Chapter 4

Adaptive Ensemble of Gaussian Processes for Trust Modelling

4.1 Motivation

In chapter 3, we proposed a framework for trust value prediction for future time slot using Gaussian Process Regression (GPR). We reintroduce here the final trust equation once again.

$$T(A, c_i, t) = \lambda * T_d(A, c_i, t\ ) + (1 - \lambda) * T_r(A, c_i, t) \tag{4.1}$$

By selecting the value of λ, Eq. (4.1) can generate a direct trust model or an indirect trust model or a hybrid of these two. Further we have seen that the model can be used to calculate the direct trust and indirect trust in a given context and during the current time slot using Algorithm 3.2 and Algorithm 3.3 respectively. Here we want to use Eq. (4.1) only for either direct trust or indirect trust based sequential interactions with the service provider. The core of Algorithm 3.2 and 3.3 is the GPR model. Kernel in GPR is the prior belief about the structure of data. So the selection of proper kernel in any regression work is crucial. Further, the trustworthiness of the service provider may change over time and this makes the kernel selection more challenging. A single GPR model with a fixed kernel may not be able to capture its dynamic nature properly. For example, [152] shows that the use of multiple kernels help in determining the underlying patterns while analyzing the CO_2 emission rates (Figure 4.1).

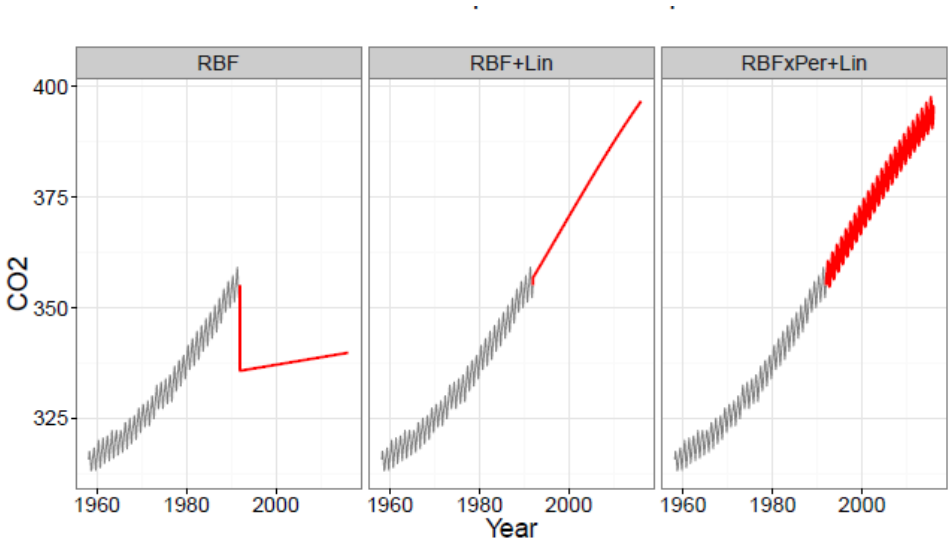

Figure 4.1: Prediction of CO_2 emissions using a Radial basis function (RBF) kernel, sum of a RBF kernel and a Linear (Lin) kernel, sum of a Lin kernel to the product of a RBF kernel and a Periodic (Per) kernel [152]

In the above figure, grey lines show observed CO_2 emissions and red lines show posterior predictions of Gaussian process regressions with different kernels. It was found that a kernel composed by multiplying a radial basis and periodic kernel and adding a linear kernel was able to capture the true nature of the data.

Motivated by this observation, we proposed a better method using ensemble approach for combining different GPR models. We are motivated by the fact that using a kind of model selection approach we can select the model(s) which will best predict the future value at a given time.

4.2 GP Kernels

There are different kernels available for Gaussian process [120]. Here we report some of those kernels used in the machine learning circle and discussed the structure they represent.

4.2.1 Basic kernels and their structures

A kernel is a semi-positive functions of two inputs $x, x^{'} \in R^d$. Gaussian process models use kernel to define the prior covariance between any function values. Many kernels are available, here we discuss commonly used kernels and the types of structures expressible through them.

Square Exponential Kernel:

The *square exponential covariance* (SE) function has already been introduced in Chapter 3, and is also very often referred to as the *Gaussian* covariance function given by

$$k(x, x') = \sigma_f * \exp(-\frac{(x - x')^2}{2l^2}) \tag{4.2}$$

The corresponding Gaussian process will give higher priors to smooth function.

Rational Quadratic Kernel:

The *rational quadratic* (RQ) covariance function is given by

$$k(x, x') = \sigma_f * (1 + \frac{(x - x')^2}{2\alpha l^2})^{-\alpha} \tag{4.3}$$

When $\alpha \succ 0$ RQ is related to SE kernel. For $\alpha \to \infty$ it converges to the SE kernel.

Periodic Kernel:

The *periodic* (PE) covariance function is given by

$$k(\mathrm{x}, \mathrm{x}') = \sigma_f * \exp(-\frac{2}{l^2}\sin^2(\pi\frac{(x - x')}{p})) \tag{4.4}$$

where, p is the period.

Linear kernel:

The *linear* (Lin) covariance function is given by

$$k(x, x') = \sigma_f * (x - c)(x' - c) \tag{4.5}$$

White noise kernel:

The *white noise* covariance function is given by

$$k(x, x') = \sigma_f * \delta(x - x') \tag{4.6}$$

These basic kernels can capture different structures in the input data when they work alone. The Table 4.1 summarizes the structure each of these kernels represents

Table 4.1: Commonly used basic kernels for Gaussian Process

Kernel Name	Structure
$SE(x, x')$	Local Variation
$RQ(x, x')$	Multiscale Variation
$Lin(x, x')$	Linear Function
$PE(x, x')$	Repeating Structure
$WN(x, x')$	Uncorrelated Noise

4.2.2 Combining kernels

Kernels can be combined to create new ones with different properties. This will allow us to include as much high-level structure as necessary into our models.

Given two kernels $k_1(x, x')$ and $k_2(x, x')$, the following will also be valid kernels.

$$k(x, x') = k_1(x, x') + k_2(x, x') \tag{4.7}$$

$$k(x, x') = k_1(x, x') * k_2(x, x') \tag{4.8}$$

Multiplying or adding two positive-definite kernels together always results in another positive definite kernel. Figure 4.2 shows some kernels obtained by multiplying and adding two basic kernels together. Additive structure sometimes allows us to make predictions far from the training data i.e. extrapolation.

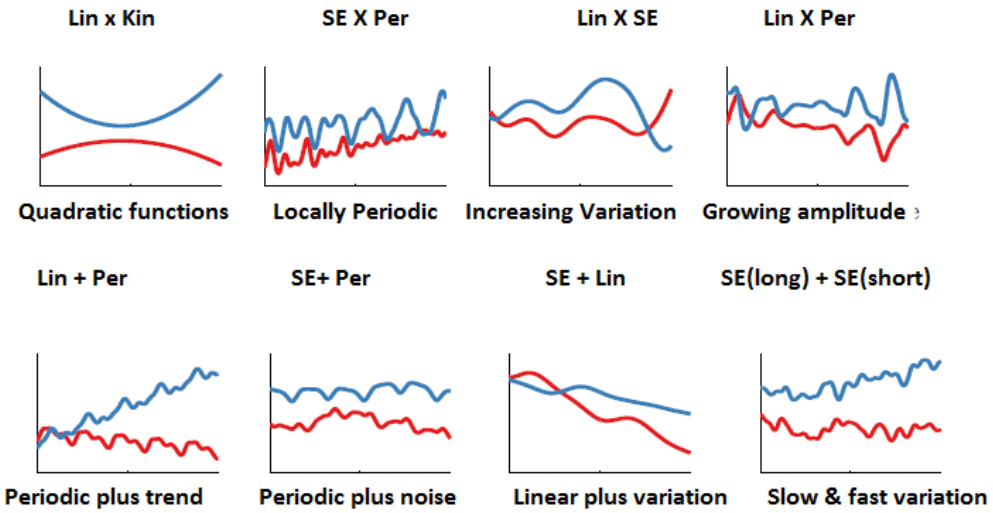

Figure 4.2. Compositions of basic kernels by addition and multiplication and the structures that they represent

4.3 Proposed Model

4.3.1 Model Assumption

In the proposed model, we assumed that service user is locked into a situation of sequential interactions with the service provider. Every time, a service user has to make a trust value based decision of interaction with the service provider by taking into account both the dynamic nature and context specific nature of trust. The trust value can be a direct interaction-based trust value or indirect trust value generated from the recommendations for other intermediaries. Figure 4.3 shows a typical scenario of sequential interactions.

