estimate the FMV of our portfolio of VA contracts. We still use the 30 training sets and their corresponding testing sets for each k.

4.3.1 Training Boosted Tree Regression Model

When building a boosted tree regression model, we mainly follow [7] to do a grid-search on model parameters. The **R caret** package can help facilitate the search of the optimal set of parameters. For a boosted tree regression model, there are four major tuning parameters, which are:

- number of iterations, i.e. trees, denoted by n.trees
- maximum depth of a regression tree,denoted by max tree depth
- learning rate, denoted by shrinkage
- the minimum number of training data points required in a node to start splitting, denoted by n.minobsinnode

Table 4.9 shows the values of each parameter that will be used in the grid-search, so in total we will search for 240 possible combinations of these four parameters. To make the cross-validation results more robust, for each parameter combination instead of just doing 5-fold cross-validation one time, we repeat it five times. Note time spent on this step is included in the final CPU time we record.

Table 4.9: Values of boosted tree model parameters used in grid-search

Parameter	Values
n.trees	50 to 1000, with step size 50
max tree depth	4,5,6
learning rate	0.001,0.01
n.minobsinnode	5,10

Figure 4.2 to Figure 4.6 show the behaviour of cross-validation error (RMSE) as tuning parameters varied for different k on one randomly selected training set. From these figures we observe the following for this data set:

- learning rate 0.01 is better than 0.001
- effects of the maximum tree depth and n.minobsinnode are not significant for larger k
- when k increases from 1000 to 2000, the reduction of error is not obvious
- when learning rate is fixed at 0.01, the cross-validation error first decreases fast as the number boosting iteration increases; however when the number boosting iteration reaches a certain value, add more iterations does not reduce the cross-validation error significantly.

When choosing the parameter values to build the final model for each k, as suggested in [7], we could select a less complex model by allowing certain amount of loss in performance. This may avoid the over-fitting problem. Thus, we aim to find a model $\mathbf{M}'$ whose cross-validation error is within 1.5% of that of the model $\mathbf{M}_{opt}$ with the smallest cross-validation error.

Table 4.10 shows the final parameters selected by 5-fold cross-validation. We can see from this table, where the same set of parameters is used for $k = 1000$ and $k = 2000$, the

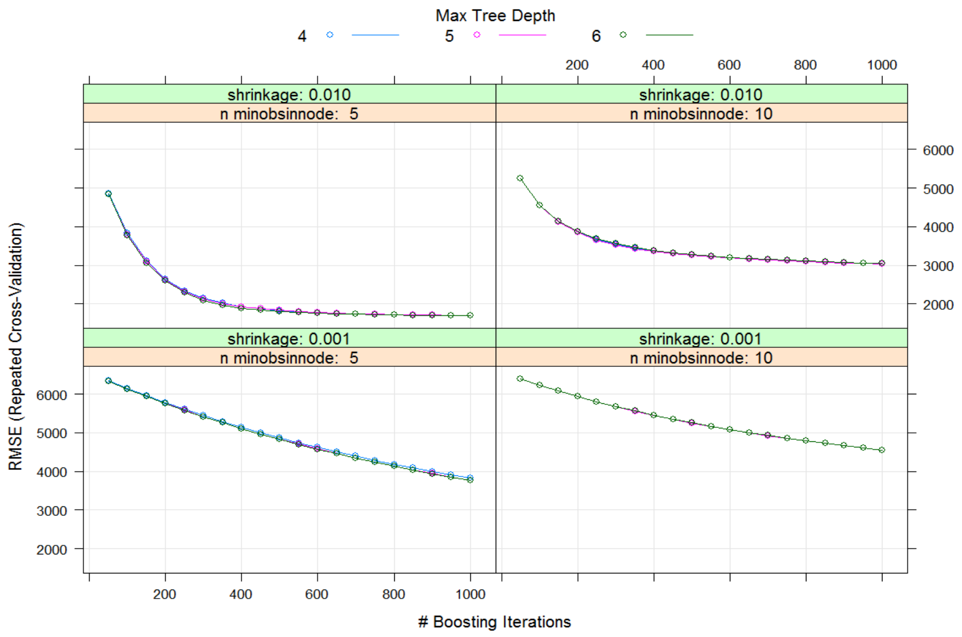

Figure 4.2: Cross-validation error curve($k = 100$)

