Chapter 3

Markov Model Driven Gaussian Process Based Trust Models

3.1 Introduction

One important requirement in a Web services network is that a user can make security decisions regarding usage of a Web service. Such decisions aim at minimizing the risk of abusing or destroying resources in an interaction. To make such security critical decisions, a user needs to assess its confidence that an interaction with a Web service is secure. Such confidence is derived from trust and reputation. An essential component in the trust management framework is a behaviour model base with which any user can predict the future trust or reputation of a service provider.

In this chapter we propose two behaviour model frameworks for evaluation of trustworthiness of a service provider based on the experience of direct interactions and/or the recommendations from the other service users. The behaviour model frameworks are based on the theory of Markovian process and the principle of non-linear time series prediction. Markovian process models facilitate learning the dynamic and state based behaviour of a Web service.

Both the frameworks use Gaussian Process Regression (GPR) for prediction. A GPR model is equipped with a kernel function and is able to learn useful patterns from the available training datasets and perform data interpolation and extrapolation. A GPR is able to make predictions using small and scarce training datasets. In addition, it provides a predictive distribution defined by the mean value together with the respective prediction variance.

3.2 Background

3.2.1 Markov Models

3.2.1.1 Markov Chains

We have a set of states, $S = \{s_1, s_2, ..., s_R\}$ .The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state s_i , then it moves to state s_j at the next step with a probability denoted by p_{ij} , and this probability does not depend upon which states the chain was in before the current state. The probabilities p_{ij} are called transition probabilities. The process can remain in the state it is in, and this occurs with probability p_{ij}. An initial probability distribution, defined on *S*, specifies the starting state. Usually this is done by specifying a particular state as the starting state.

3.2.1.2 Hidden Markov Model (HMM)

First introduced and studied in the late 1960s, Hidden Markov Model is a very powerful model for predicting the future trend based on sequential datasets [122]. A HMM is a Markov chain with each state associated with a particular probability distribution over the set of possible symbols, also called observations. However, a key difference between a HMM and a Markov chain is that in a HMM, state transitions are not observed as is the case in a Markov chain, and only observations are visible. We precisely define discrete-time first-order HMM. The reader is referred to Rabiner [122] for lucid details.

Definition
Formally, a discrete-time HMM is defined by the following elements:

- A set $S = \{s_1, s_2, ..., s_R\}$ of (hidden) R states.

- A state transition matrix $A = \left\{a_{s_i s_j}\right\}$ of size $R \times R$, where each element $a_{s_i s_j}$ is the probability of transition from state s_i to state s_j.
- A set $V = \left\{z_1, z_2, \ldots, z_K\right\}$ of observation symbols. A sequence of observation symbols is the physical output of the underlying dynamic process.
- An emission matrix $B = \left\{b_{s_i}\left(z_k\right)\right\}$ of size $R \times K$, where an element $b_{s_i}\left(z_k\right)$ is the probability of observing symbol z_k given the current state of the dynamic process is S_i.
- Initial state probability distribution $\pi = \left\{\pi_{s_i}\right\}$, where an element π_{s_i} is the probability of being in state S_i at the time 1

Thus an HMM is completely defined by $\lambda = \left(S, V, A, B, \pi\right)$. The probability distributions A, B, and π are called the parameters of the given HMM.

3.2.1.3 Basic problems of HMM

For any HMM model, there are three basic problems of interest to be solved for the model to be useful in real-world applications. We describe them as follows:

Problem 1: *Given the observation sequence* $\mathrm{O} = \left\{o_1, o_2, \ldots, o_T\right\}; o_i \in V$ *and a model* λ*, how to efficiently compute the probability of the observation sequence* $P\left(O \mid \lambda\right)$*?*

- **Problem 2:** *Given the observation sequence* O *and a HMM model* λ*, how to choose a corresponding state sequence* $\mathrm{Q}=\{q_1, q_2, \ldots, q_T\}; q_i \in S$*?*
- **Problem 3:** How to adjust the model parameters λ?

3.2.1.4 Forward-Backward Algorithm

Forward-Backward algorithm [122] is used to solve **Problem 1** at a low computational cost. In the following, we present a brief sketch of this algorithm.

For a give HMM λ, the joint probability of a sequence of observation $O = \{o_1, o_2, \ldots, o_T\}; o_i \in V$ and the underlying sequence of states $Q=\{q_1, q_2, \ldots, q_T\}; q_i \in S$ is defined as

$$P(O,Q \mid \lambda) = \pi_{q_1}.b_{q_1}(o_1).a_{q_1q_2}.b_{q_2}(o_2)...a_{q_{T-1}q_T}b_{q_T}(o_T).$$

The probability of the outcome sequence $O = \{o_1, o_2, \ldots, o_T\}$ given the HMM λ is therefore equal to the sum of joint probability $P(O,Q \mid \lambda)$ over all possible sequences Q of states. That is,

$$P(O \mid \lambda) = \sum_{q_1, \ldots q_T \in S} \pi_{q_1}.b_{q_1}(o_1).a_{q_1q_2}.b_{q_2}(o_2)...a_{q_{T-1}q_T}b_{q_T}(o_T) \qquad (3.1)$$

This probability is obtained by evaluating inductively the *forward* variable given in Eq. (3.2)

$$\alpha_t(s_i) = P(o_1, o_2, \ldots, o_t, q_t = s_i \mid \lambda). \qquad (3.2)$$

Here $\alpha_t(s_i)$ is the joint probability of the partial observation sequence, $o_1, o_2, \ldots, o_t$ and given the state S_i at time t. A *forward* procedure evaluates $P(O \mid \lambda)$ as given below.

1. Initialization :

$$\alpha_1(s_i) = \pi_{s_i} b_{s_i}(o_1), \qquad 1 \le i \le R$$

2. Induction Step :

$$\alpha_{t+1}(s_j) = (\sum_{i=1}^{R} \alpha_t(s_i) a_{s_i s_j}).b_{s_j}(o_{t+1}), \quad 1 \le t \le T, 1 \le j \le R$$

3. Termination :

$$P(O \mid \lambda) = \sum_{i=1}^{R} \alpha_T(s_i)$$

In a similar way to the above procedure, the probability $P(O|\lambda)$ can also be computed by evaluating inductively of the *backward* variable, denoted by $\beta_t(s_i)$ defined below.

$$\beta_t(s_i) = P(o_{t+1}o_{t+2}....o_T \mid q_t = s_i, \lambda). \tag{3.3}$$

Here $\beta_t(s_i)$ is the conditional probability of the partial observation sequence $o_{t+1}o_{t+2}....o_T$ given the state s_i at time t. The *backward* procedure works as below.

1. Initialization :

$$\beta_T(s_i) = 1, \qquad 1 \le i \le R$$

2. Induction Step :

$$\beta_t(s_i) = \sum a_{s_i s_j} b_{s_j}(o_{t+1}).\beta_{t+1}(s_j), \qquad 1 \le t \le T, 1 \le j \le R$$

3. Termination :

$$P(O|\lambda) = \sum_{i=1}^{R} \pi_{s_i} b_{s_i}(o_1).\beta_1(s_i)$$

3.2.1.5 Viterbi algorithm

The **problem-2** of HMM is solved by the Viterbi algorithm originally proposed by [158]. The algorithm is designed to find the most likely state sequence $Q=\{q_1, q_2, \ldots, q_T\};\ q_i \in S$ given the observation sequence $O = \{o_1, o_2, \ldots, o_T\}; o_i \in V$ by maximizing the probability, $P(Q|O,\lambda)$. Let us define

$$\chi_t(q_i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(s_1, s_2, \ldots, s_t = q_i, o_1, o_2, \ldots, o_t \mid \lambda)$$

as the largest probability of a state sequence until time t that ends in the state q_i In order to obtain the best sequence that maximizes $\chi_t(q_i)$, the following Viterbi algorithm is used.

Algorithm 3.1: Viterbi

Steps:

1. [**Initialise for all** $q_i; i = 1, 2....T$]

$$\chi_1(q_i) \leftarrow \pi_{s_i} b_{s_i}(o_1) \ ; \ \psi_1(q_i) \leftarrow 0$$

2. [**Recurs for all** $q_j; j = 1, 2....T$]

$$\chi_t(s_j) \leftarrow \max_t [\chi_{t-1}(s_i) a_{s_i s_j}] b_{s_j}(o_t) \quad 2 \le t \prec T$$

$$\psi_t(s_j) \leftarrow \operatorname{argmax}_t [\chi_{t-1}(s_i) a_{s_i s_j}] \quad 2 \le t \prec T$$

2. [**Terminate]**

$$s_T^* \leftarrow \operatorname{argmax}_t [\chi_T(s_i)]$$

3. [**State sequence backtracking]**

$$s_t^* = \psi_{t+1}(s_{t+1}^*) \quad t = T-1, T-2, ..., 1$$

Here $\psi_t(s_j)$ is an array used to keep track of the arguments which actually maximized $\chi_t(s_j)$. The final resulting optimal state sequence is $S_t^* = (s_1^*, ..., s_T^*)$

3.2.1.6 Baum-Welch algorithm

Bauam-Welch algorithm [123] is used for solving **Problem 3**. This technique works in the framework of Expectation Maximization (EM) [124]. In this technique, we assume that a priori HMM λ' is given. Then our aim is to derive a posterior HMM λ that maximizes the expected complete data likelihood defined below.

$$L(\lambda', \lambda) = \sum_q P(q \mid O, \lambda') \log P(O, q \mid \lambda). \tag{3.4}$$

This function is called the *Baum's auxiliary function.* The objective now is to determine the optimal parameter values of the posteriori HMM λ which maximizes $L(\lambda', \lambda)$. The term $\log P(O, q \mid \lambda)$ in Eq. (3.4) can be written in terms of the parameters of λ as,

$$\log P(O, q \mid \lambda) = \log \pi_{q_1} + \sum_{t=2}^{T} \log a_{q_{t-1} q_t} + \sum_{t=1}^{T} \log b_{q_t}(o_t) \tag{3.5}$$

Using Eq. (3.5) in Eq. (3.4), we have,

$$L(\lambda^{'},\lambda)=\sum_{i=1}^{R}P(q_1=s_i\,|\,O,\lambda^{'})\log\pi_{s_i}+$$

$$\sum_{i=1}^{R}\sum_{j=1}^{R}\sum_{t=2}^{T}P(\mathrm{q}_{t-1}=s_i,q_t=s_j\,|\,O,\lambda^{'})\log a_{s_is_j}+$$

$$\sum_{i=}^{R}\sum_{k=1}^{K}\sum_{t=1}^{T}P(q_t=s_i\,|\,O,\lambda^{'})\delta(o_t,q_k)\log b_{s_i}(q_k). \tag{3.6}$$

Here $\delta(o_t,q_k)$ is defined as

$$\delta(o_t,q_k)=\begin{cases}1 & if \quad o_t=q_k\\ 0 & otherwise\end{cases} \tag{3.7}$$

Now Eq. (3.6) can be rewritten as follows.

$$L(\lambda,\lambda^{'})=Q_{\pi}(\pi)+\sum_{i=1}^{R}Q_{a_i}(A_{s_i})+\sum_{i=1}^{R}Q_{b_i}(B_{s_i}) \tag{3.8}$$

Here $\pi=\left[\pi_{s_1},\pi_{s_2},...\pi_{s_R}\right]$ is the initial state probability distribution, $A_{s_i}=\left[a_{s_is_1},a_{s_is_2}....,a_{s_is_R}\right]$ is probability distribution from state s_i to other states, and $B_{s_i}=\left[b_{s_i}(z_1),b_{s_i}(z_2),...,b_{s_i}(z_K)\right]$ is the emission probability distribution over the outcomes given the state s_i.