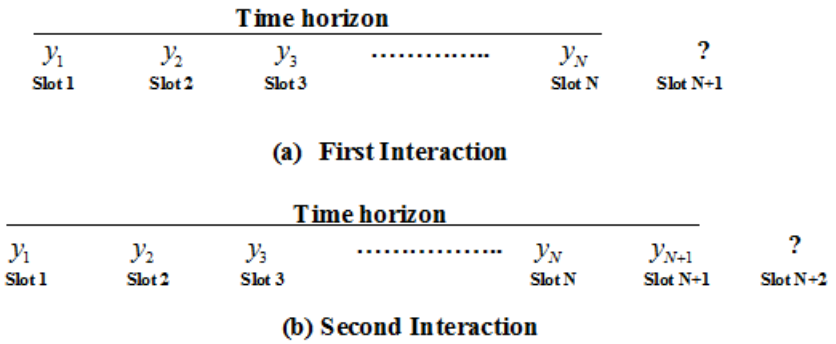

Figure 4.3: Scenario of sequential interactions; (a) first interaction, trust value is to be predicted from the previous values available in a given time horizon, (b) second interaction, trust value is predicted by extending time horizon to the next time spot.

In each interaction, it is required to predict the trust value from a given time horizon. The lengths of time slot and time horizon are to be fixed by the service user. The previous value in each time slot can be a direct trust value generated by using Eq. (3.26) or recommended trust value generated using Eq. (3.33) via a reputation query. After making an interaction based on the predicted trust value, the service user will form a fresh opinion about the trustworthiness of the service provider from the outcomes of the interaction. This opinion will further be used to select a prediction model suitable for next interaction. Prediction model and adaptation procedure are explained in the following section.

4.3.2 Trust prediction framework

The prediction of trust value is done by an ensemble of Gaussian Process Regression models. In section 4.2 we have seen a wide variety of models could be constructed by adding and multiplying a few base kernels together. However selecting the right kernel combination is a hard decision. Here we propose a method in which many GP models work together to increase accuracy in the prediction of trust value. The conceptual framework of the proposed prediction model is given in the Figure 4.4

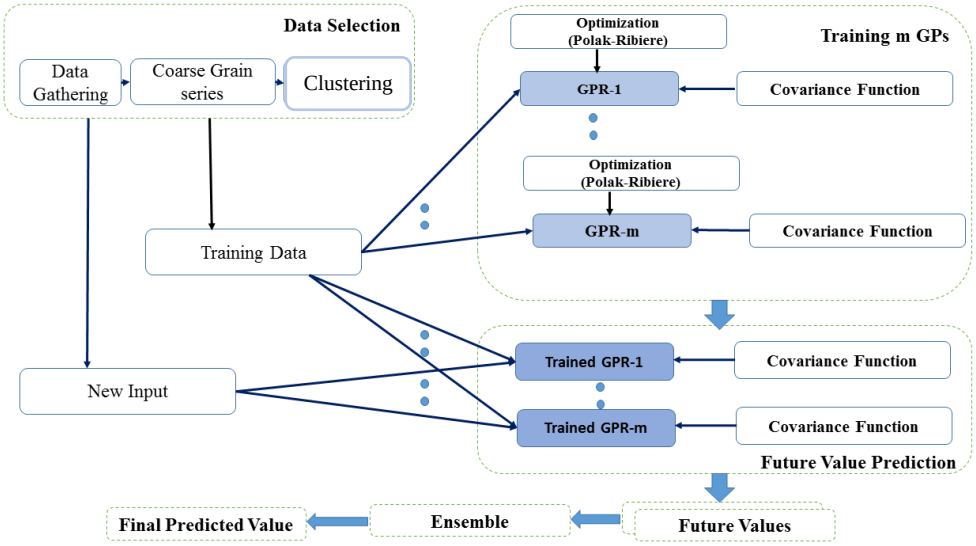

Figure 4.4: Framework for ensemble Gaussian Processes

This framework extends our two proposed models in Chapter 3. Similar to Chapter 3, in data selection phase, the direct trust values or the recommended trust values from the intermediaries are collected and arranged into a time series with each observation tagged by its time slot. Then using a fuzzy clustering technique, the series is clustered into clusters known as *regimes*, Each regime represents the similar behavior patterns occurred in the history of evolution of the trustworthiness values of the service provider. In the extension, instead of using only one expert for each regime, we used a group of experts trained for each regime. Each expert in the group is a GPR with a different kernel function. The reason behind using different GPR is that each of them, when consulted, will provide different generalization of the regime vectors in each cluster. The reason is that the clustering algorithm always does not group the regime vectors perfectly. In other words, two regime vectors in a cluster may still have different volatility patterns as shown in Figure 4.5. Due to this, a single kernel will provide only a generalized profile of these different volatility signatures. Therefore fusing the opinions from more than one expert will provide a better accuracy in the prediction. The mechanism of fusion is explained in section 4.3.2.1.

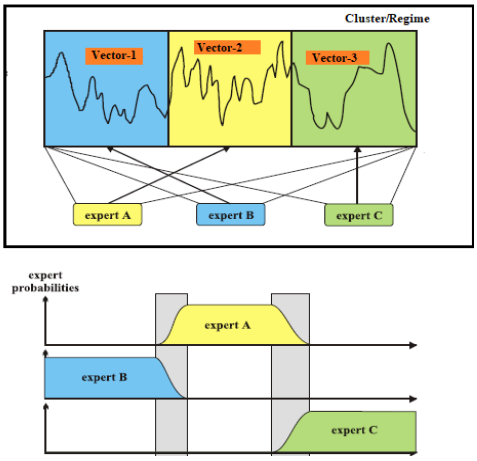

Figure-4.5: Illustration that a regime/cluster may have different regime vectors each sensitive to a particular dynamic. Each expert will be more specialize on a particular dynamic.

Based on this view, extensions to prediction model-1 and model-2 are shown in Figure 4.6 and Figure 4.7. In Figure 4.6, after the regime construction phase, using the transitions of regimes over time, a Markov transition matrix is constructed and R groups of P experts in each group are also trained to find the optimal values of their kernel parameters. In the final prediction step, all the activities of the prediction phase of our model-1 in chapter-3 are performed. Through these activities, when a query point is submitted, the Markov matrix will select one of the regimes and its associated group of experts will then be consulted to provide an ensemble prediction.

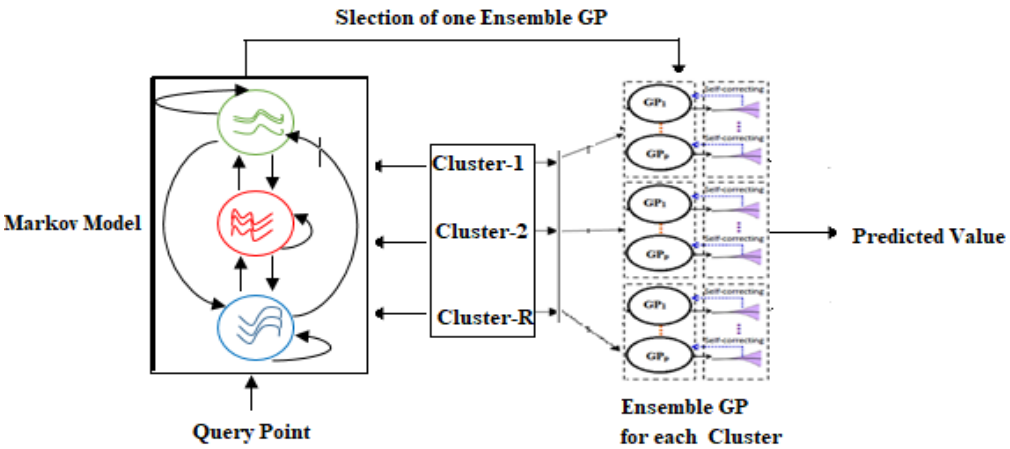

Figure 4.6: Overview of Markov Chain Local Ensemble Predictor Model

In figure 4.7, the extension to model-2, will train a Hidden Markov model using the coarse grain trust time series. The same Baum-Welch method of chapter 3 is used for training the HMM. The only difference is that all the R states are now subjected to a Gaussian emission probability function in Eq. (4.9)

$$b_{s_i}(y_t) = \frac{1}{\sqrt{2\pi\sigma_{s_i}^2}} \mathrm{e}^{\frac{\left|(y_t - f_{*s_i}(\mathbf{X_t}))\right|^2}{2\sigma_{s_i}^2}} \tag{4.9}$$

where $f_{*s_i}(\mathbf{X}_t)$ is the ensemble prediction from the i^{th} state s_i at time t.