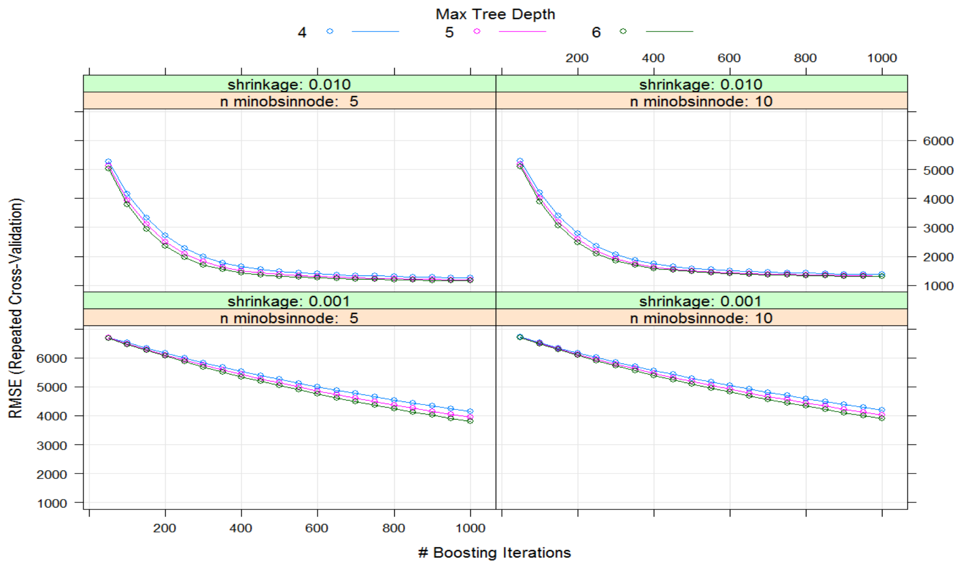

Figure 4.3: Cross-validation error curve ($k = 500$)

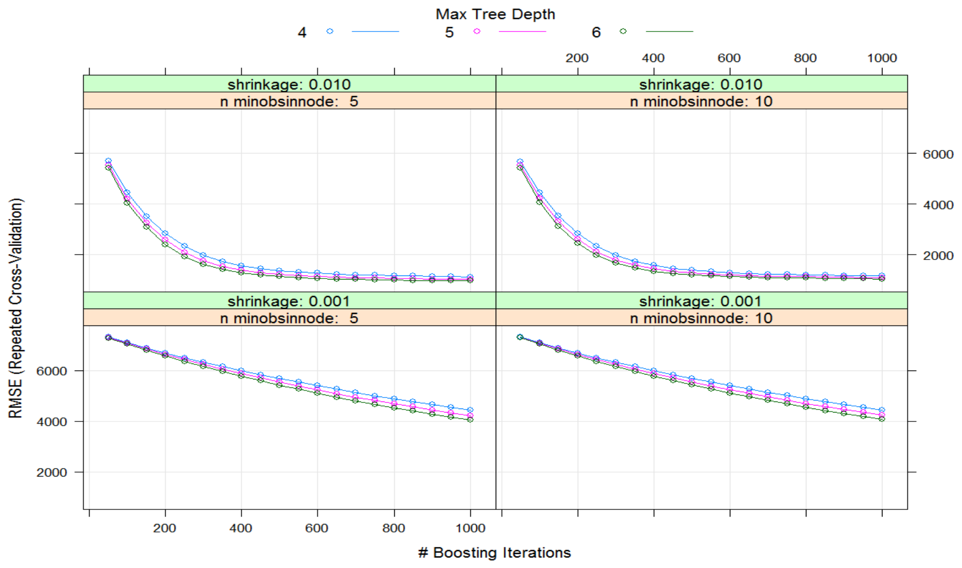

Figure 4.4: Cross-validation error curve ($k = 1000$)

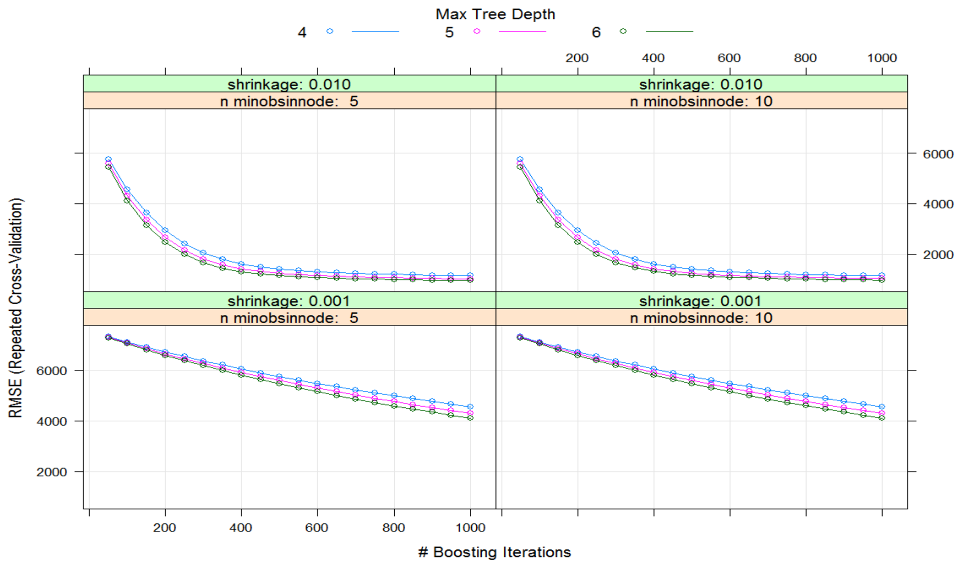

Figure 4.5: Cross-validation error curve ($k = 2000$)

time used on selection is almost doubled, but the error decreases only by 1.2%, which may indicate it is not worth using 2000 VA representative contracts to build the boosted tree regression model.