The three Qs in Eq. (3.8) are defined as following.

$$Q_{\pi}(\pi)=\sum_{i=1}^{R}P(q_1=s_i\,|\,O,\lambda^{'})\log\pi_{s_i} \tag{3.9}$$

$$Q_{a_i}(A_{s_i})=\sum_{j=1}^{R}(\sum_{t=2}^{T}P(q_{t-1}=s_i,q_t=s_j\,|\,O,\lambda^{'}))\log A_{s_is_j} \tag{3.10}$$

$$Q_{b_i}(B_{s_i})=\sum_{k=1}^{K}(\sum_{t=1}^{T}P(q_t=s_i\,|\,O,\lambda^{'})\delta(o_t,z_k))\log B_{s_i}(z_k) \tag{3.11}$$

Maximizing the auxiliary function is achieved by maximizing each term of Eq. (3.8). Further using the *Lagrange multiplier* technique for optimizing functions in Eq. (3.9) to Eq. (3.11), the parameters of the optimal a posteriori model λ are given as

$$\overline{\pi_{s_i}} = P(q_1 = s_i \mid O, \lambda') \tag{3.12}$$

$$\overline{A}_{s_i s_j} = \frac{\sum_{t=2}^{T} \xi_{t-1}(s_{i,} s_j)}{\sum_{t=2}^{T} P(q_{t-1} = s_i \mid O, \lambda')} \tag{3.13}$$

$$\overline{B}_{s_i}(z_k) = \frac{\sum_{t=1}^{T} P(q_t = s_i \mid O, \lambda')\delta(o_t, z_k)}{\sum_{t=1}^{T} \gamma_t(s_i)} \tag{3.14}$$

Here $\gamma_t(s_i) = P(q_t = s_i \mid O, \lambda')$ and $\xi_{t-1}(s_{i,} s_j) = P(q_{t-1} = s_i, q_t = s_j \mid O, \lambda')$. The former represents the probability of visiting state s_i at time t given an observation sequence O and an HMM λ' while the latter is the probability of visiting states s_i and s_j at times $t-1$ and t respectively. These probabilities are efficiently evaluated in linear time using the *forward* and *backward* variables $\alpha_t(s_i)$ and $\beta_t(s_i)$ defined in Eq. (3.2) and Eq. (3.3) respectively. The Eq. (3.12) to Eq. (3.13) are termed, in the literature, as *parameter re-estimation equations.*

Given the parameters of the a priori HMM λ', these three equations represent one iteration in the Baum-Welch algorithm for estimation of the parameters of the a posteriori HMM λ . Now we can write the re-estimation equations as follow.

$$\overline{\pi_i} = \gamma_1(s_i) \tag{3.15}$$

$$\overline{A}_{s_i s_j} = \frac{\sum_{t=2}^{T} \xi_{t-1}(s_{i,} s_j)}{\sum_{t=2}^{T} \gamma_{t-1}(s_i)} \tag{3.16}$$

$$\overline{B}_{s_i}(z_k) = \frac{\sum_{t=1, o_t = z_k}^{T} \gamma_t(s_i)}{\sum_{t=1}^{T} \gamma_t(s_i)} \tag{3.17}$$

With respect to the a priori HMM λ' Eq. (3.15) to Eq. (3.17) can be described as follows.

$$\pi_i = \text{expected number time visiting state } s_i \text{ at time } (t = 1)$$

$$\overline{A}_{s_i s_j} = \frac{\text{expected number of transitions from state } s_i \text{ to state } s_j}{\text{expected number of transitions from state } s_i}$$

$$\overline{B}_{s_i}(z_k) \frac{\text{expected number of times in state } i \text{ and observing symbol } z_k}{\text{expected number of times in state } s_i}$$

Using discrete probability functions greatly simplifies greatly modeling of the problem. However they have limited representative power. To solve this problem, either a single or mixture of continuous probability distribution functions is used as the observation probability during the training phase. The task of learning is then to learn the parameters of these distributions. The most widely used distribution is the d-dimensional Gaussian distribution or Normal distribution.

3.2.2 Time Series Prediction

Generally, a time series is a number of data points, measured in uniform time intervals and can be denoted by

$$\boldsymbol{x} = \{x_1, x_2, x_3..x_n\},$$

where x_i can be a scalar or vector value. The field of making predictions from an available time series is called *time series prediction* and is an area of research in the field of machine learning. The basic goal of time series prediction is to generate a model of the process under observation, which is able to predict

values that have not yet been measured. Such a model can be global model or local model. Global models describe the relationship between the input and the output values as a single analytical function over the whole input domain. On the other hand, local modeling does not describe the whole physical system in one model, but creates a specific model for a given input. This means that generally not a global model has to be created, but only a model that describes the systems behaviour for a given input.

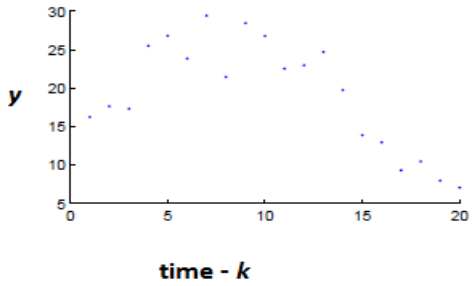

Figure 3.1: Time series plot

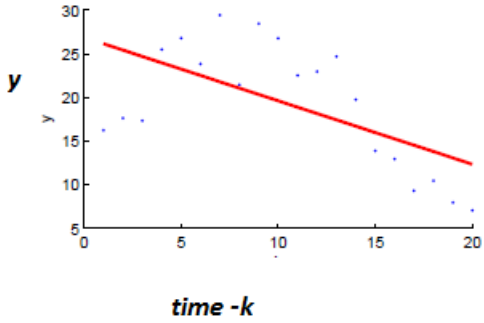

Figure 3.2: Global Model

Because the input for which the prediction has to be performed is only known at the prediction time, local model algorithms are generally lazy learners. This concept of global and local models requires some more clarification. In order to explain it, a simple example is assumed, in which, from a set of training data, the next value shall be predicted. For reasons of simplicity, scalar input and output values are assumed. The training data is shown in Figure 3.1. A linear global model is shown in plots of Figure 3.2 and a linear local model in plots of Figure 3.3.

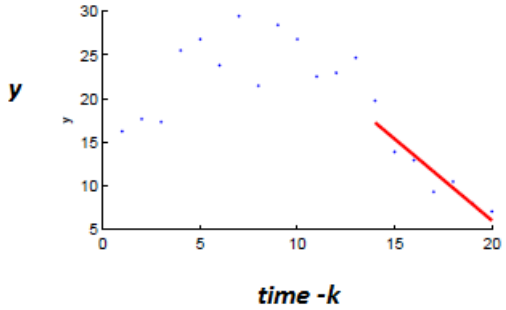

Figure 3.3: Local Model

The goal of local modelling is not to get a model which describes the whole process, but instead to simply give a reasonable output for a given input (the query). A possible local linear model is shown in Figure 3.3. As time independent local models, in which the next value is predicted based on the query, there are several strategies found in literature. One prominent approach is called the nearest neighbour. In this case, in the training data the value which is closest to the query is taken and the value which followed this nearest neighbour is taken as the prediction.

3.2.3 Gaussian Process

Gaussian process (GP) is considered as a supervised machine learning algorithm widely used in different domains in the past decades. Identification of the model of a dynamic system can be done by using GP regression. Such model identified is called as *nonparametric model*, which does not mean that there are no parameters inside the model but model has flexible parameters that can be adapted from the input data. Therefore, a Gaussian Process(GP) is completely different from the so called a *parametric model* where the parameters involved impose a fixed structure or value in advance upon the model.

A brief introduction to Gaussian Process (GP) is presented here. For a comprehensive report on GP, kindly refer [112]. Gaussian process provides a non-parametric Bayesian approach towards regression problems. It can capture relations between inputs and outputs by utilizing a theoretically infinite number of parameters and letting the data decide upon the level of complexity through the means of Bayesian inference.

Formally, a Gaussian Process can be defined as:

Definition: A Gaussian Process is a collection of random variables, any subset of which has a joint normal distribution.

A Gaussian process is completely defined by its mean function and covariance function. We will write

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \sum) \tag{3.18}$$

Given a set of data $D = \{\mathrm{x}_i, y_i\}_{i=1}^{n}$ where $\mathbf{x}_i \in R^d$ and $y_i = f(\mathbf{x}_i) + \varepsilon, \mathrm{y}_i \in R$, we want to model the input-output relationship by using a Gaussian process with mean function $m(\mathbf{x})$ and covariance function $\sum$. In most of the applications, the mean function is set to zero, and any covariance function generating a positive definite covariance matrix is used.

In order to make prediction about a new input $\mathbf{x}_* \in R^d$, the joint distribution of the training outputs $\mathbf{f}$ and the test output f_*. Eq. (3.19) is conditioned on the observations and the expected value is obtained according to Eq. (3.20) and variance of the prediction according to Eq. (3.21)

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,\mathbf{x}_*) \\ K(\mathbf{x}_*, X) & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right) \tag{3.19}$$

$$E\left[f_* \mid X, x_*, \mathbf{f}\right] = K(\mathbf{x}_*, X)\left[K(X,X), \sigma_n^2 I\right]^{-1}\mathbf{f} \tag{3.20}$$

$$\mathrm{cov}(f_*) = k(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, X)\left[K(X,X) + \sigma_n^2 I\right]^{-1} K(X, \mathbf{x}_*) \tag{3.21}$$

where X represents the matrix of training inputs, K denotes the covariance matrix which is obtained by pairwise evaluation, $\sum_{ij} = \mathrm{cov}(y_i, y_j) = \mathrm{cov}(f(\mathbf{x}_i), f(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$ of the covariance function for the given inputs. Writing in shorthand form, we have the predicted value of the process at the new input and variance as

$$\overline{f}_* = k_*^T (K + \sigma_n^2 I)^{-1} y = k_*^T \alpha \tag{3.22}$$

$$\mathrm{cov}(f_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^T \left[K + \sigma_n^2 I \right] k(\mathbf{x}_*) \tag{3.23}$$

Here $k_* = K(X, \mathbf{x}_*)$, $\alpha = (K + \sigma_n^2 I)^{-1} y$ is called prediction vector and y is the vector of training function outputs, and σ_n^2 is the variance of the Gaussian noise ε. The schematic view of GPR is shown in the Figure 3.4. Given the training data and a covariance kernel function, GPR can predict the function value at a test input.

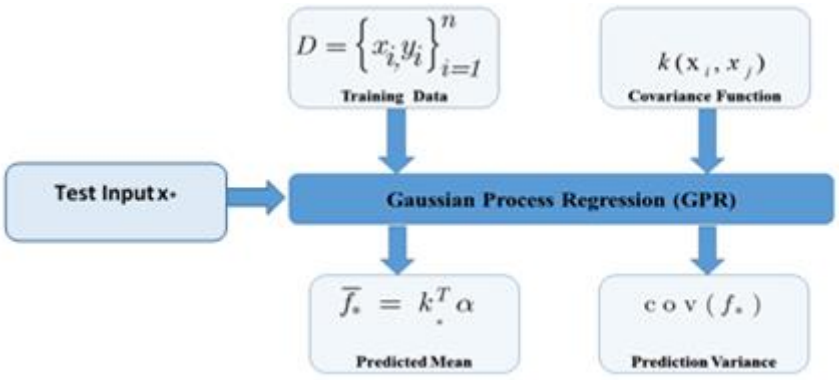

Figure 3.4: Schematic view of GPR

The covariance function chosen is not completely free from parameters and they still have some parameters called hyper-parameters.

Let us accumulate these parameters in θ. We can learn θ from the training data by marginal likelihood optimization. The log-marginal likelihood is defined as

$$\log(y \mid X,\theta) = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2}\log\left|K + \sigma_n^2 I\right| - \frac{n}{2}\log 2\pi \tag{3.24}$$

The gradient of the marginal likelihood with respect to θ can be computed by a gradient optimization technique to minimize the objective function in Eq. (3.24). The implementation of Gaussian process regression in given in Algorithm 3.2 below (reproduced from [112]).

Algorithm 3.2: Gaussian Process Regression

Inputs:

1. Input dataset, $D = \{x_i, y_i\}_{i=1}^n$
2. Covariance function, $\sum_{i,j} = k(x_{i,} x_j)$
3. Signal noise level, σ_n^2

Outputs:

1. Mean, $\bar{f}_*$
2. Variance, $\text{cov}(f_*)$

Steps:

1. $L \leftarrow cholesky(K + \sigma_n^2 I)$
2. $\alpha = L^{T\ -1}\ L^{-1} y$
2. $\bar{f}_* \leftarrow k_*^T \alpha$
3. $v = L^{-1} k_*$
4. $\text{cov}(f_*) = k(x_*, x_*) - v^T v$
5. return $\bar{f}_*$, $\text{cov}(f_*)$

Due it *Cholesky* factorization the time complexity of Gaussian process regression is $O(n^3/6)$.

3.2.4 An Example of Gaussian Process Regression

We present here an illustrative example of using a Gaussian process in regression problem. Imagine that we have collected the observations shown in Table 3.1 and that we want to predict the value of y for a new input point x_*

Table 3.1: Observation for regression example

t	x_t	y_t
1	0.9	0.1
2	3.8	1.2
3	5.2	2.1
4	6.1	1.1
5	7.5	1.5
6	9.6	1.2

In linear regression set up, we assume the outputs are a linear function of the inputs with additional noise as the following.

$$y_t = f(x_t) + \varepsilon_i = \beta_0 + \beta_1 x_t + \varepsilon_t$$

Here ε_t is the noise term that follows a normal distribution $\varepsilon_t \sim N(0, \sigma^2)$. We can write the above equation in matrix notation as the following.

$$y_t = \mathrm{x}_t^T \mathrm{w} + \varepsilon_i$$

The vectors are defined as

$$\mathrm{x}_t = \begin{bmatrix} 1 \\ x_t \end{bmatrix}, \qquad \mathrm{w} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

To predict the output for the new point x_*, we need to estimate the weights from the past observations.

$$\mathrm{X}_t = \begin{bmatrix} 1 & 0.9 \\ 1 & 3.8 \\ : & : \\ 1 & 9.6 \end{bmatrix}, \qquad \mathrm{y}_t = \begin{bmatrix} 0.1 \\ 1.2 \\ : \\ 1.2 \end{bmatrix}$$

Now we can rewrite the mean function in Eq. 3.22 as the following.

$$f_* = \sum_{i=1}^{t} w_i k(\mathrm{x}_i, \mathrm{x})$$

Here each x_i is a previously observed input value in X_t and the weights are collected in the vector, $\mathrm{w}=(\mathrm{K}(\mathrm{X}_{t,} X_t) + \sigma_\varepsilon{}^2 I)^{-1}\mathrm{y}_t$. What this equation tells us is that Gaussian process regression is equivalent to a linear regression model using the basis function k to project the inputs into a feature space. To make predictions, every output y_t is weighted by how much similar the corresponding input x_t to the to be predicted point x_* by similarity induced by the kernel. This results in a simple weighted sum to make the predictions for new points. The posterior predictive mean then is a linear combination of the features. Therefore a conceptually infinite parameter space ultimately boils down to a finite sum when making the predictions. This sum depends only on the chosen Gaussian process kernel k and the data D_t observed thus far.