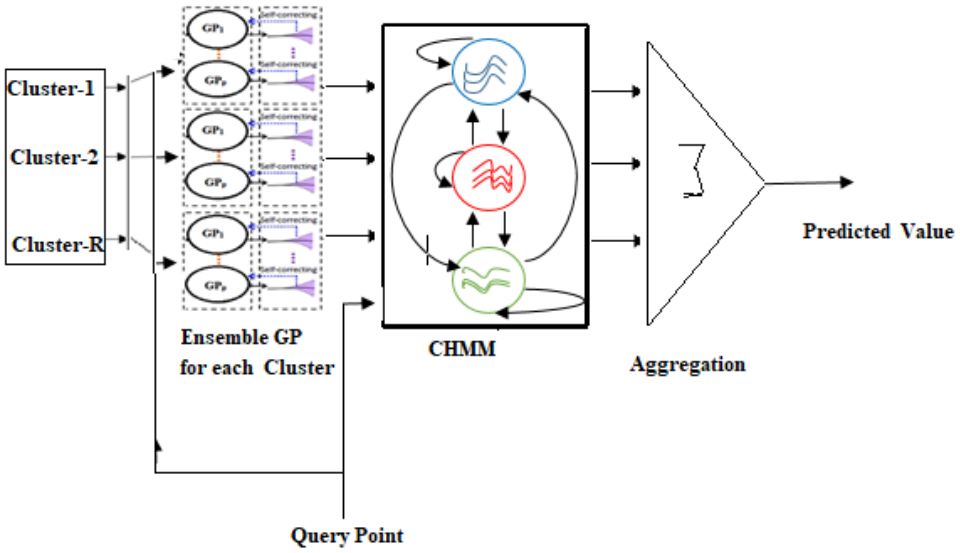

Figure 4.7: Overview of Coarse grained Hidden Markov Ensemble Predictor Model

During the prediction phase, the extended model will perform the prediction. All the steps of prediction phase of our model-2 in chatper-3 are used for the prediction. Only the last step is changed to reflect the ensemble prediction as follows.

Step 3: Prediction of future trust value.

The expert ensemble associated with each state identified by $P = [p_{s_1}, p_{s_2}, ..., p_{s_R}]$ will predict the future trust value. Finally, the future trust value is calculated as a weighted average according to Eq. (4.10).

$$T_{final} = \sum_{i=i}^{R} p_{s_i} . f_{s_i*} \tag{4.10}$$

Here f_{s_i*} is the ensemble prediction obtained from the i^{th} ensemble predictor model trained on the adjusted regime s_i

We now explain the ensemble prediction technique used in both of our extended models in the following section.

4.3.2.1 Ensemble Prediction

Given a set of data $D = \{x_i, y_i\}_{i=1}^{n}$, where $x_i \in R^d$ and $y_i = f(x_i) + \varepsilon, \in R$, we want to model the input-output relationship by using *P* Gaussian Processes $\{GP(m_j(x), \Sigma_j)\}_{j=1}^{P}$ Each Gaussian process has the mean function $m_j(x)$ and covariance function $\sum_j$.

About a new input $x_* \in R^d$, the j^{th}-GP model will predict the mean and variance of the function f using

$$\bar{f}_{j*} = k_{j^*}^T (K_j + \sigma_n^2 I)^{-1} y = k_{j_*}^T \alpha \tag{4.11}$$

$$\text{cov}(f_{j*}) = k_j(x_*, x_*) - k_{j^*}{}^T \left[K_j + \sigma_n^2 I \right]^{-1} k_j(x_*) \tag{4.12}$$

where $\alpha = (K_j + \sigma_n^2 I)^{-1}$ is the predictor vector and $K_j + \sigma_n^2 I$ is the gram matrix.

We proposed an ensemble prediction method using multiple kernels. Our ensemble is a vector of weights for individual kernels. The weight w_j is the

contribution from the j^{th}-Gaussian process model m_j. We define an ensemble vector λ as below:

$$\lambda = [w_1, w_2, ...w_P] \tag{4.13}$$

In each prediction session, the ensemble prediction model gives the final predictive value of the function f at the new point x_* as

$$f_* = \frac{1}{C_\lambda} \sum_{j=1}^{P} w_j f_{j*} \tag{4.14}$$

$$C_\lambda = \sum_{j=1}^{P} w_j \tag{4.15}$$

where C_λ is the normalizing constant summing the weight of every model in λ Since we do not have any prior information, all models are given uniform prior i.e. we set λ to ones. This means that all models will participate equally at the beginning.

We now formulate how the ensemble vector λ get updated when continuous prediction at different points are requested. When the j^{th}-model does the prediction about a new point $x_* \in R^d$ the predicted mean and variance are given by Eq. (4.11)-(4.12). With these values, the predicted value of the ensemble will be calculated using Eq. (4.14)-(4.15). Then afterward using ensemble prediction, our service user will calculate the final trust value using Eq. (4.1) and takes a decision to or not to interact with the provider.

4.3.2.2 Self Correction of Ensemble Weight

Based on the calculated final trust value, our service consumer interacts with the Web service provider and the actual value of actual trust will be available, let us call it y_t. On the basis of y_t, we can update our ensemble weights in the following manner.

Firstly, for each j^{th} model in the ensemble, the likelihood is calculated according to Eq. (4.16).

$$l_j(t) = l(y_t, \overline{f}_{j*}, \mathrm{cov}(f_{j*})) = \frac{1}{\sqrt{2\pi\,\mathrm{cov}(f_{j*})^2}} \exp(-\frac{(y_t - f_{j*})^2}{2\,\mathrm{cov}(f_{j*})^2}) \tag{4.16}$$

Larger the value of $l_j(t)$ the better is the j^{th} model. So Eq. (4.16) provides the basis for the model selection.

Secondly, the weight w_j in the ensemble vector at time *t* is adjusted as follows using Eq. (4.17).

$$\hat{w}_j(t) = w_j(t-1) + \frac{l_j(t)}{\sum_{i=1}^{P} l_i(t)} \tag{4.17}$$

Eq. (4.17) is further normalized to get the final weight w_j of the j^{th} model in the ensemble vector λ.

$$w_j(t) = \frac{\hat{w}_j(t)}{\sum_{i=1}^{P} \hat{w}_i(t)} \tag{4.18}$$

Combining Eq. (4.17) and Eq. (4.18), $w_j(t)$ is an effectively exponential smoothing average of the posterior probability of the j^{th} model over time.

4.3.2.3 Model Selection

In order to reduce the prediction cost of the GP model, "Sleep and Recovery" mechanism of [118] is used. At time slot *t* if the weight calculated using Eq. (4.18) for the model is less than a threshold, we can temporarily make the model m_j to sleep. So model m_j will not participate in the next time slot in the prediction of the trust value. After several slots, the model m_j will be recovered. The "Sleep and Recovery" mechanism is briefly explained here.

A vector of sleep counters $\xi = [\varsigma_1, \varsigma_2, ..., \varsigma_P]$ is maintained. Each counter value ς_j specifies how many time slots the model m_j would sleep. If the weight w_j is less than a threshold, $\eta = \frac{1}{P}$, we make the model m_j sleep. It will recover again when the number of prediction steps exceeds its counter ς_j . During the recovery step more than one model may be recovered. If κ is the number of models recovered, their respective weights in the ensemble vector λ are set to $\frac{\eta}{1 - \kappa * \eta}$ After normalization, the weights of the recovered models are equal to η .

To make the “weaker" predictor sleep longer, the sleep counter ς_j is self-adaptive during the continuous prediction. First ς_j is initialized as 1, which means the predictor would only sleep for one step. If after recovery the model m_j goes to sleep immediately in next step, we will double the value of ς_j. Otherwise, if the predictor successfully avoids the sleep trap, we would continuously halve the value of ς_j at very prediction step until ς_j = 1.

We now formally describe the continuous ensemble prediction with model selection procedure for a regime in the Algorithm 4.1

Algorithm 4.1: Ensemble Prediction with Model Selection

Inputs:

1. P Gaussian processes, $\{hyp_i, \text{cov}_i\}_{i=1}^{P}$
2. Initial training set, $< X, \text{y} >$

Outputs:

1. The ensemble prediction value f_*

Steps:

1. **[Initializations]**
   FOR $i = 1, 2, ..., P$ DO
   BEGIN
   $\omega_i \leftarrow 1;$ [Ensemble weight]
   $\varsigma_i \leftarrow 1;$ [Sleep counter]

$pStep \leftarrow 1;$ [Prediction session counter]

$pCount_i \leftarrow -1;$ [Subsequent prediction count]

$aIndex_i \leftarrow 1;$ [0-sleep; 1-awake]

$lrt_i \leftarrow -1;$ [Wake up session no]

$sTrap_i \leftarrow 0;$ [Sleep Trap indicator]

$f_{*i} \leftarrow 0;$ [Predicted value of model]

ENDFOR

3. **[Perform continuous prediction]**

DO WHILE NOT DONE

BEGIN

$K \leftarrow 0;\ C_\lambda \leftarrow 0;\ \eta = \frac{1}{2 * P}\ ;\ f_* \leftarrow 0;\ sum_\omega \leftarrow 0;$

$\mathrm{cov}(f_{*i}) \leftarrow 0;$ [Variance of prediction]

$\langle X, y, x_* \rangle - getData(D);$ [Get data for $pStep - th$ round]

FOR $i = 1, 2, ..., P$ DO [Who will wake up ?]