Table 4.10: Boosted tree regression model parameters selected by 5-fold cross-validation

	k			
	100	500	1000	2000
n.trees	800	900	950	950
max tree depth	6	6	6	6
learning rate	0.01	0.01	0.01	0.01
n.minobsinnode	5	5	5	5
cross-validation error	1710	1169	973	961
CPU time (seconds)	16.50	64.48	124.74	231.64

4.3.2 Testing Results of Boosted Tree Regression Model

A boosted tree regression model with optimal value of parameters is trained for each k, then it is applied on the corresponding testing set to calculate the FMV of each VA contract. The next step is to compare the estimated portfolio FMV with the benchmark portfolio FMV to obtain the values of APD, RPD, RMSE and MAD

Here we present the testing results of applying the trained boosted tree model on its corresponding testing set. APD, RPD, MAD and RMSE are calculated for each testing set. Table 4.11 and 4.12 show the average values as well as standard deviations of these four measurements on 30 independent testing sets. Table 4.13 displays the average CPU time of using boosted tree regression model for each k.

Table 4.11: Accuracy of boosted tree regression estimates

k	APD	RPD	MAD	RMSE
100	153,853,768	10.68%	3928	5460
500	1,772,048	0.12%	1907	2897
1000	4,310,227	0.30%	1468	2421
2000	6,896,494	0.48%	1202	1973

Table 4.11 to 4.13 show the accuracy of the boosted tree method with respect to the benchmark, as well as the CPU time required to complete all relevant computations. Compared with the testing results of SVR in Table 4.6 to 4.8, we see that the accuracy and

Table 4.12: Variation of boosted tree regression estimates

k	APD	RPD	MAD	RMSE
100	9,441,610	0.34%	152	215
500	1,468,954	0.10%	63	99
1000	1,459,104	0.10%	50	83
2000	742,224	0.05%	38	63

Table 4.13: Average CPU time used by boosted tree regression

	k			
	100	500	1000	2000
Boosted tree	20.86	69.37	124.66	237.24

The numbers (in seconds) shown above are the total time of training plus testing.

computation speed of boosted tree method is worse than that of SVR when $k = 100$; however, it out-performs SVR when $k = 500$ since the RPD decreases by a factor of 10, while the CPU time only increases by 11%. Additionally, it can also achieve good accuracy when $k = 2000$, but the time it uses is close to the CPU time of the benchmark computation of the portfolio value.

4.4 Ordinary Kriging Method

In this section we apply the ordinary Kriging method described in Section 2.2 to compute the portfolio FMV of the synthetic VA contracts. We do it in the following way: for each training set of size k, we first normalize the numeric features using z-score method and construct the matrix $\mathbf{A}$; secondly normalize its corresponding testing set according to same mean and standard deviation and construct the matrix $\mathbf{\Gamma}$; compute the FMV of each VA contract in that testing set by Eq.(2.14). In addition, we are also interested in how the method parameters affect the Kriging estimates so we perform a analysis of sensitivities of method parameters in Section 4.4.1. In Section 4.4.2 we presents the testing results of the ordinary Kriging method.

4.4.1 Sensitivities of Method Parameters

From Eq.(2.8) and Eq.(2.9) we see that for the ordinary Kriging method there are three parameters λ, α and β. To be consistent with the distance computations in Section 4.2, we also set $\lambda = 1$ but further investigate the influences of α and β on accuracy of the ordinary Kriging estimates.

Table 4.15 shows the RPD values of ordinary Kriging method estimates based on a randomly selected training set of size k and its corresponding testing data. We can see that basically α has no effects on the Kriging estimates. Mathematically, this is because the augmented matrix for the linear equation system in Eq.(2.7) based on Eq.(2.8) and Eq.(2.9) is

$$\left(\begin{array}{cccc|c} \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{z_1},\boldsymbol{z_1},\lambda)) & \cdots & \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{z_1},\boldsymbol{z_k},\lambda)) & 1 & \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{x_i},\boldsymbol{z_1},\lambda)) \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{z_k},\boldsymbol{z_1},\lambda)) & \cdots & \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{z_k},\boldsymbol{z_k},\lambda)) & 1 & \alpha+\exp(-\frac{3}{\beta}D(\boldsymbol{x_i},\boldsymbol{z_k},\lambda)) \\ 1 & \cdots & 1 & 0 & 1 \end{array}\right) \tag{4.7}$$

and by adding the last row multiplied by $-\alpha$ to all other rows, the matrix in (4.7) becomes