This is why Gaussian process regression is referred to as a non-parametric technique. It is not the case that this regression approach has no parameters. Actually it has theoretically as many parameters w as there are observations. However, in making predictions, we only use a finite sum over all past observations.

Table 3.2: Example of generating a prediction using a Gaussian process with a radial basis function kernel. $w_i = (K(X,X) + \sigma_\varepsilon{}^2 I)^{-1} y_i, x_* = 3$

t	x_t	y_t	w_t	$k(x_t, x_*)$	$w_t k(x_t, x_*)$
1	0.9	0.1	0.51	0.38	0.19
2	3.8	1.2	-3.88	0.87	-3.37
3	5.2	2.1	13.3	0.34	4.53
4	6.1	1.1	-12.55	0.12	-1.48
5	7.5	1.5	5.83	0.01	0.06
6	9.6	1.2	-0.34	0.00	0.00
				$\sum_{i=1}^{6} w_t k(x_t, x_*)$	-0.06

Assuming a radial basis function kernel $k(x,x') = \sigma_f^2 \exp(-\frac{(x-x')^2}{2l^2})$ with length scale $l = 1$, and observation variance $\sigma_\varepsilon{}^2 = 0.01$ the details for generating a prediction for $x_* = 3$ can be worked out as in Table 3.2.

3.3 Proposed Models

3.3.1 Definitions

Time Horizon: *Total time duration in the past over which the service user will analyze the trustworthiness of a service provider in order to make a trust-based decision for future interaction.*

The time horizon is a positive value representing years or months or days or seconds depending on the user. The idea is that for making a trust-based decision for a future interaction, a service user may like to analyse the behaviour of the service provider over previous years or months etc.

Time Slot: *A finite duration of time in the time horizon over which the direct trust value or recommended trust values are collected from the direct experiences or from other intermediaries and then aggregated into a single value for analysis of its dynamic nature of trustworthiness of the service provider.*

Time slot allows us to divide the time horizon into equidistance intervals. For example, in a time horizon of one year, we can divide it into days, giving a sequence of 365(6) equidistance intervals. For each interval, using an aggregation method, a single value of trust value can be generated. This process will generate a time series of trust values.

Time Point: *The time of interaction between a service user and the service provider and at which the trustworthiness value based on the outcome(s) of the interaction is recorded by the service user*.

Time point will help to identify which past interaction(s) falls under a given time slot.

Direct Trust: *A measure of trustworthiness of the service provider in a given context and at a given time point established by a service user from the previous interactions with this provider.*

Recommended trust: *A measure quantifying the trustworthiness of the service provider in a given context and at a given time point as communicated by an intermediary.*

Reputation Trust: *A numerical value representing the truthfulness of the recommended trust provided by an intermediary.*

3.3.2 Architecture and Data Structures

In our model, we assume a reasonable size network of service users and a single service provider (Figure 3.5.). Services from the service provider are accessed by the service users. They record the trustworthiness values of the service provider that they have previously interacted with.

Service users also communicate to each other about (1) their experience with the service provider and (2) their experience with other service users in soliciting recommendation trust. This communications serve as source of third party feedbacks. The second communication is an important one because using the information from this channel one can validate the trustworthiness of the third source of information i.e. advisor used in our model. Each service user in the network maintains the following information in its local database.

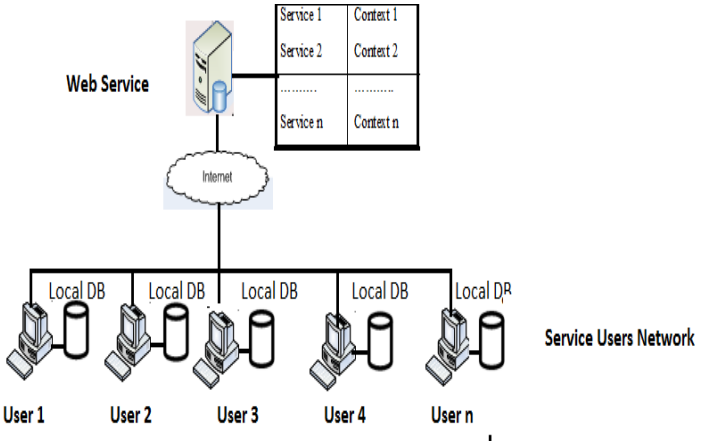

Figure 3.5: Single Service Access Architecture

Direct Trust Table: It stores the trustworthiness values of all service providers that a service user has interacted with in the past.

Reputation Trust Table: It stores the reputation trust values of all other service users from whom recommended trust values of service providers have been collected.

The structures of these tables are given in the following Figure 3.6.

Provider Id	Trust Value	Context	Time

(a) Direct Trust Table Entry

Service User Id	Reputation Trust	Time

(b) Reputation Trust Table Entry

Figure 3.6: Table Entry format

Using entries in the Direct Trust table, a service user can do the prediction of trustworthiness of a service user in a future time slot. We call it as direct trust prediction. Secondly, service user also can answer a reputation query from other service users using this table. We consider each user as having an individual level of trustworthiness determined by the quality of its answer to such query. Reputation trust value in the reputation trust table reflects this level of trustworthiness. Its numeric value can be used by a service user to decide whether other witness service users are within its reputation range. Reputation trust value of a witness service user is calculated from the difference between:

- Recommended trust value communicated by the witness agent about the target service provider.
- The actual trustworthiness value obtained on interaction with the target service provider.

We use the following simple mechanism for reputation trust calculation.

- If actual trustworthiness value is greater than or equal to the recommended trust value, reputation trust is set to 1; otherwise it is set to -1.

Context information in direct trust table and reputation trust value in the reputation trust table will enable us to categorize a service user as one of the intermediary types given in Table 3.3 in section 3.3.3.

3.3.3 Trust Intermediaries

A system based on the knowledge of trustees' past behaviour could sustain trust and a consistent degree of trustworthiness provided that information was sufficiently reliable. Following argument in [120], it is crucial to understand the role of trust intermediaries who have positions and interests either analogous or different from the trustors. When positions and interests are aligned, trustors are expected to seriously consider the opinion of the intermediaries so that their

decisions will reflect reputational information available. Intermediaries may be either an advisor or a guarantor. Trustor trusts the advisor's judgment which leads him to place trust on the potential trustee. So always there is an element of risk involved while taking the recommendations from advisors. The trustor, however places trust on a guarantor's performance and integrity just as the later does in that of the potential trustee. So we claim the following in our model:

- A guarantor intermediary is one whom the trustor has already established a recommendation trust relationship and from whom the opinion of the trustee's behaviour can be elicited.
- An advisor intermediary one whom the trustor has not established any recommendation trust relationship earlier, yet it can provide an opinion of the trustee's behaviour.

Our opinion is that a guarantor is already known to the trustor from their past exchanges of recommendations while an advisor is an unknown one. Again if a guarantor is within the reputation trust range of the trustor i.e. its reputation trust value is above a threshold, then we called it a *known* and *trusted* guarantor.

3.3.4 Information sources

According to [120], the placement of trust on a trustee is essentially based on the information available to a trustor from three sources:

1. Trustor's assessment of trustee's performance. (Direct source)
2. Recommendations from other intermediaries who have a position similar to the trustor's and similar interest on the placement of trust.
3. Recommendations from other intermediaries who do not have a position similar to the trustor's and do not have the similar interest.

Source (1) passing through no intermediaries at all, will be most likely to lead to a correct assessment. Source (2) often leads to the decision about trust as made by other intermediaries whose judgment was trusted. Finally source (3), provides the independent evidence of the decision.

In our prediction model, "position and interest" similarity is decided from the context information. We will explain with an example.

Table 3.3. Categorization of Trust intermediaries

Name	Know & Trusted	Position & Interest	Type of Source
Type I Guarantor	yes	$ConSim(c_i, c_j) = 1$	2
Type II Guarantor	yes	$0 < ConSim(c_i, c_j) < 1$	3
Type I Advisor	no	$ConSim(c_i, c_j) = 1$	2
Type II Advisor	no	$0 < ConSim(c_i, c_j) < 1$	3

Let c_i, c_j be two possible contexts of interaction with a service provider. Let *A* be a service user who wants to interact with the service provider in future and let c_i be its context. Let *B* be another service user who already has interacted with the same provider in the context c_j and has established an opinion about the provider. Let $ConSim(c_i, c_j) \in [0, 1]$ be an operator which evaluates the similarity of any two contexts with a value of 1 meaning exact match and 0 meaning exact mismatch. So, by $ConSim(c_i, c_j) = 1$ we mean that the *A*'s present position and interest is analogous to that of *B* 's past experience. *A* can directly utilize *B*'s opinion in its analysis of provider's past behaviour. By $0 < ConSim(c_i, c_j) < 1$, we mean that *A*'s position and interest is not aligned completely to that of *B.* In this case, the opinion of *B* can still be utilized in *A*'s analysis following the *transferability* property of trust and reputation. With this explanation, we can formulate our trust intermediaries as shown in Table 3.3.

Selection of intermediaries is based on $ConSim(c_i, c_j)$. There are many such functions used in the literature [7]. Here we explain three popular approaches- ontology tree, key word based modeling and task based modeling to illustrate the meaning of context similarity.

In ontology tree based approaches, the contexts are represented in a context ontology tree hierarchical structure. Each node in this tree represents a context.

A node is split into two lower level contexts and the low level contexts are sub-context of the node.

In [148] similarity between two contexts is computed by the distance between to node in the context's ontology tree:

$$ConSim(S_1, S_2) = \frac{1}{Dist(S_1, S_2)}$$

Here, the distance of two nodes is defined as the least number of intermediate nodes for one node to traverse to another node. For example, in Figure 3.7 which shows services ontology tree, service S_1 and S_2 has a distance of 3.

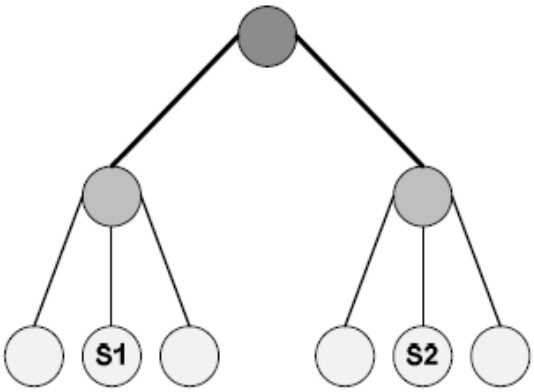

Figure 3.7: Services in context ontology tree [148]

In another similarity computation method for contexts based on ontology [149], the similarity between two nodes is calculated as the ratio between the number of shared nodes from the source node and the sink node to the root node, and the total number of nodes from the source and the sink to the root node. For example in Figure 3.7 service S_1 and S_2 has a distance of 3/5.

Second common approach is to use combination of keywords for context representation. Each keyword is referred to a different context and by ensemble the keywords the result collection is a context. In [66] they considered a file-server application having three types of services: upload PDF file with keywords $\{write, pdf, file\}$, upload DOC file with keywords $\{write, doc, file\}$

, and login with keywords $\{LoginInfo, userName, passWD\}$ Based on set theory, the similarity between two contexts, S_i and S_j, with their individual keywords sets, $K(S_i)$ and $K(S_j)$, is defined as the following.

$$ConSim(S_1, S_2) = \frac{K(S_i) \cap K(S_j)}{K(S_i) \cup K(S_j)}$$

Third common approach is based on task. Here, each task is composed of several sub-tasks which are known task's aspect or task's attribute. An aspect is the smallest element of a task which describes a special attribute of it. In [151] they worked on several tasks such as: "Tom is wondering about trusting Bob to guide him in London when it is stormy". Here, the task is model as: Location: London, Weather: stormy, Subject: guide. As it may be guessed the task's aspects are: Location, Weather and Subject. In [150], the similarity between any two tasks S_i and S_j, is defined by

$$ConSim(S_i, S_j) = 1 - \frac{1}{n}\sum_{k=1}^{n}\left|S_{i,k} - S_{j,k}\right|$$

Here n is the number of task attributes, $S_{i,k}$ is the kth attribute of task S_i and $S_{j,k}$ is the kth attribute of task S_j

3.3.5 Trust Equation

During the analysis of behaviour of a service provider, a service user calculates provider's final trust based on the history of direct interactions and/or recommended trust values obtained from other intermediaries. So the final trust of a service provider A over a context c_i at a future time point t is evaluated from the following relation.

$$T(A, c_i, t) = \lambda * T_d(A, c_i, t\ ) + (1 - \lambda) * T_r(A, c_i, t) \tag{3.25}$$

where $T_d(A,c_i,t)$ is the direct trust value obtained from direct interaction history stored in the direct trust table, $T_r(A,c_i,t)$ is the indirect trust estimated from the recommended trust values obtained from other intermediaries, and λ is the weighting factor to decide the importance between these two trust values.