BEGIN

IF $aIndex_i == 0 \,\&\&\, pCount_i \geq \varsigma_i$ THEN [Waiting over]

$K \leftarrow K + 1;$

ENDIF

ENDFOR

IF $K \neq 0$ THEN [Set the weight of awakened models]

BEGIN

FOR $i = 1, 2, ..., P$ DO

BEGIN

IF $aIndex_i == 0 \,\&\&\, pCount_i \geq \varsigma_i$ THEN

BEGIN

$\omega_j \leftarrow \frac{\eta}{1 - K * \eta};$

$sTrap_i \leftarrow 0;$

$pCount_i \leftarrow 0;$

$aIndex_i \leftarrow 1;$

ENDIF

ENDFOR

ENDIF

FOR $i = 1, 2, ..., P$ DO [Prediction by awakened models]

BEGIN

IF $aIndex_i == 1$ THEN [Awaked, so predict]

BEGIN

$[f_{*i}, \mathrm{cov}(f_{*i})] \leftarrow \mathrm{gp}(\hat{hyp}_i, \hat{\mathrm{cov}}_i, X, y, x_*);$ [Predict for x_*]

$f_* \leftarrow f_* + f_{*_i} * \omega_i$; [Weighting individual prediction]

$C_\lambda \leftarrow C_\lambda + \omega_i$;

ENDTHEN

ELSE [Sleeping, reduce sleep count]

BEGIN

$pCount_i \leftarrow pCount_i + 1$;

IF $sTrap_i \neq 1$ THEN $\varsigma_i \leftarrow \varsigma_i / 2$;

ENDELSE

ENDFOR

$f_* \leftarrow f_* / C_\lambda$; [Ensemble prediction]

$otput(f_*)$; [Output predicted value]

$y_i \leftarrow Interact()$; [Obtain real trust]

FOR $i = 1, 2, ..., P$ DO [Calculate likelihoods]

BEGIN

IF $aIndex_i == 1$ THEN [Only Awakened models]

BEGIN

$$l_i \leftarrow \frac{1}{\sqrt{2\pi * \mathrm{cov}(f_{*i})^2}} * \exp\left(-\frac{(y_i - f_{*i})^2}{2 * \mathrm{cov}(f_{*i})^2} \right);$$

$l_{sum} \leftarrow l_{sum} + l_i$;

ENDIF

ENDFOR

FOR $i = 1, 2, ..., P$ DO [Update weight and sum up]

BEGIN

IF $aIndex_i == 1$ THEN

BEGIN

$$\omega_i \leftarrow \omega_i + \frac{l_i}{l_{sum}};$$

$sum_\omega = sum_\omega + \omega_i$;

ENDIF

ENDFOR

FOR $i = 1, 2, ..., P$ [Normalize weight]

IF $aIndex_i == 1$ THEN $\omega_i \leftarrow \frac{\omega_i}{sum_\omega}$;

ENDFOR

FOR $i = 1, 2, ..., P$ [Making model selection for next round]

BEGIN

IF $\omega_i \prec \eta$ && $aIndex_i == 1$ THEN

BEGIN

$pCount_i \leftarrow 0$;

$aIndex_i \leftarrow 0;$
IF $lrt_i == (pStep - 1)$ THEN
BEGIN
$\varsigma_i \leftarrow 2 * \varsigma_i;$
$sTrap_i \leftarrow 1;$
ENDIF
ENDTHEN
ELSE
$\varsigma_i \leftarrow 1;$
ENDIF
$lrt_i = pStep + \varsigma_i;$
ENDIF
ENDFOR
$pStep \leftarrow pStep + 1;$ [Go for next prediction]
ENDWHILE

The algorithm first receives the p GP models. Each model is defined by its initial hyperparameters hyp and covariance kernel cov (here mean is assumed to be zero for all model). In each round of prediction, the training data and query point are obtained by the function $getData()$. All ensemble members are trained on these inputs to obtain their optimised hyperparameters using the function $trainModel(\{hyp_i, \mathrm{cov}_i\}_{i=1}^{P}, \mathrm{X}, \mathrm{y})$. At the beginning of a prediction round a check is made to find out those members who will be awakened or recovered from last sleep. If there is at least one member to be recovered, then its weight in the ensemble vector is first set and then proceeds for prediction along with all other members who are not put to sleep in the previous round. Prediction by each member is done by calling the function $\mathrm{gp}(\hat{hyp}_i, \hat{\mathrm{cov}}_i, X, y, x_*)$. Weighted sum of the all predictions and finally the ensemble prediction is formed in f_*. Using this predicted value our service consumer will interact with the service provider to obtain the real QoS value y. For the next round of interaction, the ensemble weights are updated based on the likelihood value l. At the same time which member model will be put to sleep at the end of just concluded round is done by checking the validity of the condition $\omega_i \prec \eta$. If a model is to sleep,

first we check its last recovery time lrt. If $lrt == (pStep - 1)$ then the model is going to sleep immediately after the last recovery, therefore it will fall into a sleep trap. Consequently its sleep trap flag, $sTrap$ will be set and sleep counter, ς will be doubled.

4.3.2.4 Incremental update of the Ensemble Members

In our set up, the predictions are performed sequentially as shown in the Figure 4.3. By the time the prediction for time t is over, the service user has to predict the trust value for time $t+1$, so we need to have the query regime vector $\mathbf{X}_{N+1}$ To obtain this we have to gather the observation y_{t+1}. Either the direct trust value generated by using Eq. (3.26) or the recommended trust value generated using Eq. (3.33) can be used to form y_{t+1}. Once we obtain y_{t+1}, the query regime vector for time $N+1$ as Eq. (4.19)

$$X_{t+1} = [y_t y_{t-1}, ..., y_{t-(m-1)}] \tag{4.19}$$

The update of the observation vector $\mathbf{y} = [y_1, y_2, ..., y_t, y_{t+1}]$ and its associated course time series $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, \mathbf{x}_{t+1}]$ can be done sequentially in this manner. However, if we do not update Markov model and ensemble, the proposed models will fail to produce desirable results. On the other hand, retraining of $R \times P$ members will be highly costly. Therefore, we adopted the online update method of [160] for the covariance matrix of all GPR experts.

All $R \times P$ members are not updated at the end of each prediction session. We followed the following step for the update.

Step 1: Insertion of new pair $[\mathbf{x}_{t+1}, y_{t+1}]$.

We first insert the pair $[\mathbf{x}_{t+1}, y_{t+1}]$ to one of the R regimes (clusters). The target cluster is identified from the future state vector V_f created in Activity-2 of the

prediction phase of model-1. As this vector has only one non-zero entry, the column index of this non-zero entry is the regime where the pair must be inserted.

For the case of extended model-2, we identified the regime by using the posterior probability vector $[p_{s_1}, p_{s_2}, ..., p_{s_R}]$. We selected the target regime as the regime that has the maximum probability value in $[p_{s_1}, p_{s_2}, ..., p_{s_R}]$.

Step 2: Update of ensemble members

Once the target regime is selected, the update of the covariance matrices of the P GPR experts is done by using the method explained in [160].

The predictor vector $\alpha = (K_j + \sigma_n^2 I)^{-1}$ of Eq. (4.12) can be incrementally updated by directly adjusting Cholesky decomposition of the Gram matric $(K_j + \sigma_n^2 I)^{-1}$. For this purpose the predictor vector can be rewritten as $\mathbf{y} = \mathbf{L}\mathbf{L}^{\mathbf{T}}\alpha$, where the lower triangular matrix $\mathbf{L}$ is a Cholesky decomposition of the Gram matrix. Incremental insertion of a new point is achieved by adding a an additional row to the matrix $\mathbf{L}$. $\mathbf{L}_{new}$ and $\mathbf{K}_{\mathbf{new}}$ are obtained by adding an additional row and column, such that

$$\mathbf{L}_{new} = \begin{bmatrix} \mathbf{L}_{\mathbf{new}} & \mathbf{0} \\ \mathbf{l}^{\mathbf{T}} & l_* \end{bmatrix}, \quad K_{new} = \begin{bmatrix} \mathbf{K}_{\mathbf{new}} & \mathbf{K}_{\mathbf{new}}^{\mathbf{T}} \\ \mathbf{k}_{\mathbf{new}} & k_* \end{bmatrix}$$

where $\mathbf{k}_{\mathbf{new}} = k(\mathbf{X}, \mathbf{x}_{\mathbf{new}})$ and $k_{new} = k(\mathbf{x}_{\mathbf{new}}, \mathbf{x}_{\mathbf{new}})$ then $\mathbf{l}$ and l_* can be computed by solving

$$\mathbf{L}\mathbf{l} = \mathbf{k}_{\mathbf{new}} \; and \quad l_* = \sqrt{k_{new} - \|\mathbf{l}\|^2}$$

Now the update steps are as follows:

- Compute $\mathbf{l}$ after computing the kernel vector $\mathbf{k}_{\mathbf{new}}$ and by substituting it back in $\mathbf{L}\mathbf{l} = \mathbf{k}_{\mathbf{new}}$

- Compute l_* from $l_* = \sqrt{k_{new} - \|\mathbf{l}\|^2}$
- Compute α_{new} by twice back-substitution while solving

  $\mathbf{y_{new}} = \mathbf{L_{new}} \mathbf{L_{new}^T} \alpha_{\mathbf{new}}$

After the update of the covariance matrices the Markov model can be either retrained or continue for prediction with the previously trained hyperparameters.