$$\left(\begin{array}{cccc|c} \exp(-\frac{3}{\beta}D(\boldsymbol{z_1},\boldsymbol{z_1},\lambda)) & \cdots & \exp(-\frac{3}{\beta}D(\boldsymbol{z_1},\boldsymbol{z_k},\lambda)) & 1 & \exp(-\frac{3}{\beta}D(\boldsymbol{x_i},\boldsymbol{z_1},\lambda)) \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \exp(-\frac{3}{\beta}D(\boldsymbol{z_k},\boldsymbol{z_1},\lambda)) & \cdots & \exp(-\frac{3}{\beta}D(\boldsymbol{z_k},\boldsymbol{z_k},\lambda)) & 1 & \exp(-\frac{3}{\beta}D(\boldsymbol{x_i},\boldsymbol{z_k},\lambda)) \\ 1 & \cdots & 1 & 0 & 1 \end{array}\right) \tag{4.8}$$

we can easily see that α does not affect the solution to (2.7). Therefore we set $\alpha = 0$ for any computations related to Eq.(2.8) and Eq.(2.9).

Table 4.14: Accuracy of ordinary Kriging estimates with different α values

	k			
	100	500	1000	2000
$\alpha = 0$	9.32%	2.97%	1.07%	0.82%
$\alpha = 10$	9.32%	2.97%	1.07%	0.82%
$\alpha = 100$	9.32%	2.97%	1.07%	0.82%

The numbers are RPD values.

From (4.8) we know β can affect the solution to (2.7), then to assess the effects of β, we use the 5-fold cross-validation approach on a randomly selected training set of size k. Let $D_{max} = \max\{D(\boldsymbol{z_r}, \boldsymbol{z_s}, \lambda), 1 \leq r < s \leq k\}$, the β values we have tried are $0.5D_{max}, D_{max}, 1.5D_{max}, 2D_{max}, 2.5D_{max}, 3D_{max}$.

Table 4.15: Cross-validation error (RMSE) with different β values

	k			
	100	500	1000	2000
$\beta = 0.5D_{max}$	4451	1930	1567	1075
$\beta = D_{max}$	3338	1553	1229	899
$\beta = 1.5D_{max}$	3188	1470	1245	869
$\beta = 2D_{max}$	2811	1537	1202	866
$\beta = 2.5D_{max}$	2978	1492	1162	838
$\beta = 3D_{max}$	2723	1405	1190	849

Numbers in red are the smallest errors of each k.

Table 4.15 shows how the cross-validation errors change for different β values for each k. As we can see β does play a role in affecting the Kriging estimates. Gan [4] uses a fixed value of β (the 95^{th} percentile of the distances $D(\boldsymbol{z_r}, \boldsymbol{z_s}, \lambda), 1 \leq r < s \leq k$), which may not be the optimal one. To select the optimal β for each training set of size k, we perform the 5-fold cross-validation using the β values specified in Table 4.15.

4.4.2 Testing Results of ordinary Kriging Method

Table 4.16: Accuracy of ordinary Kriging estimates

k	APD	RPD	MAD	RMSE
100	102,044,452	7.08%	4356	5424
500	12,086,435	0.84%	2249	3166
1000	5,430,605	0.38%	1702	2589
2000	1,986,349	0.14%	1102	1862

Table 4.16 to 4.18 show the accuracy and variation of the ordinary Kriging estimates with respect to the benchmark, as well as the CPU time required to complete all computation (including 5-fold cross-validation). Compared with the testing results of SVR and

Table 4.17: Variation of ordinary Kriging estimates

k	APD	RPD	MAD	RMSE
100	5,323,923	0.19%	213	257
500	1,694,116	0.10%	77	109
1000	985,586	0.06%	57	86
2000	491,350	0.04%	36	62

Table 4.18: Average CPU time used by the ordinary Kriging method

	k			
	100	500	1000	2000
ordinary Kriging method	3.11	17.03	34.78	81.24

The number are in seconds.

boosted tree method, the ordinary Kriging method runs much faster (partly because we only need to tune one parameter β). Its accuracy is also quite good when $k = 2000$ while it only takes about $\frac{1}{3}$ of CPU time of the benchmark computation of the portfolio value.

4.5 Comparison of the Three Regression Methods

In this part we mainly compare the testing results of the SVR, boosted tree, ordinary Kriging method with the benchmark Monte Carlo method, in terms of accuracy, robustness and computation speed.

In Figure 4.6 each line represents the average CPU time of representative contracts selection and valuation plus the one regression method, as size of representative contract set (training set) k increases. Note that each regression method uses the same set of representative contracts as training set.

It is easy to see the ordinary Kriging method the slowest rate of increase in CPU time as k increases, whereas boosted tree method has faster rate of increase than ordinary Kriging method. We also observe that when $k = 100$ the CPU time of each method does not differ quite much, however when k reaches 2000, SVR runs even slower than the benchmark. One explanation is that for SVR when $k = 2000$ in the model testing step we need to construct a testing kernel matrix of size 98000×2000 (uses about 1.46 GB of memory), and applying the trained SVR on this large matrix can be slow.