3.3.6 Direct Trust Calculation

To calculate the direct trust value, the service user must decide the time slot over which all its direct interactions with the service provider must be examined. For example, if the future time point of interaction is next month, then the time slot can be the current year. If the service user already had directly interacted with the service provider in the identified time slot, we proposed to evaluate $T_d(A,c_i,t)$ as

$$T_d(A,c_i,t) = \frac{1}{n}\sum_{i=1}^{n} e^{-\nu(t-t')} * conSim(c_i,c_j) * T_{-d}(A,c_j,t') \tag{3.26}$$

where $T_{-d}(A,c_j,t')$ is the direct trust value recorded in the direct trust table from a previous interaction with the provider at time point t' and context c_j, $conSim(c_i,c_j)$ allows the transfer of trust from similar context, $\nu \in [0,1]$ takes care of the trust dynamics over time, n is the number of all previous direct interactions recorded.

While using Eq. (3.26), it may happen that the service user has no previous interactions in the identified time slot. This is true from two possibilities: (1) all previous interactions were in the previous time slots, *for example in the previous years,* and (2) the provider is a complete stranger. In this scenario, our service user has to choose one or both of the following:

Choice 1: Calculate the final direct trust value $T(A,c_i,t)$ only from $T_r(A,c_i,t)$ component of Eq. (3.25).

Choice 2: Use the prediction mechanism presented in the **Section 3.3.8** to get an estimate $\hat{T}_d(A,c_i,t)$ by considering a larger time horizon.

With our direct trust calculation mechanism, we give the procedure for direct trust calculation in the following algorithm.

Algorithm 3.3: Direct Trust

1. Decide the time point t and context c_i of future interaction.
2. Decide the time slot over which the analysis of behaviour of the service provider is to be done.
3. If previous interactions are available in the identified time slot then
4. Calculate $T_d(A,c_i,t)$ from Eq.(3.26).
5. else
6. Case based on Choice
   i. Choice is 1: Set $T_d(A,c_i,t)$ to 0.
   ii. Choice is 2: Estimate $\hat{T}_d(A,c_i,t)$.
7. End Case
8. End If

3.3.7 Indirect Trust Calculation

The service user first seeks the recommendation trust from the intermediaries by submitting a recommendation trust query specifying time slot and the Id of the service provider. All the intermediaries, who have previously interacted with the service provider, in some context and within the specified time slot, reply back to querying service user with a recommended trust value for the target service provider. After receiving all the replies, our service user will categorized the values as coming from the types of intermediaries mentioned in Section

3.3.3. We use the Beta distribution to select guarantors and also to weight the feedback from advisor. We measure a quantity called reputation range measure of every replying intermediary as

$$R = \frac{\#p}{\#n + \#p} \tag{3.27}$$

where $n\#$ and $\#p$ is the number of negative entries and positive entries respectively, against each replying user, in the reputation trust table of our service user. If $R \succ R_{th}$ then a replying intermediary is said to be within the reputation query range of the querying service user and hence it is termed as guarantor. All other intermediaries for which there is no entry in the reputation trust table are considered as unknown and hence they are termed as advisors. To solicit feedback from them, our querying agent must measure their reputation trust indirectly. First a reputation trust query must be forwarded to all its guarantors, identified in the previous step, by specifying the name of the advisor. Each guarantor, if they know, the targeted advisor will return a pair $(\#n, \#p)$ from their reputation trust tables. From these pairs, a final reputation range measure of the targeted advisor will be calculated as

$$Tot(\#n) = \sum_{i=1}^{ng} (\#n)_i \tag{3.28}$$

$$Tot(\#p) = \sum_{i}^{ng} (\#p)_i \tag{3.29}$$

$$R_a = \frac{Tot(\#p)}{Tot(\#p) + Tot(\#n)} \tag{3.30}$$

where ng is the number of guarantors identified. If $R_a \succ R_{ath}$ then the feedback from the targeted advisor will be solicited.

Having devised the mechanism for identification and selection of intermediaries, we explain the calculation of $T_r(A, c_i, t)$. It is calculated as

$$T_{rg\_tot}(A, c_i, t) = \frac{1}{ng} \sum_{i=1}^{ng} R_i * e^{-\nu(t-t')} * conSim(c_i, c_j) * T_{rg_i}(A, c_j, t') \tag{3.31}$$

$$T_{ra\_tot}(A,c_i,t) = \frac{1}{na}\sum_{i=1}^{na} R_{a_i} * e^{-\nu(t-t')} * conSim(c_i,c_j) * T_{ra_i}(A,c_j,t') \quad (3.32)$$

$$T_r(A,c_i,t) = 0.5 * [T_{rg\_tot}(A,c_i,t) + T_{ag\_tot}(A,c_i,t)] \quad (3.33)$$

where na is the total number of identified advisors. We summarize the indirect trust calculation procedure in the following Algorithm 3.3.

Algorithm3.4: Indirect Trust

1. Decide the time point t and context c_i of future interaction.
2. Decide the time slot over which the analysis of behaviour of the service provider is to be done.
3. Decide the thresholds R_{th}, R_{ath} .
4. Issue a reputation query specifying the time slot and service provider Id.
5. For each replying intermediary
6. Begin
7. if there exist a record in the reputation trust table
8. Calculate R_i using Eq.(3.27)
9. if $R \succ R_{th}$
10. Mark the intermediary as a guarantor.
11. End If
12. else
13. Mark the intermediary as advisor.
14. End If
15. End Begin
16. End For
17. From all marked guarantors calculate $T_{rg\_tot}(A,c_i,t)$ using Eq. (3.31).
18. For each advisor
19. Begin
20. Issue reputation trust query to marked guarantors.
21. Calculate R_a using Eq. (3.28)-(3.30).
22. if $R_a \succ R_{ath}$
23. Mark the advisor as useful.
24. End Begin
25. From all useful advisors calculate $T_{ra\_tot}(A,c_i,t)$ using Eq (3.32).
26. Calculate $T_r(A,c_i,t)$ from Eq. (3.33)

It may so happen that our service user is unable to obtain the recommended trust values from the intermediaries because none of the intermediaries has interacted with the provider during specified time slot. In such scenario, the analysis is to be done in a larger time horizon to get an estimate $\hat{T}_r(A, c_i, t)$.

3.3.8 Trust Prediction

When our service user cannot measure $T_d(A, c_i, t)$ and/or $T_r(A, c_i, t)$, analysis is to be done in a wider time horizon. We proposed a prediction mechanism using non-linear time series mechanism and Gaussian Process Regression (GPR) [112]. We used Markov models – a Markov chain and a Hidden Markov Model to drive the GPR. We explain the prediction mechanism in the following subsections.

Model 1: Markov Chain model based prediction

The Schematic view of the prediction procedure is given in Figure 3.8. We called it as Model-1.

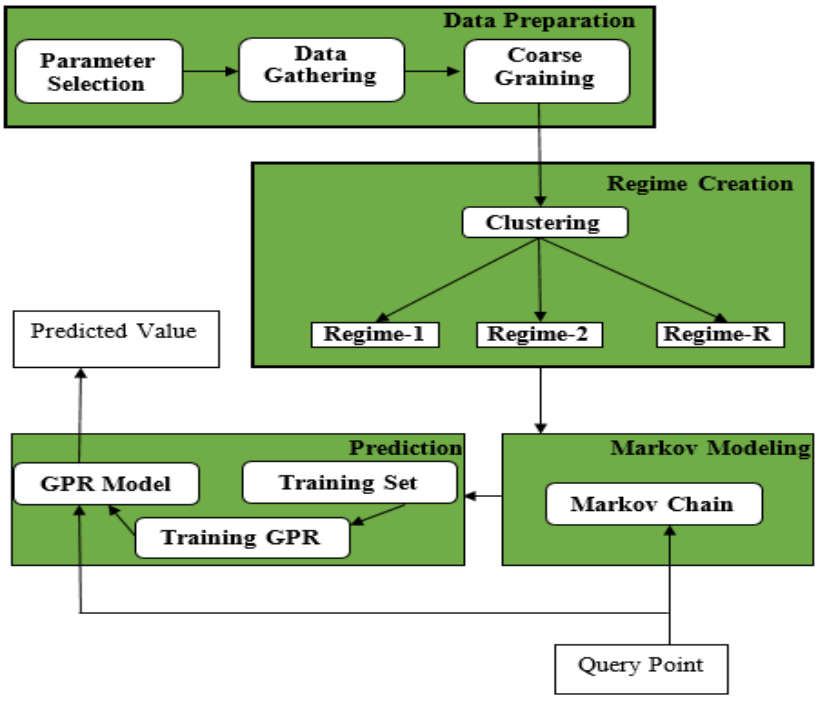

Figure 3.8: Schematic view of Prediction Procedure of Model 1

Data selection phase constructs a coarse grained trust time series. Using a clustering technique the time series is segmented into regimes. A regime represents a region of volatility in the trust time series. Dynamics of regime change over time is captured in a Markov chain model. The Markov model is then used to generate the training data set of a local prediction model based on GPR and the new input (test input) about which the prediction of the future trust value is to be done. In the training phase, the GPR model is fed with training data to evaluate its parameters. The training process finds the optimal values for the kernel parameters by fitting the GPR model to the training data. In the current work, evaluation is performed by the Polak-Ribiere line search optimization method [117]. Once evaluation of kernel parameters is over, the GPR model is considered as "trained", and hence, it is ready to make predictions. In the last phase i.e. the prediction phase, GPR gets as an input the test point about which the prediction of future value should be made. It should be noted that for prediction GPR framework utilizes both the input and the training data simultaneously and provides the prediction. The whole process is explained in detail below.

Data Preparation Phase

Data selection phase collects the direct trust data and recommendations for the construction of the Markov models and training of GPR models for prediction. We explain each activity included in this phase in the following

Activity 1: Parameters Selection.

- Select the time horizon T_h over which the trustworthiness of the service provider is to be analyzed.
- Select duration T of time slot to divide the time horizon into N equal intervals of length $\frac{T_h}{T}$ each.

- Select time spot T_p at which a trust based decision of whether or not to interact with the service provider in a given context is to be made. This will be considered to fall in the immediate next time slot.

Activity 2: Data gathering and Time series formation.

Case 1: Direct trust data.

- Our service user determines if it has the context specific trust information of the service provider for the specified time horizon in its direct trust table. Available data are then distributed into time slot intervals over the time horizon using the time point of each record. In other words, all the interactions with their time points falling in a particular time slot are grouped together. Then in each slot, aggregation method in Eq. (3.8) is applied to generate a single direct trust value. This value is tagged with the interval number i.e. the time slot number. Here we assume that all intervals will receive at least one data point. This can be done by adjusting the slot length.

Case 2: Recommendation trust data.

- Our service user issues a recommendation trust query to all other service users by specifying the time horizon and the Id of the service provider. The replies are collected and distributed into the time intervals over the horizon. Using Eq.(3.14)-(3.16), the data points falling in each interval are converted into a single recommendation trust value. Each value is tagged with the respective interval number.

We call the time series generated in either case as $Y = \{y_t\}_{t=1}^{N}$ where N is the number of intervals in the time horizon and y_t is the direct trust or recommended trust value of each slot. Thus Y becomes a time series sampled with time equal to slot duration.

Activity-3: Coarse Graining of Time Series

In this phase of our prediction procedure, the data required for model making are collected. The trajectory of the time series is first segmented as explained below.

From the field of non-linear time series analysis, there exists a mapping f in the state space satisfying:

$$y_t = f(y_{t-1}, y_{t-2}, ..., y_{t-m}) \tag{3.34}$$

The prediction of the time series is the regression of the trajectory over the observed samples and generating the future value from the reconstructed state space

$$X_t = [y_{t-1}, ..., y_{t-m}] \tag{3.35}$$

Using Eq. (3.36) and Eq. (3.37) the original series $Y = \{y_t\}_{t=1}^{N}$ is converted to series of segments. This may be noted that each time value t has a corresponding pair $[X_t, y_t]$ assigned to it. Therefore we can visualize the original series as a series with $[X_t, y_t]$ as its observed value. We called this new series as coarse grained series of our trust series $D = \{[X_t y_t]\}_{t=m+1}^{N}$.