4.4 Experiments and Results

To evaluate the effectiveness of our proposed models, we compare them with one of the popular state of the art reputation trust models [80]. The simulated experiment was conducted in MATLAB. The MATLAB functions were used to simulate the behavior of Web service providers and service consumers. First we explain the simulation environment in the following.

Experimental Environment Set Up

Our simulation environment is a community of Web service providers, Service consumers and Service raters (Figure 4.8). Service raters are also service consumers, but they are the sources of recommendation in our experiments.

We have created an environment of 200 raters, five service providers of different behaviours and two service consumers. We ran three rounds of simulation (for each value of k) consisting of 1000 transactions. In each round, a particular type of Web service provider from the pool is selected and all the models performed the assessment of its behavior from 1000 transactions. Final comparison among the performance of models is performed taking the average

of these three rounds. In one transaction of each round the following events are designed to occur:

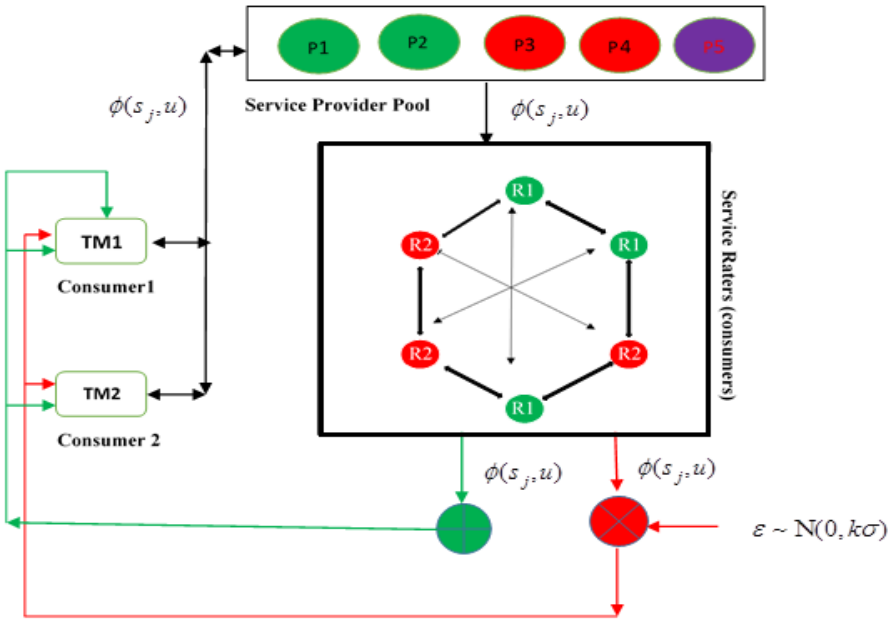

Figure 4.8: Simulation Environment where $\phi(s_j,u)$ is the experienced QoS of a service provider s_j by a user in a transaction u. Dishonest users add a noise term $\varepsilon \sim N(0,k\sigma)$ to their experienced QoS to form recommendation. k is a scaling factor in the range (1-3) and $\sigma = 0.01$ is the standard deviation.

- First, a random number of service raters interact with one of the four service providers. The experienced QoS value from this selected service provider is then recorded.
- Second, these service raters exchange the experienced QoS among themselves and to update two counters $\langle n\#, p\# \rangle$ to asses each other's reputation feedback trust by using Eq.3.27. If received QoS value differs from the actual value by a threshold, $n\#$ increased 1; otherwise $p\#$ is increased by 1. We used a threshold value of 0.02 in our experiment.
- Third, the two service consumers collect the rating values tagged with their time of interaction from these service raters. Rating values were

then used to evaluate the trust value of the service provider using their own trust models (TR1 and TR2). In case of consumer-1, for each rater, from which the rating is collected, a check is made to see whether the rater is in its reputation range using a value of the threshold R_{th}. If the rater is not in the reputation range, then the values of the two counters $\langle n\#, p\# \rangle$ corresponding to this rater are also collected from other raters to assess the reputation trust of the rater. From the collected information, Eq. (3.28) to Eq.(3.30) is used for the evaluation of reputation trust of the rater.

- Finally, two service consumers interact with the service provider selected in this round to get their own assessment of its quality. This self-assessed quality value is recorded.

In order to cover all possible scenarios, we considered the following types of Web service providers in our provider pool.

Consistently High performer (P1): This category of provider does not engage in any malicious activity. They deliver consistently high QoS. We modeled this type of service provider by using MATLAB function to produce a QoS values in the range (0.8-1.0). For generation of QoS values we used MATLAB's *random number* generator function.

Consistently Low performer (P2): This category of provider is same as the above category, however they consistently deliver low QoS values in the range (0.0-0.2). These type of providers are the ones who always take advantage of the consumer.

Malicious High to Low performer (P3): The providers in this category, perform honestly at the beginning to derive a good reputation trust and then start 'milking' the derived reputation value. We model them by another MATLAB function which generated the QoS values in the range (0.8-1.0) initially in the first half of the interactions and then generated low QoS values in the range (0.0-0.2) in the latter half of the interactions.

Malicious Low to High performer (P4): The forth category behaves in an opposite manner to the malicious High to Low performer. They are the ones who learn from their previous mistakes.

Oscillatory Performer (P5): The last category of providers who sometimes perform satisfactorily and unsatisfactorily at other times. Our MATLAB function generates the QoS values in the range (0.0-1.0) for this type of provider.

Our service consumers, who were designed to act as raters behaved in one of the following models.

Honest Rater(R1): Rater of this type always provides the values of QoS as their experienced QoS values without any modification.

Dishonest Rater(R2): They always do not provide the QoS values experienced without modification. They generate the rating QoS values by adding a noise to the QoS value experienced, $\phi(s_j, u)$. In our simulation, we added to the experienced QoS value, $\phi(s_j, u)$ a white noise term, $\varepsilon \sim \mathrm{N}(0, k\sigma)$, that follows a zero-mean Gaussian distribution. The variance of the distribution is set to be $k\sigma$, where k a scaling factor in the range (1-3) and σ is the standard deviation. To accommodate subjective inaccuracy in the recommended QoS values, we set $\sigma = 0.01$, which means a relatively big deviation noise. Hence, for different values of k, each feedback QoS value will have a different deviation that follows a zero mean normal distribution with variance in the range (0.01- 0.03).

Consumer-1 is equipped with our trust model. Consumer-2 is loaded with RATEWeb [80] trust model. These two consumers collect the ratings from the group of service raters. Then they combine these feedbacks along with their personal experience to derive the final trust value for each interaction with any selected service provider.

Consumer-2 uses a trust model based on RATEWeb. RATEWeb trust model is multi-attribute trust model. On the other hand our proposed models are single-value trust models. Therefore, for equitable comparisons with our models, we considere only one QoS attribute of the service provider while implementing

RATEWeb. In other words, our consumer-2 and all service raters will evaluate the reputation of Web service provider based on single QoS attribute. The original metric for reputation of a service s_j given in Eq.(4.20).

$$\text{Reputation}(s_j) = \frac{\sum_{x=1}^{L}\left[\frac{\sum_{h=1}^{m}\left(\phi_h(s_j,u)^x * RSV_h\right)}{\sum_{h=1}^{m} RSV_h} * f_d * C_r(x)\right]}{\sum_{x=1}^{L} C_r(x)} \tag{4.20}$$

Here $\phi_h(s_j,u)^x$ is the rating assigned to the QoS attribute h by the service rater x for the service provider s_j in transaction u, RSV_h is the preference of the service consumer for the attribute, m is the number of QoS attributes, and L is the set of service consumers.