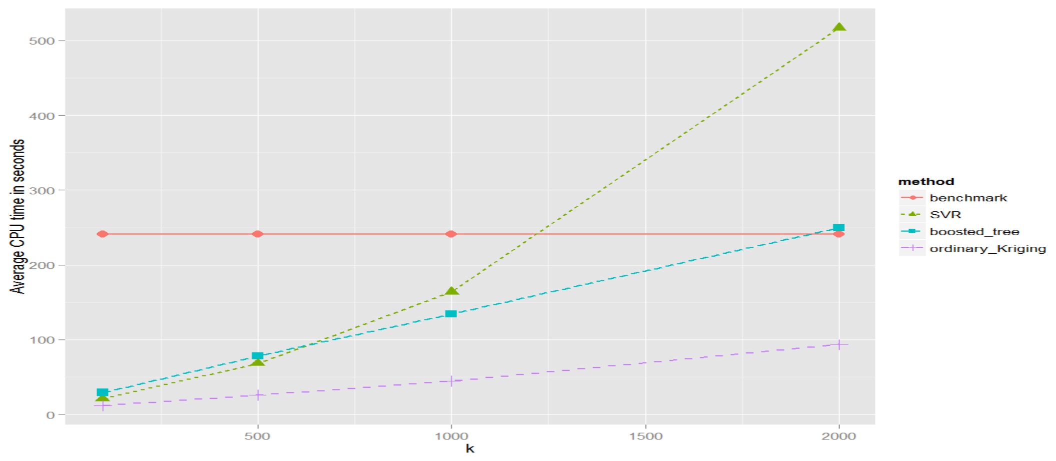

Figure 4.6: Average CPU time of representative contracts selection and valuation plus regression

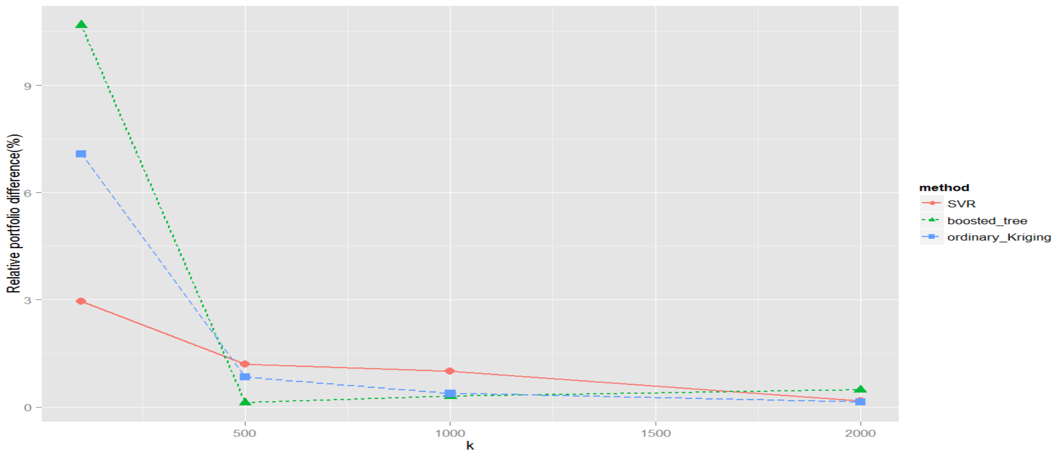

Figure 4.7: Accuracy of SVR, boosted tree, ordinary Kriging method

Figure 4.7 shows the average values of RPD for SVR, boosted tree and ordinary Kriging method. If we treat RPD as the main criterion for accuracy and we need to balance the accuracy and computation time, SVR should be chosen over the other two if we only have a set of 100 representative contracts. When $k = 100$ or 500, boosted tree method has good overall performance because it keeps a good balance between accuracy and computation speed. However when k reaches 2000, we should use the ordinary Kriging method because the other two perform worse than the benchmark method. On the other hand, we observe in Section 4.1.3 that the benchmark portfolio FMV itself has 4% relative variation. Due to this relative variation, increasing k to make the RPD as small as possible may not be reasonable. In that sense, SVR using 100 training data points works the best. Note that this computational investigation is entirely based on a single set of synthetic VA contracts. Therefore, the conclusions we draw here is subject to change of the VA portfolio under study.

4.6 Effectiveness of the Data Clustering Method

From Section 4.2 to 4.4, training data points are generated using the data clustering method, we assess the effectiveness of this method by using another approach of obtaining training set of a certain size. To do this we only need to modify the second step among those five steps described in Section 4.1. We replace the data clustering method with the random sampling method in generating a training set of size k, while all other steps stay the same. In Section 4.6.1 we compare the mean accuracy of regression estimates, using training data points from data clustering method and random sampling respectively. In Section 4.6.2 we compare the variation of regression estimates based on these two approaches.