Regime selection and regime evolution

We define a *regime* as the cluster that contains vectors representing same volatility region of the coarse grained trust time series $D = \{[X_t y_t]\}_{t=m+1}^{N}$.

Regime selection is implemented in the space $\Re^{m+1}$ of the joint velocity of the state vector and the corresponding output - $\left[\Delta X_t, \Delta y_t\right] = \left[X_t - X_{t-1}, y_t - y_{t-1}\right]$.

The velocity vector of the state vector is

$$D_t = \left[y_t - y_{t-1}, y_{t-1} - y_{t-2}, ..., y_{t-m} - y_{t-(m+1)}\right] \tag{3.36}$$

As the instantaneous velocity of the trajectory, $\frac{d\mathbf{X}}{dt}$ is unavailable from the discrete time series, this segmentation operation using D_t intends to enable the velocity based segmentation more precisely. The regime selection is achieved by using fuzzy-C mean clustering [22]. The clustering mechanism calculates the cluster membership degree μ_i as the degree to which D_t belongs to cluster S_i and updates the cluster centers C_i iteratively to minimize the objective function

$$O = \sum_{i}^{R}\sum_{j}^{N}(\mu_i(t))^2 \left\| D_t - C_i \right\| \tag{3.37}$$

where R is the number of clusters. Using the membership degree obtained from the clustering algorithm, each $[X_t, y_t]$ is assigned to a cluster S_i if its membership degree for S_i is maximum. If each cluster now contains vectors representing same volatility region of the coarse grained trust time series $D = \{[X_t y_t]\}_{t=m+1}^{N}$.

The whole time series is regarded as the *evolution of the regimes* over time. In this manner, we look into the trust series at a coarse level. The principle behind our approach is that each regime represents a set of similar behaviour patterns over time horizon T_h. Such regimes must be extracted to learn a set of local models. The number of regimes (of clusters) R is an open parameter to be decided empirically.

Markov Modeling Phase

In this phase, we generate a Markov chain model to represent the dynamic evolution of the regimes. The transition matrix of the chain is constructed in the following manner.

Each observed trust value y_t is associated with a vector $z_t = [X_t, y_t]$ from a particular regime, so we can model the original trust value sequence as a regime transition network using a Markov chain model.

We construct the Markov transition matrix $A_{R\times R}$ to find the probability of changing from one regime to another regime between any two consecutive time slots. This is a way of finding the change in the behaviour pattern of our service provider. First we defined the following:

Inter-regime transition, $S_i \to S_j$: A transition $S_i \to S_j$ is said to occur from time t to time $t+1$ if $z_t \in S_i$ implies $z_{t+1} \in S_j$

Intra-regime transition, $S_i \to S_i$: A transition $S_i \to S_i$ is said to occur from time t to time $t+1$ if $z_t \in S_i$ implies $z_{t+1} \in S_i$

Then we define the regime transition probability of the Markov matrix as

$$a_{S_iS_i} = p(\mathrm{S}_i \to S_i) = \frac{TotOf(\mathrm{S}_i \to S_i)}{\sum_{k=1}^{R} TotOf(\mathrm{S}_i \to S_k)} \tag{3.38}$$

$$a_{S_iS_j} = p(S_i \to S_j) = \frac{TotOf(S_i \to S_j)}{\sum_{k=1}^{R} TotOf(S_i \to S_k)} \tag{3.39}$$

where the $TotOf$ operator counts the total number of transitions from cluster to cluster.

Markov chain model based prediction

Once the Markov transition matrix $A_{R\times R}$ is constructed in the Markov chain modeling phase, we are ready for prediction of the future trust value. The prediction for trust value for future time point is done in the following steps.

Activity 1: Construction of current state vector (V_c)

Current state vector is a vector of size $1\times R$. First for the last time spot N, the associated vector $z_{t=N}$ is identified and its owner cluster V_i is decided. Then the ith entry of the state vector S_c is set to 1 and all other entries are set to 0.

Activity 2: Finding of Next state vector (V_f)

Future state vector is generated by $V_f = V_c * A$.

Activity 3: Training data selection

Using the future state vector, S_f the next regime(s) are identified to select the training data for the predictor GP. Selection is based on the following observation.

Observation 1: The current regime vector $z_{t=N}$ and the next regime vector $z_{t=N+1}$ in the coarse time series will have $(m-1)$ values in common.

This can be clarified from the Figure 3.9. This observation helps in filtering the training set of our GP.

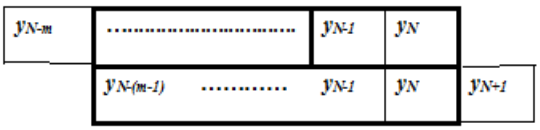

Figure 3.9: Common values between adjacent regime vectors

This may be noted that due to windowing effect in the generation of coarse grained series, the substring $[y_{N-(m-1)},.....y_N]$ of X_N is the state vector for next time spot $N+1$. The value y_{N+1} is our target of the prediction. From the next state vector V_f we can find one or all the regimes (clusters) that immediately follow the regime at time N . Either only the regime with the highest transition probability value in S_f can be taken as the next regime or all regimes with none zero transition probability value can be considered. Former can be considered as *winner takes all* policy of lazy learning while the latter in *ensemble* approach. We use the ensemble approach but with a *filtering* procedure. We generate the training set in the following manner.

Current regime vector $z_n = [X_N, y_N]$ is identified as the last regime vector in the coarse time series. Its owning regime as given by current state vector V_c is called S_N .

All the regimes, whose transition probability in future state vector V_f is non-zero, are identified. They represent the groups of possible behaviour patterns of the service provider in the future time slot. The regime vectors of the form $[X_i, y_i]$ belonging to these clusters are filtered further by examining their time slots. Only the vectors whose time slot comes next to the time slot of any of the vector in S_N are retained in a group. Before applying this filtering we have to remove, z_n from S_N .For each regime vector, $z_i = [X_i, y_i]$ in the group, a modified form of normalized cross correlation between X_N and X_i of z_i is calculated as

$$xC(X_N, X_i) = \frac{\sum_{j=1}^{m} \left[y_{N-j} - \bar{Y} \right] \left[y_{i-j} - \bar{Y} \right]}{\sqrt{\sum_{j=1}^{m} \left[y_{N-j} - \bar{Y} \right]^2 * \sum_{j=1}^{m-1} \left[y_{i-j} - \bar{Y} \right]^2}} \tag{3.40}$$

Where $\bar{Y}$ is the mean of all observations in the trust time series $Y = \{y_t\}_{t=1}^{N}$ For this calculation please refer to Eq. (3.17). We use $\bar{Y}$ as specified because when the state vector X_i or X_N is a flat pattern, $xC(X_N, X_i)$ is undefined if we take the mean of X_i or X_N in calculation of Eq. (3.47).

Only the regime vectors whose $xC(X_N, X_i)$ value is above a given threshold xC_{th} are selected from the group and are retained in the final training data set of our GP. This process is like finding the nearest neighbour of X_N .

Activity 4: Local GPR training

The training set formed in the above process are used to train a GPR model. For this purpose, first we select the kernel of our GP as the square exponential kernel.

$$k(x,x') = \sigma_f^2 \exp(-\frac{(x-x')^2}{2l^2}) \tag{3.41}$$

This kernel is a stationary kernel with the hyper-parameter *l* representing its length scale. The full set of hyper parameters of our GP is accumulated in $\theta = \{l, \sigma_f, \sigma_n\}$. These hyper parameters are learnt by Polak-Ribiere line search optimization method to minimize the objective function in Eq. (3.24). We used the Matlab source code available at http://www.gaussianprocess.org/gpml.

Activity 5: Future Value Prediction

Now we take $x_* = X_N$ as the new point for the prediction of $\bar{f}_*$ from the GP using Eq. (3.5). The prediction model is a local GP model because the regime vectors are selected not from all clusters. Only the clusters with non-zero transition probability from S_f are used. To check the efficiency of our model, we also use a global GP which is trained by using all the regime vectors of the coarse grained trust time series. The query point x_* of this global model is selected from the training set of the local GP model. The selection approach is as given below.

For every pair of regime vectors (z_i, z_j) in the local training set formed above, Eq. (3.40) is extended to calculate correlation between them by including $[y_i, y_j]$ in the calculation. The required equation is given in Eq. (3.42). After calculating the pairwise distances, we select the regime vector which average similarity is the maximum. Let it be z_i . Average similarity is calculated by Eq. (3.43)

$$xC(z_i, z_j) = \frac{\sum_{k=0}^{m} \left[y_{i-k} - \bar{Y} \right] \left[y_{j-k} - \bar{Y} \right]}{\sqrt{\sum_{k=0}^{m} \left[y_{i-k} - \bar{Y} \right]^2 * \sum_{k=0}^{m-1} \left[y_{j-k} - \bar{Y} \right]^2}} \tag{3.42}$$

$$\gamma_{z_i} = \sum_{j=1}^{K} xC(z_i, z_j) \tag{3.43}$$

where *K* is the total number of regime vectors in the training set. By using the average similarity we are, in a way trying to find the regime vector which has occurred most frequently in the past. This is the most likely pattern to follow the current regime z_N .Then $x_* = X_i$ is use as the test point to predict $\bar{f}_*$ from the global GP using Eq. (3.22).

Model-2: Hidden Markov Model based prediction

Model-2 has the same activity flow as model-1. The only difference between the two models is in the Markov modeling and predictions step. We explain only these two phases of model-2 in the following.

Markov Modeling Phase

The schematic view of the model-2 is shown in the Figure 3.10 below.

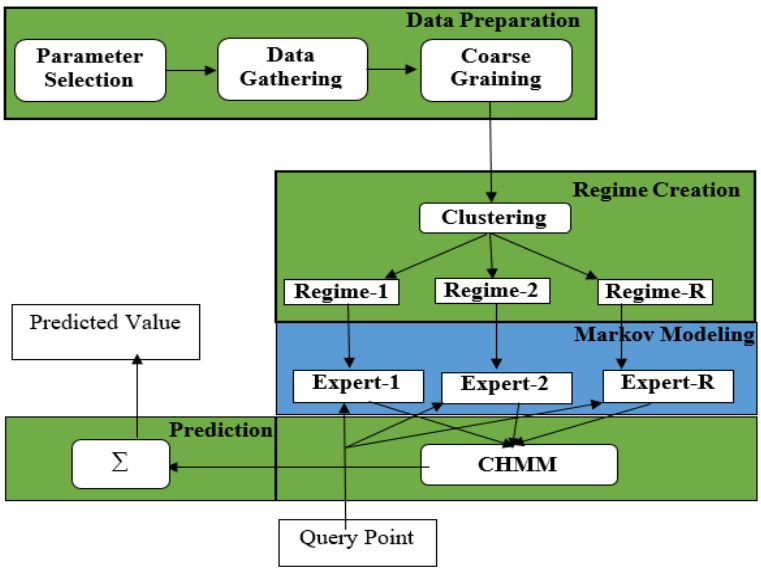

Figure 3.10: Schematic view of Prediction Procedure of Model 2

Activity 1: Local GPR Training

In model-2, we trained one local expert in the form of a GPR for each regime identified. The same GPR training procedure used in model-1 is followed.

Activity-2: Training of CHMM

Using the trained local experts, a HMM model with the emission probability given by Eq.(3.49) is trained to model the evolution of the coarse grained series $D = \{[X_t y_t]\}_{t=m+1}^{N}$. We called it *Coarse Grained Hidden Markov Model* (CHMM). We explain the CHMM model training and prediction in the following.

Definition of CHMM:

Formally, a discrete time CHMM, $\lambda = (S, A, B, \pi)$ is defined by the following elements:

- A set $S = \{s_1, s_2, ..., s_R\}$ of (hidden) states. States correspond to regimes.
- A state transition matrix $A = \{a_{s_i s_j}\}$ of size $R \times R$, where each element $a_{s_i s_j}$ is the probability of transition from state s_i to state s_j.
- The matrix $B = \{\sigma_{s_1}, \sigma_{s_2}, ..., \sigma_{s_R}\}$ where an element σ_{s_i} the variance of the state s_i when the Gaussian error distribution $b_{s_i}(y_t)$ is applied.

$$b_{s_i}(y_t) = \frac{1}{\sqrt{2\pi\sigma_{s_i}^2}} \mathrm{e}^{\frac{\left|(y_t - \hat{y}_{s_i}(\mathbf{x_t}))\right|^2}{2\sigma_{s_i}^2}} \tag{3.49}$$

- Initial state probability distribution $\pi = \{\pi_{s_i}\}$, where an element π_{s_i} is the probability of being in state s_i at the time 1

Thus our CHMM is completely defined by a three-tuple $\lambda = (A, B, \pi)$. The transition matrix A, the vector of variances $B = \{\sigma_{s_1}, \sigma_{s_2}, ..., \sigma_{s_R}\}$, and π are called the parameters of the given CHMM. The number of states $S = R$ is the only hyper-parameter in CHMM model which is to be fixed a priori. It reflects the number of behavioural patterns of our service provider.