For a single QoS attribute Eq. (4.20), has been reduced to Eq. (4.21).

$$\text{Reputation}(s_j) = \frac{\sum_{x=1}^{L}\left[\phi(s_j,u)^x * f_d * C_r(x)\right]}{\sum_{x=1}^{L} C_r(x)} \tag{4.21}$$

Here $\phi(s_j,u)$ is the rating assigned to the single QoS attribute by the service rater x for the service provider s_j in transaction u. It may be noted that when $m=1$, RSV_h cancels out.

The trust model of the consumer-2 is given in Figure 4.9. A loop is iterated over a list of service rates that have personal assessment of the service provider s_j in question and reputation feedbacks are collected. The collected reputation feedbacks *Vi's* are used to compute majority rating *M*, Then the credibility C_r of each rater is computed by using the majority rating *M*¸ last assessed reputation *A* and rating usefulness factor Uf_r .

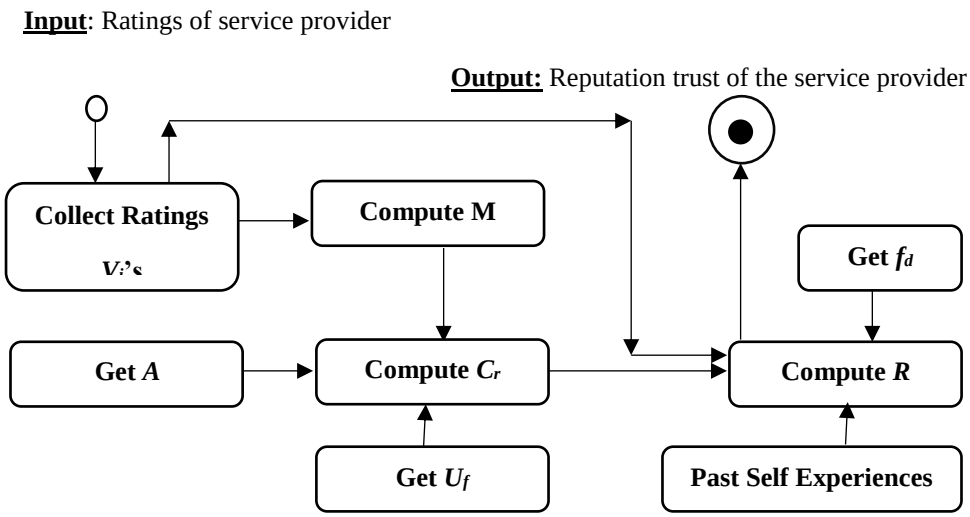

Figure 4.9: Trust model for consumer-2 based on RATEWeb[80]

Finally, the reputation of the service provider s_j is computed using Eq.(4.21). The reputation value is diluted by a time-dependent decay factor f_d.

Consumer-1 used all the trust model proposed in Chapter-3 and their extended versions explained earlier in this Chapter. The short names of the models that we used in the description are summarized in the following Table-4.2.

Table -4.2: Short names for the four proposed models

Short name	Model Details
M1	Markov chain based Local GPR (MLGPR)
M2	Markov Chain Local Ensemble Predictor Model
M3	Coarse Grained Hidden Markov GPR (CHMMGPR)
M4	Course grained HMM ensemble predictor model

To implement the ensemble models M2 and M4, we use seven GPR models. The corresponding covariance functions used to contrast the seven members are given Table 4.3.

Table 4.3: Models with their kernels

Ensemble Member	Corresponding Kernel used
P1	SE
P2	RQ
P3	Lin
P4	Lin x SE
P5	C+Lin+WN
P6	Lin+SE+WN
P7	SExPer

Table-4.4: Parameters Settings for Simulation

Parameter Name	Value
Time Horizon, T_h	100 time instances
Time spot, T	1 time instance
Weighting factor, λ of Eq.(3.25)	0.5
Reputation range threshold, R_{th}	0.5
Advisor usability threshold, R_{ath}	0.9
Regime vector selection threshold, xC_{th}	0.9
Model order, m	1
Number of regimes (clusters), R	3

The opened parameters of the models M1-M4 in the simulation were set according to the Table 4.4.

We need to fill up two buffers each of length equals to the Time horizon T_h with direct trust values from the personal direct experiences of our consumer-1 and reputation feedbacks from the raters respectively. This is the *booting period* to use the proposed prediction mechanisms. Evaluation of the final trust value of each service provider at different time instances by service consumer-1 was done according to following.

First transaction:

The final trust value of the Web service provider is assessed using Choice-1 of our direct trust algorithm 3.2 given in Chapter-3. This is because there is no transaction recorded in the direct trust table of our service consumer. So trust value is calculated from only the indirect trust component of Eq. (3.25) using Eq. (3.31) to Eq. (3.33)

From second transaction up to the T_h^{th} transaction:

This is the booting period for the prediction models. Final trust values during this period are calculated using Eq. (3.25). Its direct trust component was evaluated using Eq.3.26 and the indirect trust component was obtained from Eq. (3.31) to Eq. (3.33).

From $(T_h+1)^{th}$ transaction up to *1000th* transaction:

Trust evaluation during this period was done using Eq. (3.25). However the direct trust and indirect trust components were estimated using the prediction mechanism proposed in models M1-M4 given in Table 4.3.

When the buffers are full, every time a transaction happens, the actual experience of QoS value and recommended reputation value are saved and the oldest values from both the buffers are discarded. This is done to maintain only the records of the last T_h transactions. Since RATEWeb does not use the context information, in the simulation we assume that all interactions happened for only one context and hence only the Type 2 of information sources given in Table-3.3 of Chapter 3 are used.

In our implementation, the value of f_d and time decay factor for Eq. (3.26) and Eq.(3.31)-Eq.(3.33) are decided from an exponential decay model

$e^{-k\Delta t} = e^{-k(u_n - u_{n-1})}$ where k is the decay rate and Δt is time difference between two transactions. Figure 4.10 displays the effect of decay rate over transactions. We have divided the time horizon T_h into equal halves. All feedbacks in the first half are given a decay factor of $k = 0$ and last half is associated with a decay factor of $k = 0.01$.

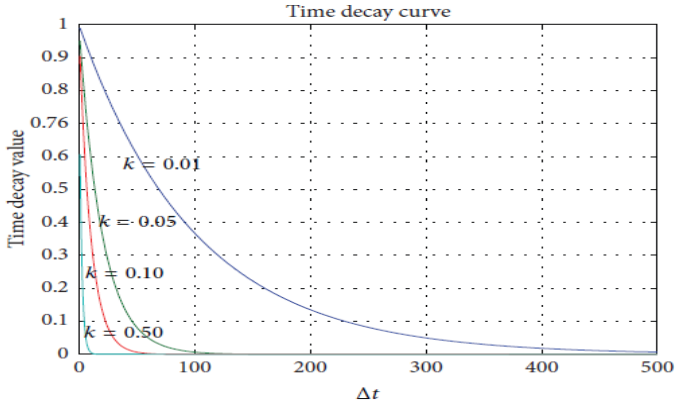

Figure 4.10: Exponential Time Decay Model

Experimental Results

We performed two sets of experiments. In the first experiment, we maintained the ratio of dishonest raters to honest raters at 75:25. In the second experiment, we reversed this ratio. In all experiments, we conducted three rounds of simulations with the noise variance 1σ, 2σ and 3σ. The average behaviours of all the models are presented below.

The comparison of the four proposed models with the RATEWeb [8] is shown in Table 4.5 and Table 4.6. The comparison is done using the average of Root Mean Square Errors (RMSE) and average Normalized Mean Square Error (NMSE). Given N trust predictions, the RMSE and NMSE are calculated from the Eq. (4.22) and Eq.(4.23).

$$RMSE = \sqrt{\frac{\sum_i \left(\hat{T}_i - T_i\right)^2}{N}} \tag{4.22}$$

$$NMSE = \frac{1}{N}\sum_{i}\frac{\left(\hat{T}_i - T_i\right)^2}{\bar{\hat{T}} \times \bar{T}} \tag{4.23}$$

Here, $\hat{T}$ and T are the predicted and the real trust values, $\bar{\hat{T}}$ and $\bar{T}$ are the means of the predicted trust and real trust values. Average of RMSE and NMSE are calculated by taking the average of Eq. (4.23)-(4.24) over N predictions.

Results presented in these two tables clearly show that our proposed models except M1 have smaller values of average Root Mean Square Error (RMSE) and average Normalized Mean Square Error (NMSE) as compared to RATEWeb model. Therefore it can be concluded that Models M2, M3, M4 outperformed the RATEWeb model in trust prediction. The best model is M4 followed by M2, M3, RATEWeb and M1.

Table-4.5: Comparison of prediction performance of all models when the honest raters outnumbered the dishonest raters.