4.6.1 Comparing the Data Clustering Method and Random Sampling on Mean Accuracy

Figure 4.8 shows the average values of RPD for the boosted tree method using two types of training data points selection methods. Surprisingly when $k = 100$, using randomly selected contracts from the portfolio can result in much better mean accuracy than using the so-called representative contracts obtained from the data clustering method. However as k increases from 500 to 2000, the average RPD does not change much for both methods, which indicates large set of training data points is somehow unnecessary for the boosted tree method.

We can see from Figure 4.9 that for the ordinary Kriging method, using randomly sampled training data points could also lead to smaller RPD for each k. In addition, when using randomly sampled data the ordinary Kriging method is not sensitive to the change in k as k gets larger than 500.

Figure 4.10 shows results of applying SVR on both randomly selected training data points and representative contracts obtained from the data clustering method. Their dif-

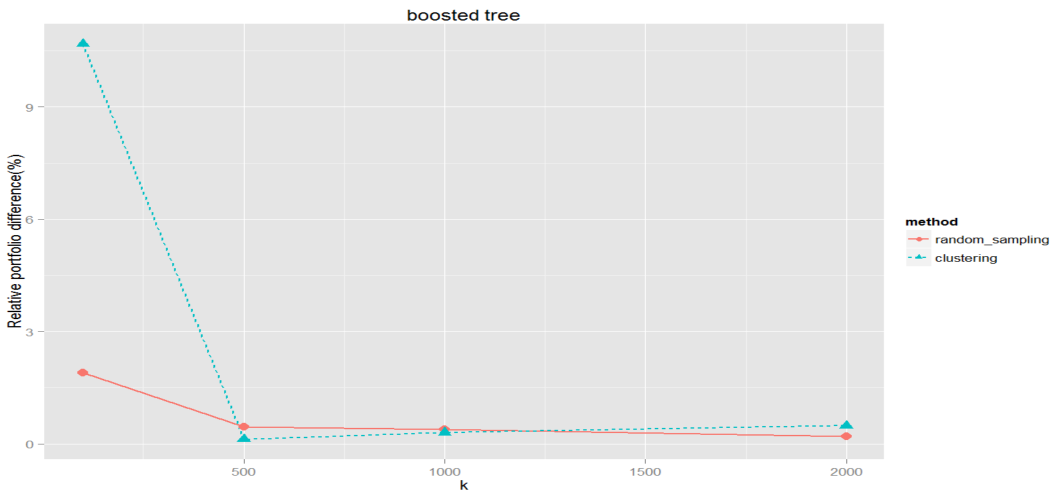

Figure 4.8: Accuracy of boosted tree regression estimates, using random sample and clustering respectively

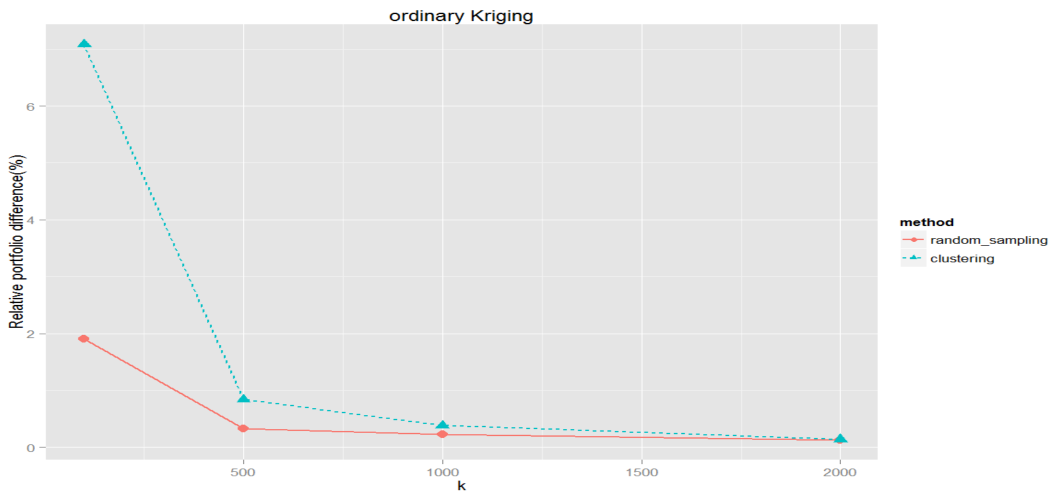

Figure 4.9: Accuracy of ordinary Kriging estimates, using random sample and clustering respectively

ferences are not obvious for large k, and random sampling gives better mean accuracy when $k = 100$.