Parameter estimation of CHMM:

The parameters of our CHMM estimated according to the maximum likelihood principle using the *expectation-maximization* (EM) algorithm [125] which is *Baum-Welch* algorithm. Our implementation of the *Baum-Welch* algorithm updates these parameters over iterations of two steps of expectation and maximization.

Expectation (E-step)

In the E-step of the r^{th} iteration, the forward variable $\alpha^{(r)}$ and backward variable $\beta^{(r)}$ are calculated from the current parameter estimates according to recurrence Eq. 3.50 and Eq. 3.51. Then the probability $\gamma^{(r)}_{s_i s_j,n}$ of having the successive pair of regimes $q_n = s_i$ and $q_{n+1} = s_j$ given all the data is estimated in Eq. 3.52. Also the posterior probability $\delta^{(r)}_{s_i,n}$ having the regime, given all data, is estimated according to Eq. 3.53.

$$\left.\begin{aligned} \alpha_{s_i,n+1} &= b_{s_i}(y_{n+1})\sum_{j=1}^{R}\alpha_{s_j,n}a_{s_j s_i}; i=1,....,R; \quad n=1,....,N-1 \\ \alpha_{s_i,1} &= \pi_{s_i,1}b_{s_i}(y_1); \qquad i=1,....,R \end{aligned}\right\} \tag{3.50}$$

$$\left.\begin{aligned} \beta_{s_i,\mathrm{n}} &= \sum_{j=1}^{R}a_{s_i s_j}.b_{s_j}(y_{n+1}).\beta_{s_j,n+1}; \qquad i=1,...,R; n=1,...,N-1 \\ \beta_{s_i,N} &= 1 \qquad i=1,...,R \end{aligned}\right\} \tag{3.51}$$

$$\gamma_{s_i s_j,n} = p(q_n = s_i, q_{n+1} = s_j \mid y_1 y_2 ... y_N) = \frac{\alpha_{s_i,n}.a_{s_i s_j}.b_{s_j}(y_{n+1}).\beta_{s_j,n+1}}{\sum_{l=1}^{R}\sum_{k=1}^{R}\alpha_{s_l,n}.a_{s_i s_k}.b_{s_k}(y_{n+1}).\beta_{k,n+1}}, \tag{3.52}$$

$$i, j = 1,....,R; \qquad n = 1,....,N$$

$$\delta_{s_i,n} = p(q_n = s_i \mid y_1 y_2 ... y_N) = \frac{\alpha_{s_i,n}\beta_{s_i,n}}{\sum_{j=1}^{R}\alpha_{s_j,n}\beta_{s_j,n}}, \qquad n = 1,....,N \tag{3.53}$$

Maximization (M-step)

In the r^{th} iteration, the hyper-parameters $\{\sigma_{s_i}\}_{i=1}^{R}$, A and π are re-estimated using Eq.(5.44) and (Eq.3.56) respectively.

$$\sigma_{s_i}^{(r+1)} = \frac{\sum_{n=1}^{N}\left(y_n - \hat{y}_{s_i}^{(r+1)}(\mathbf{X}_n)\right)^2 \delta_{s_i,n}^{(r)}}{\sum_{n=1}^{N}\delta_{s_i,n}^{(r)}}; \quad i = 1,...,R \tag{3.54}$$

$$a_{s_i s_j}^{(r+1)} = \frac{\sum_{n=2}^{N}\gamma_{s_i s_j,n-1}^{(r)}}{\sum_{n=2}^{N}\delta_{s_i,n-1}^{(r)}}; \quad i, j = 1,....,R \tag{3.55}$$

$$\pi_{s_i}^{(r+1)} = \delta_{s_i,1}^{(r)}; \quad i = 1,...,R \tag{3.56}$$

Initial guesses for these hyper-parameters need to be provided to start the iteration. We initialize σ_{s_i} of the i^{th} regime, A and π uniformly.

Future Value Prediction Phase

In this phase, the prediction of the trust value is performed by using trained GPR model(s). The various activities performed during prediction are different from model-1. The prediction for trust value for future time point in this model is done in the following steps.

Step 1: Selection of the query point x_*

We take the last m values of our trust time series as our query point i.e. $x_* = X_{query} = [y_{N-(m-1),}.....y_{N-1}, y_N]$

Step 2: Construction of posterior probability vector (P **):**

Using the observed sequence, $\mathrm{O} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\}$, we run one execution of the Viterbi algorithm 3.1 in our trained CHMM to find the

most likely state sequence at time N. The last state in the state sequence returned by the Viterbi algorithm is the most likely state at time, N. Let this state be s_i. Then we find the s_i^{th} row in the transition matrix A of our optimized CHMM. Let the row be $[p_{s_1}, p_{s_2}, ..., p_{s_R}]$. Each p_{s_j} in this row provides the transition probability of state s_i in time N to the state s_j in time $N+1$.

Step 3: Prediction of future trust value.

Finally, the future trust value is calculated as a weighted average according to Eq. (3.57).

$$T_{final} = \sum_{i=i}^{R} p_{s_i} . f_{s_i*} \tag{3.57}$$

Here f_{s_i*} is the prediction obtained from the i^{th} Local GPR expert.

We formally summarize the prediction steps for both models in the form of algorithms in the following.

Algorithm 3.4: Markov Chain Based Trust Prediction

Inputs:

1. No of regimes, *R.*
2. Regime vector selection threshold, xC_{th}. .
3. Kernel function, cov .
4. Mean function, *mean* .

Outputs:

1. Predicted future trust value, f_*.
2. Prediction variance, var .

Steps:

1. **[Set the time horizon of prediction]**

   $T_h \leftarrow g$etTHorizon()

2. **[Set the length of a time interval]**

   $T \leftarrow g\text{etTDuration}()$

3. **[Set the time point for prediction]**

   $T_p \leftarrow g\text{etTSpot}()$

4. **[Calculate the number of time intervals]**

   $N \leftarrow \text{calcNumInter}(T_h, T)$

5. **[Collect trust/reputation values over the time horizon]**

   $Y \leftarrow \text{genTrustSeries}(T_h)$.

6. **[Obtain model order]**

   $m \leftarrow \text{getModelOrd}()$

7. **[Prepare coarse grained series]**

   $D \leftarrow \text{genCoarsedSeries}(Y, m)$.

8. **[Obtain regimes]**

   $[s_1, s_2, ..., s_R] \leftarrow \textit{fuzzyCmean}(R, D)$.

9. **[Generate Markov chain]**

   $A_{R\times R} \leftarrow \textit{getMarkMatrix}([s_1, s_2, ..s_R])$

10. **[Obtain current state vector]**

    $S_c \leftarrow \text{getCurrState}([s_1, s_2, ..., s_R], R)$

11. **[Obtain next state vector]**

    $s_f \leftarrow \text{getNextState}(S_c, A_{R\times R})$

12. **[Obtain query state vector]**

    $x_* \leftarrow \text{getQPoint}(D)$

13. **[Generate training points]**

    $X \leftarrow \text{getTrainingSet}([s_1, s_2, ..., s_R], x_*, s_f, xC_{th})$

14. **[Train GPR prediction model]**

    $hyp \leftarrow \text{minimize}(mean, \text{cov}, X)$

15. **[Predict future trust value]**

    $[f_*, \text{var}] \leftarrow \text{gp}(hyp, mean, \text{cov}, X, x_*)$

16. **[Return the predicted future trust value and prediction variance]**

    return f_*, var

Algorithm 3.5: CHMM Model Based Trust Prediction

Inputs:

1. No of regimes, R.
2. Kernel function, cov.
3. Mean function, $mean$.

Outputs:

1. Predicted future trust value, T_{final}.

Steps:

1. **[Set the time horizon of prediction]**

   $T_h \leftarrow g\mathrm{etTHorizon}()$

2. **[Set the length of a time interval]**

   $T \leftarrow g\mathrm{etTDuration}()$

3. **[Set the time point for prediction]**

   $T_p \leftarrow g\mathrm{etTSpot}()$

4. **[Calculate the number of time intervals]**

   $N \leftarrow \mathrm{calcNumInter}(T_h, T)$

5. **[Collect trust/reputation values over the time horizon]**

   $Y \leftarrow \mathrm{genTrustSeries}(T_h)$.

6. **[Obtain model order]**

   $m \leftarrow \mathrm{getModelOrd}()$

7. **[Prepare coarse grained series]**

   $D \leftarrow \mathrm{genCoarsedSeries}(Y, m)$.

8. **[Obtain regimes]**

   $[s_1, s_2, ..., \mathrm{s}_R] \leftarrow fuzzyCmean(R, D)$.

9. **[Train one GPR expert for each regime]**

   $[hyp_{s_i} ... hyp_{s_R}] \leftarrow \mathrm{minimize}(mean, \mathrm{cov}, [s_1, s_2, .. s_R])$

10. **[Estimate CHMM Model]**

    $[A_{R\times R}, [\mu_{s_i} ... \mu_{s_R}], [\Gamma_{s_i} ... \Gamma_{s_R}]) \leftarrow paraEstCHMM([s_1, s_2, .. s_R])$

11. [Obtain query point]

$x_* \leftarrow \text{get}Qpo\text{int}(Y, m)$

12. [Obtain mixing probability vector]

$[p_{s_1}, p_{s_2}, ..., p_{s_R}] \leftarrow \text{getPProbs}(\ \mathbf{X}_1,\ \mathbf{X_2, \ldots, X}_N\ ,\ A, B, \pi\ )$

13. [Predict future trust value]

$T_{final} \leftarrow \text{gp}([hyp_{s_i} ... hyp_{s_R}], mean, \text{cov}, [s_1, s_2, .. s_R], [p_{s_1}, p_{s_2}, ..., p_{s_R}], x_*)$

14. [Return the predicted future trust value]

return T_{final}

3.4 Experiments and Results

The objective of the experiments is to compare the performance difference between our proposed prediction models and the existing methods using synthetic data, two real datasets and a simulation test-bed.

We compare our models with two other typical general trust models: Summation model [21] and Bayesian model [145]. We present here a brief explanation of these two models.

The summation model is widely used general model. Commercial services like eBay and specific environment like P2P networks (in the Engentrust [135]) use summation model. The summation model calculates the feedback score as $S = P - N$, where P is the number of positive rating and N is the number of negative ratings. Finally, the reputation score is calculated as $R = P / (P + N)$. The model in [8] computes the trust value according to the beta probability density function (PDF). The QoS value is mapped to one of the $\varpi \geq 2$ levels. Corresponding to the i^{th} QoS level a counter e_i is maintained. After each transaction, QoS experience is mapped to one of the levels and increase the corresponding count by one. Using a Dirichlet-Multinomial model and

Bayesian inference, the reputation score for the i^{th} level is generated after each transaction by using the formula.

$$r_i = \frac{e_i + \alpha_i}{\sum_i^{\varpi} e_i + \alpha_i}$$

Here α_i is the "prior count" of the i^{th} level and is uniformly set to 1 for all $1 \le i \le \varpi$. At the end of any transaction, a user calculates, the trust value as the aggregate of the reputation scores of his/her desired levels according to the formula.

$$t_f = \sum_{i=\upsilon_l}^{\upsilon_h} r_i$$

Here υ_l and υ_h the lowest and highest QoS levels defined by the user. For example, if the QoS attribute, say time t(in seconds) , may be classified as six levels : $t \ge 60(l_1), 48 \le \mathrm{t} \prec 60(l_2), 36 \le t \prec 48(l_3), 24 \le t \prec 36(l_4), 12 \le t \prec 24(l_5),$ and $t \prec 12(l_6)$. For a user, if the minimum and maximum levels are l_3 and l_1 respectively, then a time value greater than or equal to 36 seconds will be considered a good one.

3.4.1 Experiment using Synthetic data

3.4.1.1 Data Generation

We generate a data series of 1000 data points. The next value in the series changes randomly with a factor (next value/current value). We assume a wide range [0.6, 1.4] for the factor so that the models can be tested in a difficult situation. Moreover, the minimum and maximum values of the series are set to be 0.1 and 1, respectively. The generated synthetic data used for the experiment is shown in Figure 3.11. Out of these 1000 data point 500 points are used as the past direct trust or reputation trust series to train the models and remaining 500 points are used in the prediction.