Model	Provider Model									
	P1		P2		P3		P4		P5	
	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE
M1	0.0020	0.0039	0.0021	0.0156	0.00103	0.0089	0.00457	0.0047	0.0069	0.0287
M2	**0.0005**	**0.0023**	**0.00115**	**0.00288**	**0.00090**	**0.0071**	**0.00021**	**0.0020**	**0.0032**	**0.0057**
M3	0.00011	0.0025	0.00121	0.00335	0.00089	0.0068	0.00026	0.00198	0.0076	0.0348
M4	**0.0001**	**0.0018**	**0.0011**	**0.0022**	**0.00012**	**0.00017**	**0.00017**	**0.00088**	**0.0011**	**0.0045**
RATEWeb	0.0011	0.0022	0.0014	0.0037	0.00094	0.0067	0.00029	0.0020	0.0078	0.0357

Table-4.6: Comparison of prediction performance of all models when the dishonest raters outnumbered the honest raters.

Model	Provider Model									
	P1		P2		P3		P4		P5	
	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE	RMSE
M1	0.0043	0.0058	0.00301	0.6235	0.0036	0.0645	0.0235	0.0287	0.0898	0.0632
M2	**0.0013**	**0.0032**	**0.0011**	**0.2845**	0.002	0.0068	**0.0016**	0.0132	**0.0085**	**0.0207**
M3	0.00128	0.0034	0.0013	0.3105	0.003	0.0065	0.0019	0.0129	0.0802	0.0396
M4	**0.0011**	**0.0030**	**0.0015**	**0.1212**	**0.0010**	**0.0032**	**0.0016**	**0.0124**	**0.0040**	**0.0134**
RATEWeb	0.0013	0.0034	0.0011	0.0550	0.0010	0.0070	0.0018	0.0133	0.0432	0.0835

Further, to visualize graphically the performance of prediction, we selected M4 and RATEWEb and showed their prediction performances in Figure-4.11-Figure-4.15. In all the figures, the two left most plots are made from 100 samples selected out of 1000 transactions. The rightmost plots show the cumulative distribution function of prediction errors from 1000 transactions.

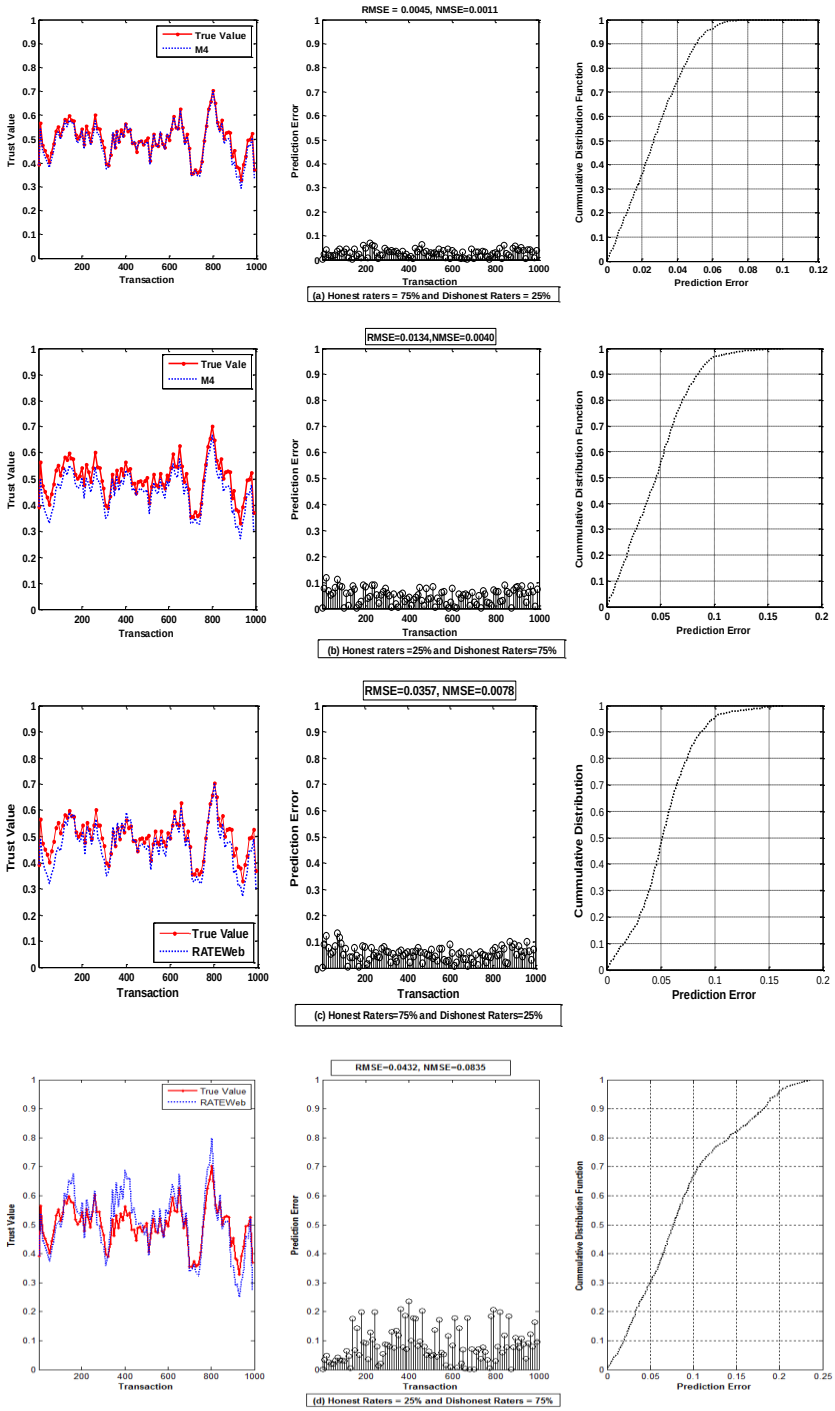

Figure 4.11: Performance of M4 and RATEWeb in prediction of behavior of an oscillatory Web service provider

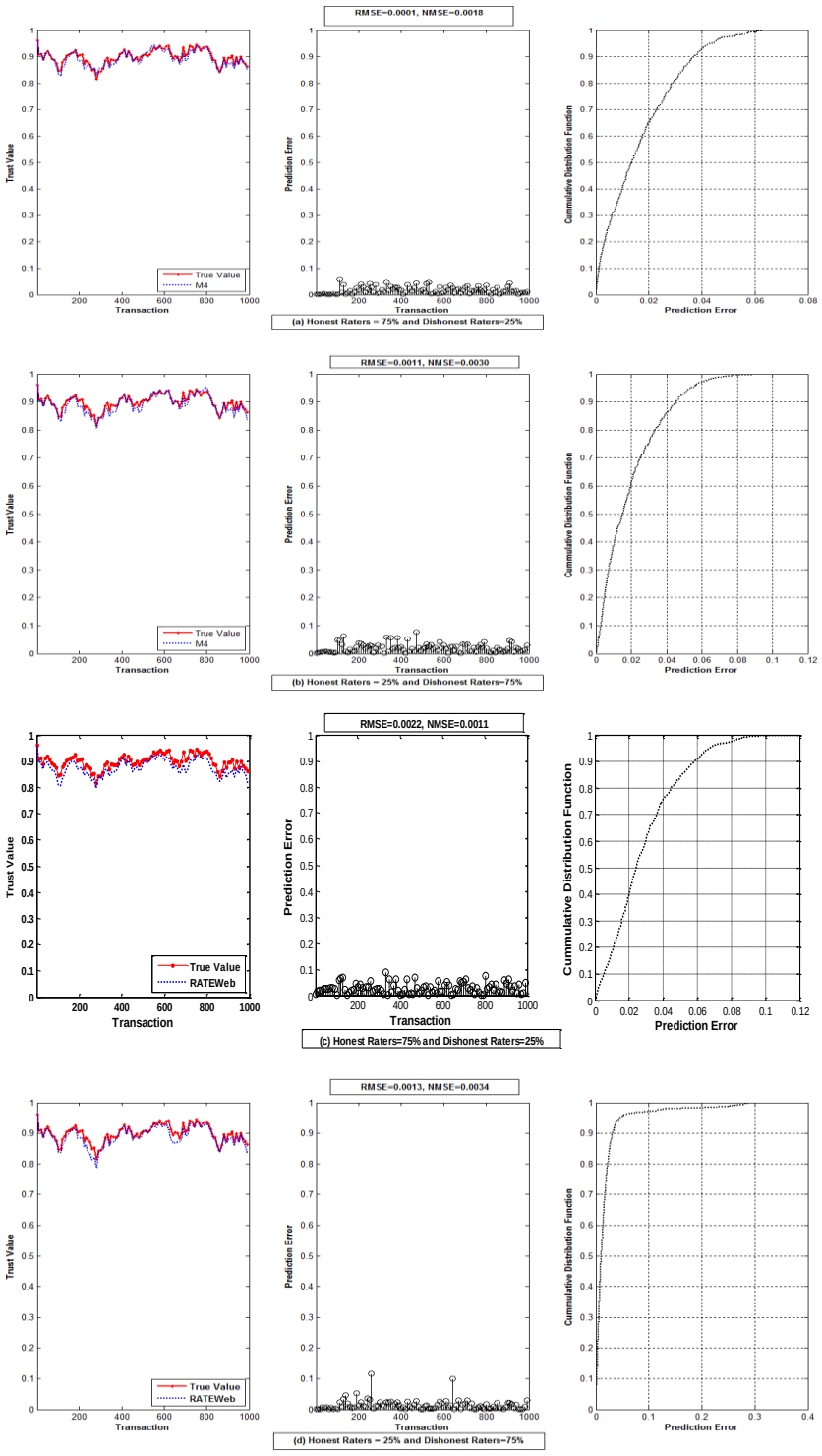

Figure 4.12: Performance of M4 and RATEWeb in prediction of behavior of consistently good Web service provider