As shown in Figure 4.8 to 4.10 we can conclude that using the data clustering method

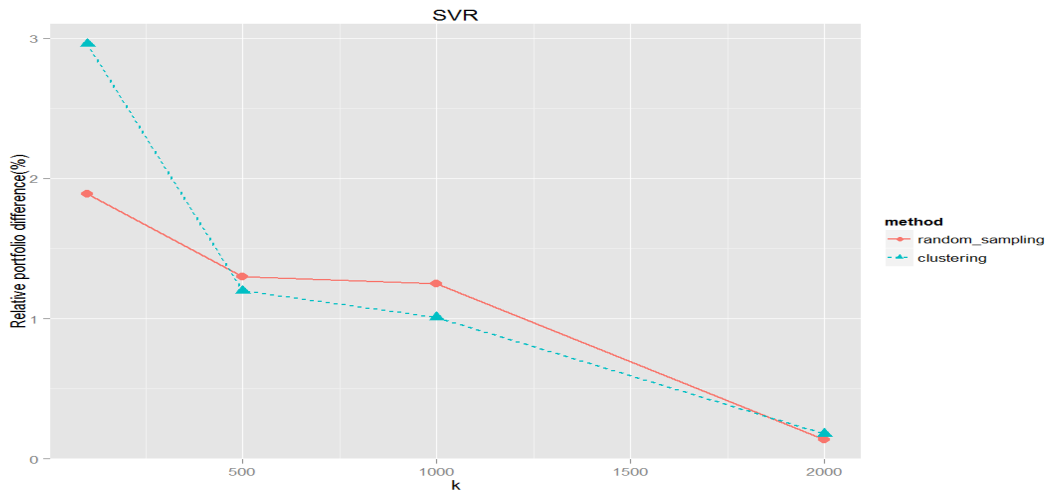

Figure 4.10: Accuracy of SVR estimates, using random sample and clustering respectively

to select representative contracts as training data points in fact does not help improve the mean accuracy of regression estimates, while in many cases using random sampling, which is simple and fast can yield better mean accuracy.

4.6.2 Comparing the Data Clustering Method and Random Sampling in Coefficient of Variation (CV)

Figure 4.11 to 4.13 shows the CV of three regression estimates, using training data points from both random sampling and the data clustering method. Since CV measures the amount of variability relative to the mean, we can see that using random sampled training data points generally causes the regression estimates to have more variation. This is partly because random sampling introduces more variation into the set of training data points, causing the regression estimates based on them to vary a bit more than those based on the data clustering method, which is a more systematic way of selecting training data points.

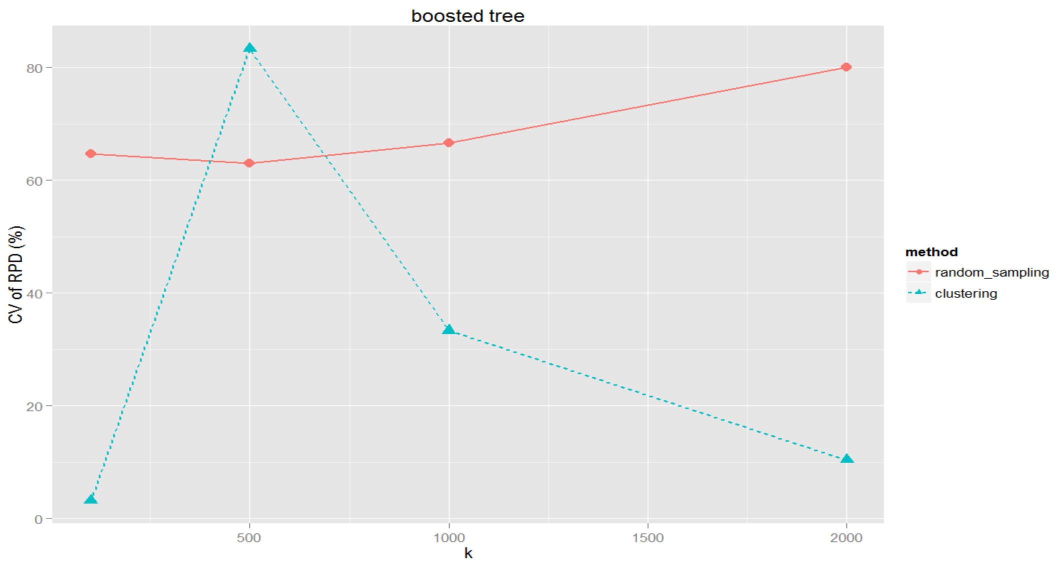

Figure 4.11: CV of boosted tree regression estimates, using random sample and clustering respectively