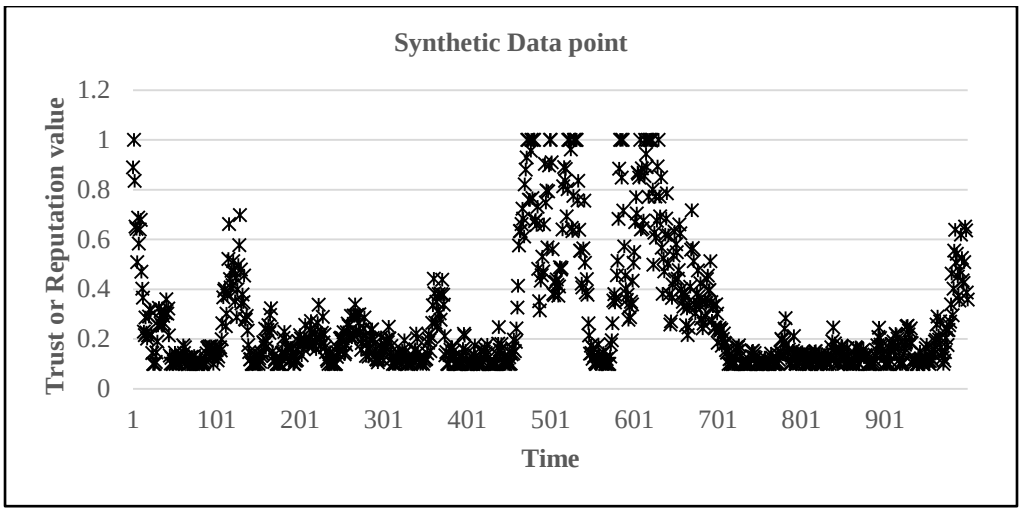

Figure 3.11: Reputation or Trust synthetically generated.

3.4.1.2 Parameters Setting

There are some open parameters that are to be set before the experiment. The model order i.e. lag value *m* is decided by FNN method [24]. The number of clusters, R is selected experimentally by Elbow method [159]. We found $R = 4$.

3.4.1.3 Experimental Results

For comparison, a rolling prediction is performed by allowing the next value of the series to enter into the training set sequentially. In the first prediction step, first 500 time points (i.e. data points) are used as the trust time series to train both global and local GPRs. Using the last regime vector of the training series, the future value at time point 501 is predicted. In the second prediction last 501 points are used to train the models and the value at time point 502 is predicted from the models. The process is repeated till prediction for the last time point 999 is done.

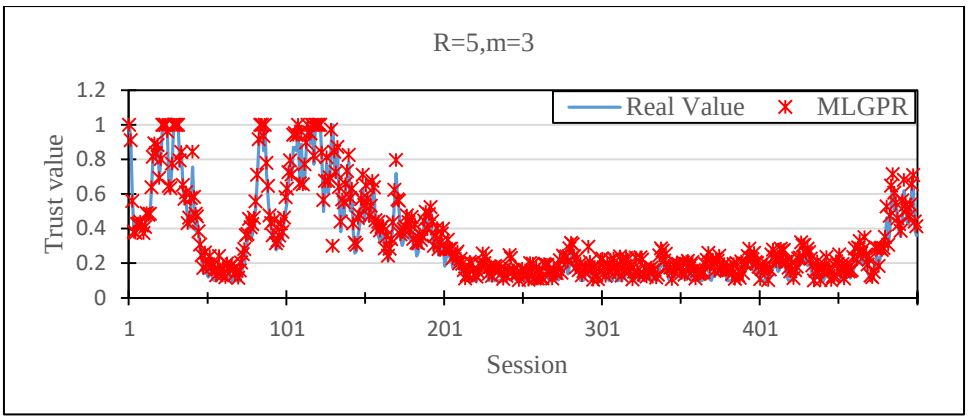

Figure 3.12: Predictions from Markov Chain Local GPR (MLGPR) using square exponential GP kernel**.**

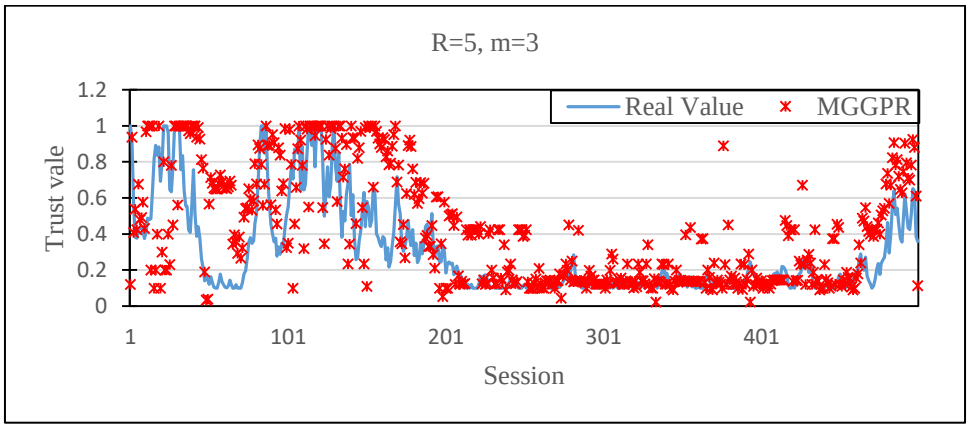

Figure 3.13: Predictions from Markov chain Global GPR (MGGPR) using square exponential GP kernel**.**

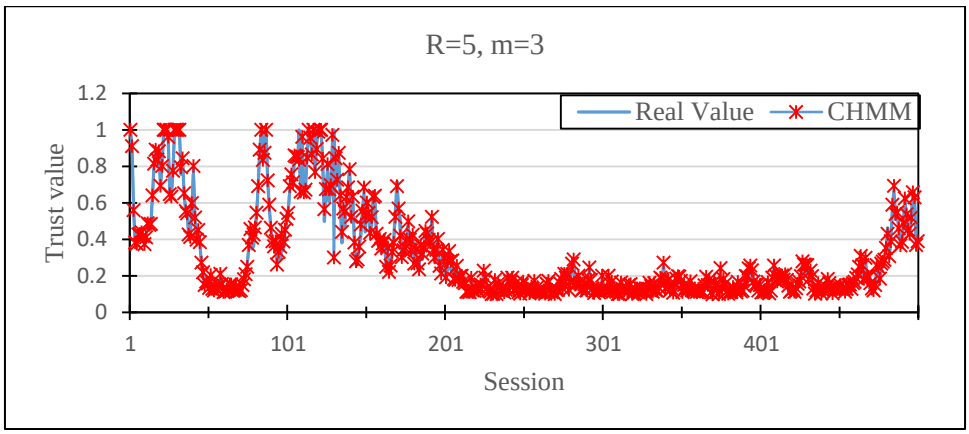

Figure 3.14: Prediction from Coarse Grained Hidden Markov GPR (CHMMGPR) using square exponential GP kernel**.**

The predicted result is shown in the Figure 3.12, Figure 3.13 and Figure 3.14. The blue line represents the actual trust/reputation values. Red stars are the predicted values. We can observed that CHMM based model and Markov chain local GPR (MLGPR) outperform the Markov chain global GPR (MGGPR). Among CHMMGPR and MLGPR, CHMMGPR has better prediction accuracy.

In the next experiment, we compare our models with other two models [21][145]. We run all the models using the same synthetic data. Since the trust values are in the range [0.1 1], we considered any value less than 0.45 as "bad" and value equal to or greater than 0.45 as "good". In this way we obtained only two levels for the implementation of the model in [145]. The levels are $(y_i \prec 0.45)(l_1)$ and $(0.45 \le y_i \le 1.0)(l_2)$. We take $v_l = v_h = l_2$. The cumulative distribution functions of the prediction errors given by the four models are plotted in Figure 3.15.

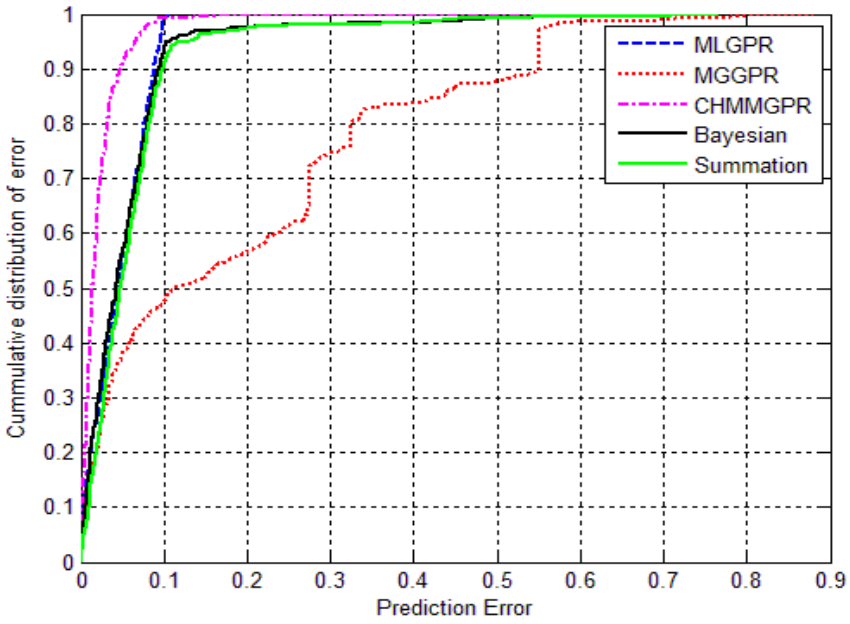

Figure 3.15: Cumulative distribution of prediction errors.

We can see from the plot that CHMM based model (CHMMGPR) and Markov chain based local GPR (MLGPR) are better than the Bayesian and Summation models. The majority error given by CHMGPR and MLGPR are less than 0.1 However due to slow growth in the cumulative, curve, CHMMGPR has more prediction accuracy compared to MLGPR

Errors in Bayesian and Summations models spread to 0.5. Further, we can see from the plot that Bayesian model is slightly better than summation model. The Markov driven global MGGPR does not perform well in the prediction.

3.4.2 Experiment using Real Data set.

3.4.2.1 About the Real Data set

We conducted second round of experiments to test the accuracy of our proposed models by using real life the dataset available at Cloud Armor Project web site http://cs.adelaide.edu.au/~cloudarmor/ds.html. The dataset consists of 10,000+ trust feedbacks given by nearly 7,000 consumers to 113 real-world cloud services.

3.4.2.2 Data Preparation

The Cloud Armor dataset contains values of QoS attributes – *availability, security, response time, accessibility, price, speed, storage space, features, ease of use, technical service,* and *level of expertise*. Along with these attributes, the trust values of the services are also recorded at various time intervals. However most the services have their QoS and trust values recorded for short duration. So we have selected a service "1and1". There are 567 feedback sessions recorded for this service. Not all QoS attributes have their values recorded. Only the values for *availability, price, and technical service* are recorded. Out of these 567 feedback ratings, there are missing values of these QoS attributes on

three cases date September 10, 2009, March 18, 2011 and March 26, 2011. Another 26 cases where all values were entered as zeros were found.

Table 3.4: Details of data set selected from Cloud Armor dataset.

Cloud Service Name	Data points		Time		QoS Attribute Value (continuous)						Trust	
					Availability		Price		Technology Support			
	Before Preprocessing	After Preprocessing	From	To	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
land1	567	538	September 12, 2004	June 19,2012	0	5	5	5	0	5	0	5

We removed these cases from the data set and finally obtained a trust time series 538 time points. The details of the data set selected are given in the **Table 3.4**.

Figure 3.16 visually shows the nature of the whole dataset we used in this experiment. The plot shows the variation of the trustworthiness of cloud service '1and1' with the perceived QoS attributes. The color bar shows the range of trust values. It can be observed that majority of the trust values are distributed in the middle range. There are only 44 cases in which the user obtained the best service from the cloud service provider "1and1" and hence were given a trustworthiness level of 5. In 26 cases, the service failed to deliver and were reported its trustworthiness level as 0. In the rest of the transactions, the cloud service was changing its trust level.

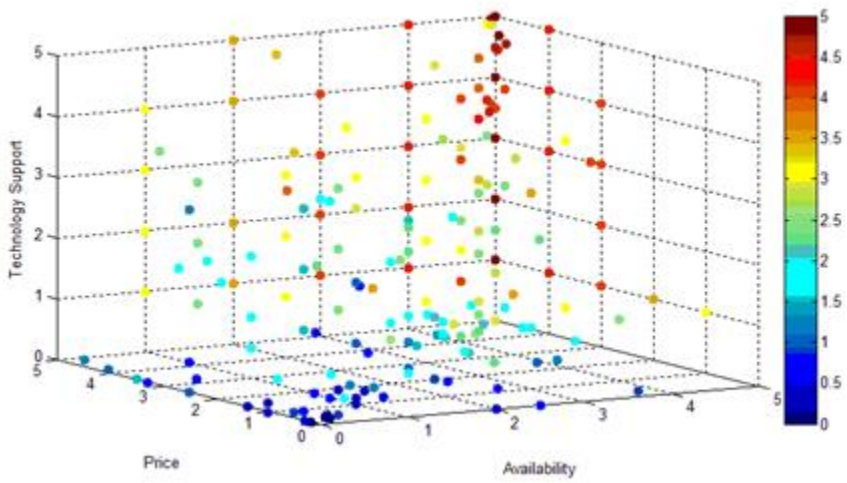

Figure 3.16: 3D plot of the trust values against the QoS attribute for the Cloud service '1and1'

3.4.2.3 Experimental Results

We performed experiments to access the correctness of our proposed models in the prediction of trust value from a given set of <QoS attributes,trust> pairs.

In this experiment, we wanted to predict the trust value using the corresponding QoS attributes. In other word, we want to recover the following functional relationship between the QoS attributes and trust.

$$f(availability, price, TechnologySupport) = trust$$

Since the number of data points is 538, we used a smaller number of points i.e. only first 100 points as the training set. Using Elbow method, the number of cluster $R = 3$ was decided for $xC_{th} = 0.90$. Since the predictor vector is a 3-dimension consisting the values of the three QoS attributes, the model order is fixed at m=1. Using these values of the opened parameters, we first clustered 100 test data points into three clusters. Using these three clusters, we constructed our Markov chain based models – both local and global, and the CHMM based model. Using these models, the predictions for the remaining 438 QoS attributes values were performed.

The prediction from the Markov chain Global GPR show exact matches of the real value and predicted value at 311 cases out of 438 cases. The scatter plot of absolute Error in Figure 3.17 shows this results. Here the cross marks in blue color are the predicted values and red ones are the real values. These are the cases where MGGPR made maximum errors in prediction. The range of prediction error ranges from 0 to 3 as shown in Figure 3.18.

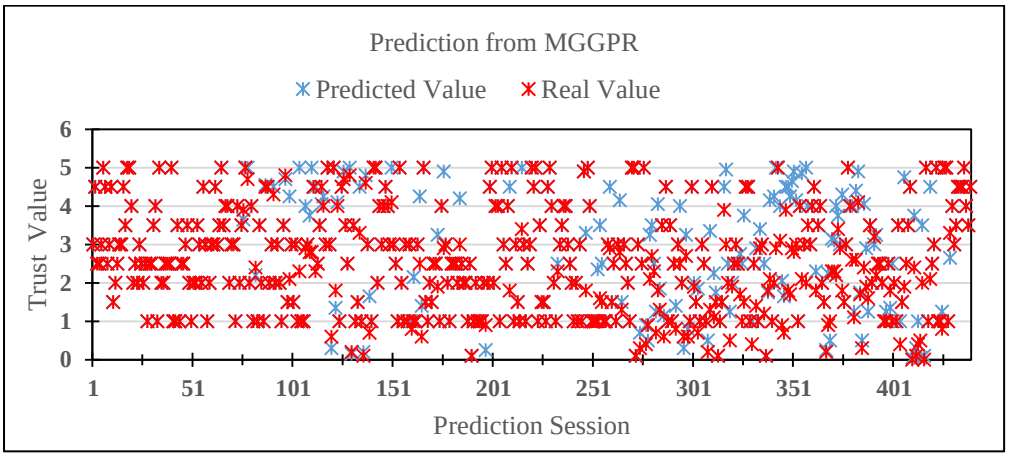

Figure 3.17: Prediction from Markov Chain based Global GPR.

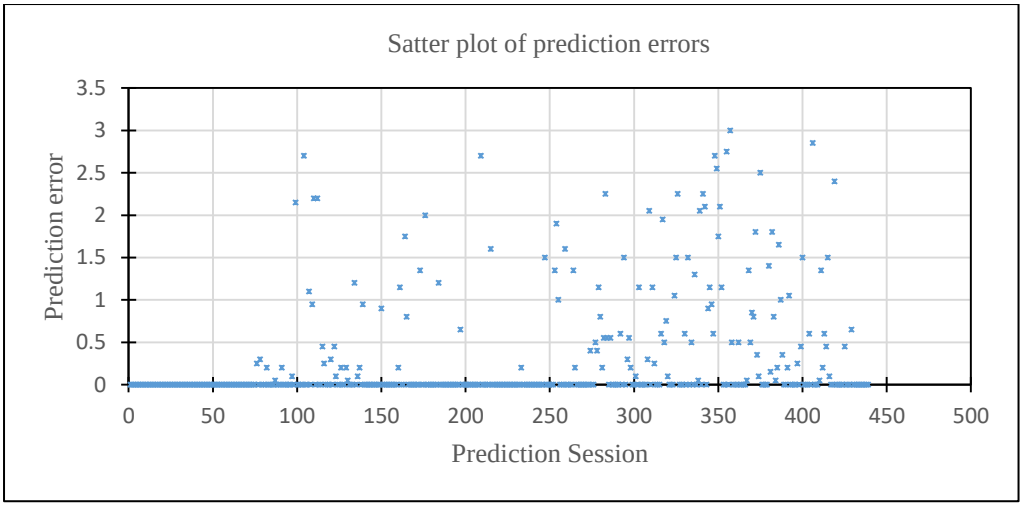

Figure 3.18: Scatter plot of prediction error using Markov chain Global GPR

The performance of the Markov chain based Local GPR is shown in Figure 3.19 and Figure 3.20. The number of exact prediction cases has been increased to 322 i.e. an increase of 2%. Though it is not a significant improvement, it can be

observed that the error margin in the prediction has been reduced to 1.5 as shown in the scatter plot of the errors in Figure 3.21.

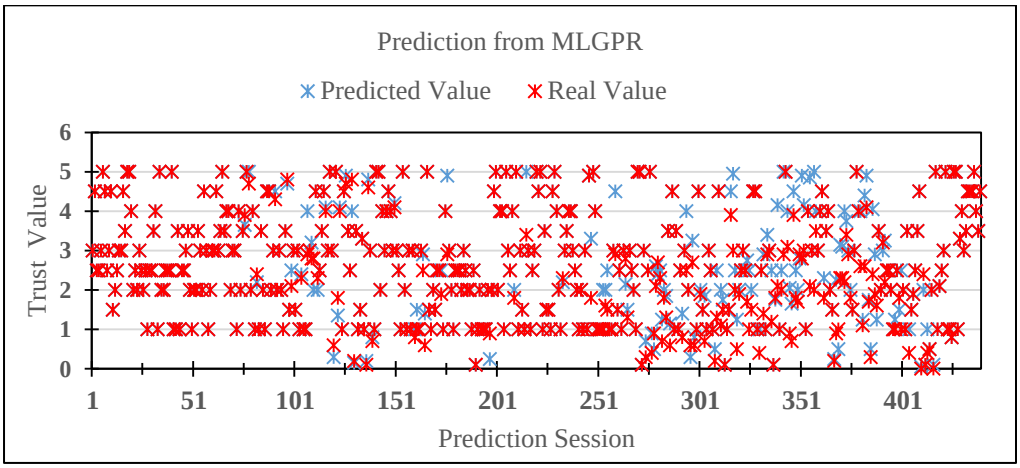

Figure 3.19: Prediction from Markov Chain Local GPR.

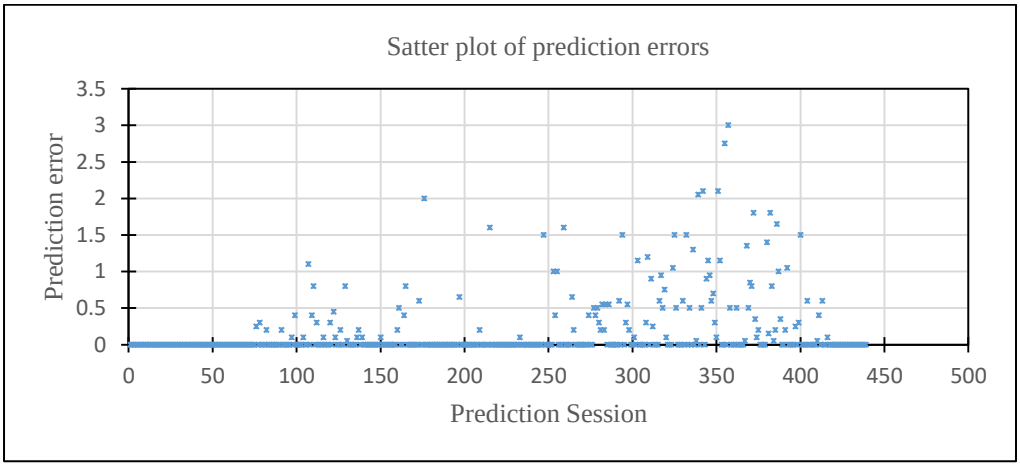

Figure 3.20: Scatter plot of prediction error using Markov chain Local GPR.

The best results are obtained from our CHMM based GPR model. There is an increase of 7% in the number of exact matches between the predicted and real values as shown in Figure 3.21.

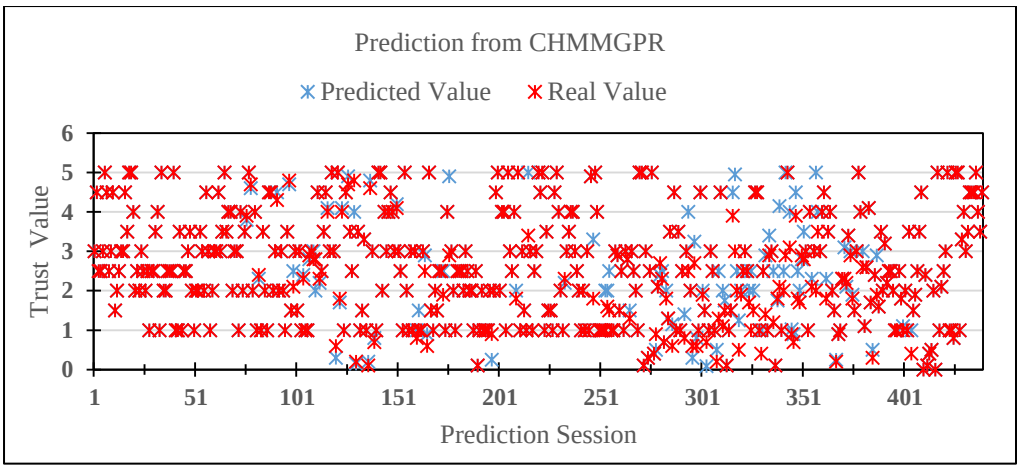

Figure. 3.21: Prediction from Coarse Grained HMM GPR

Moreover, there is a significant reduction in the error margin in the prediction. This can be confirmed from the scatter plot of prediction error for Coarse Grained Hidden Markov Model GPR in Figure 3.22

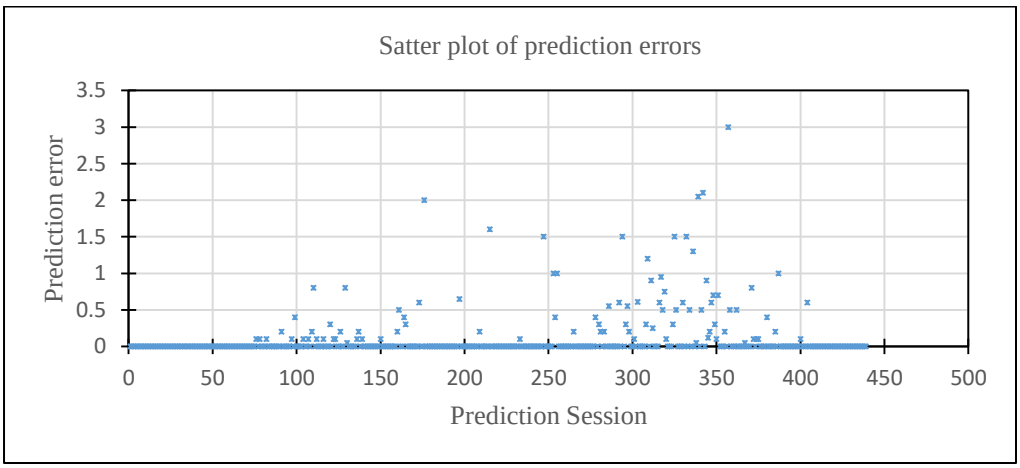

Figure. 3.22: Prediction error from Coarse Grained HMM GPR

3.5 Discussion and Conclusion

We have proposed a trust model based on Markov modeling of dynamic behaviour and the machine learning approach of Gaussian Process. Two models have been proposed. These models provided a framework of using

nonparametric regression technique of Gaussian process in the prediction of direct trust and indirect trust from the past interactions.

The models' prediction accuracy has been investigated experimentally using both synthetic and real life data. The empirical results show promising future of the framework. One of the advantages of the proposed framework is their ability to recover the trust value from the associated QoS attributes. This may be of particular importance whenever we do not have any information of the function that mapped the QoS attributes to the trustworthiness value. What our model can achieve is to learn this mapping from QoS attributes to trust value. Such requirements arise in the prediction of missing values in a particular dataset.

Another advantage is the use of local model over the global model. As trustworthiness bahaviour of a Web service is dynamic, depending on a single global model for future value prediction may not be able to capture this time dependent variations in the predicted value. This fact was experimentally shown by the better accuracy rate of our Markov chain Local GPR and Coarse grained Hidden Markov Model GPR over the Markov chain global GPR. Further it was shown that Hidden Markov Models are more powerful tool to model time variant dynamic processes such as trust and reputation.

Kernels of a GP describe the input data pattern. So an extension can be done to use different GPs with different kernels and use a model selection approach to choose the best model at a given time. This is the main motivation of our work in Chapter 4.