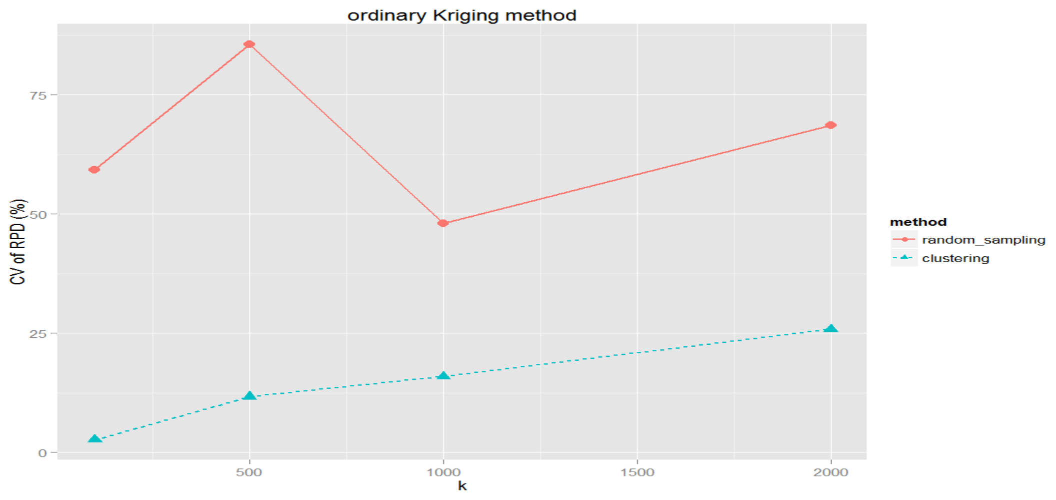

Figure 4.12: CV of ordinary Kriging estimates, using random sample and clustering respectively

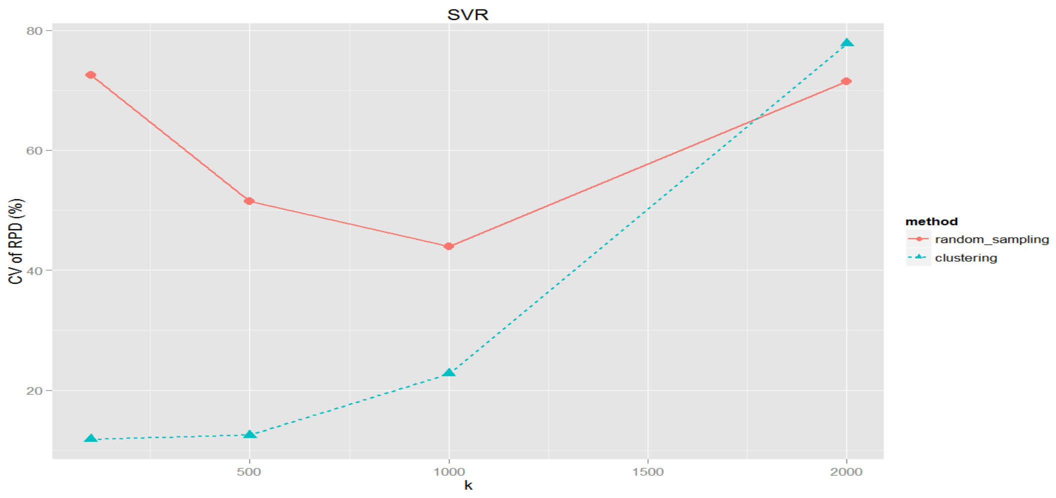

Figure 4.13: CV of SVR estimates, using random sample and clustering respectively

Chapter 5

Conclusion

In this computational study we focus on comparing the accuracy and speed of three regression methods, namely SVR, boosted tree regression and ordinary Kriging method with the traditional Monte Carlo method, which serves as the benchmark for comparison, in estimating the FMV of portfolio of VA contracts. We also assess the effectiveness of the data clustering method in comparison with the random sampling technique in selecting training data points for the regression step.

Our study have found that in fact compared with random sampling, data clustering method does not help improve the mean accuracy of the regression estimates, at least for the synthetic portfolio considered here and in Gan [4]. Although random sampling in our experiment gives more accurate regression estimates on average, it indeed brings more variation into the regression estimates. However, random sampling does have one advantage in practice since it can run more faster than the data clustering method on a large portfolio of VA contracts.

We also observe that based on a smaller (e.g. $k = 100$) set of training data points selected by the data clustering method, SVR seems to have the best performance since it has the smallest RPD among all three methods while the CPU time used is close to the other method. As the training set size gets larger, the difference in accuracy tends to diminish whereas the speed comes to play an important role in determining which regression method is preferred. However, given the fact the benchmark portfolio values have about 4% relative variation itself, it seems that selecting $k = 100$ is a reasonable choice here.

One important thing that we have not discussed is the hedging issue. For the ordinary Kriging method, the estimated portfolio FMV is a explicit linear combination of the FMV of the representative contracts. Since FMV of a single representative contract is a function of the underlying asset **S**, the "Greek", i.e. the derivative of the portfolio FMV with respect to **S**, is just a linear combination of the "Greek" of the representative contracts. However, for the SVR and boosted tree regression, the relationship between the estimated portfolio FMV and the FMV of the representative contracts is implicit, therefore calculating the "Greek" of the portfolio is not as straightforward as it in the ordinary Kriging method and this is a topic for future work.

References

[1] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[2] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

[3] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[4] Guojun Gan. Application of data clustering and machine learning in variable annuity valuation. *Insurance: Mathematics and Economics*, 53(3):795–801, 2013.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[6] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Min. Knowl. Discov.*, 2(3):283–304, 1998.

[7] Max Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[8] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[9] Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification, 2010.