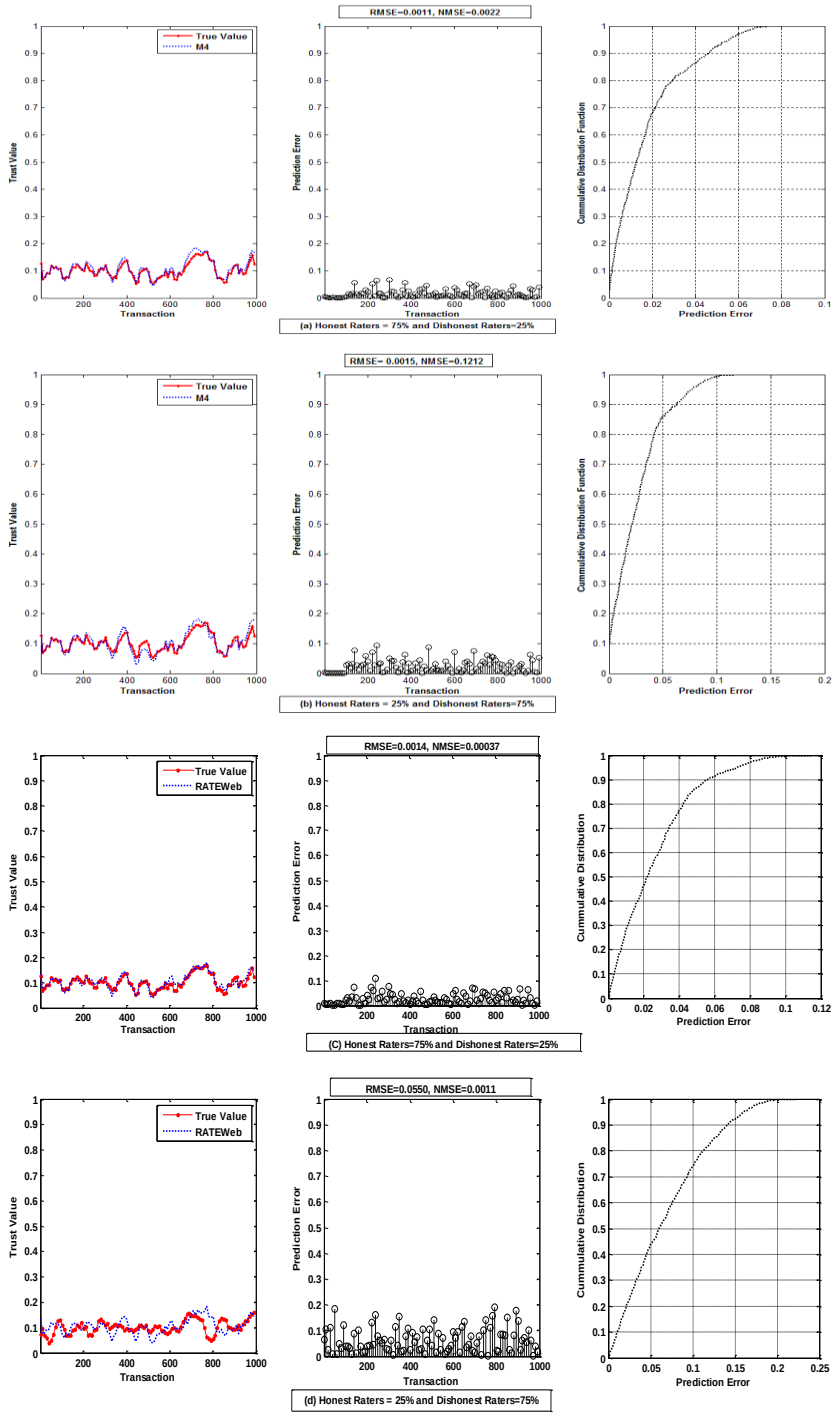

Figure 4.13: Performance of M4 and RATEWeb in prediction of behavior of consistently bad Web service provider

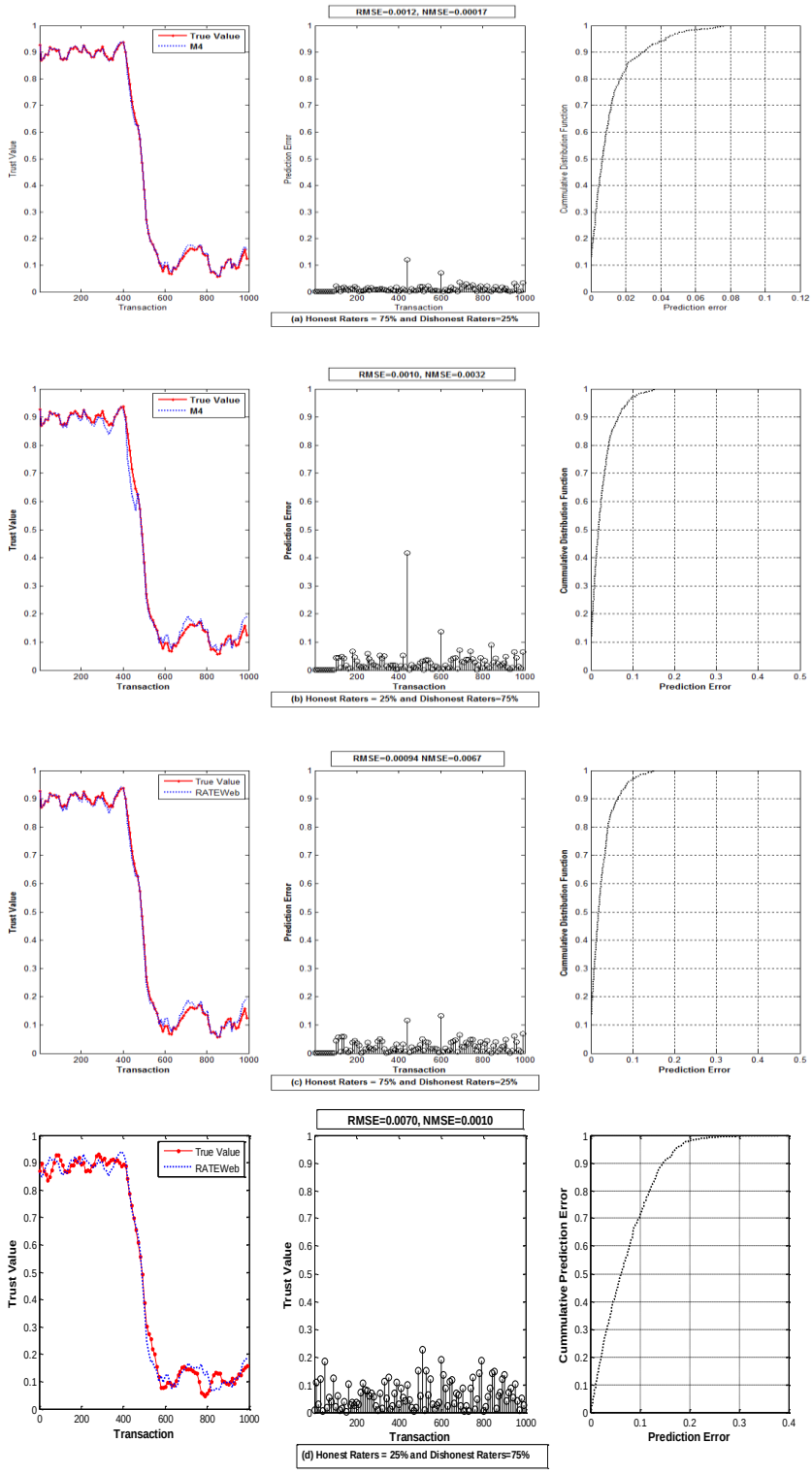

Figure 4.14: Performance of M4 and RATEWeb in prediction of behavior of Web service provider who swings from high to low performance

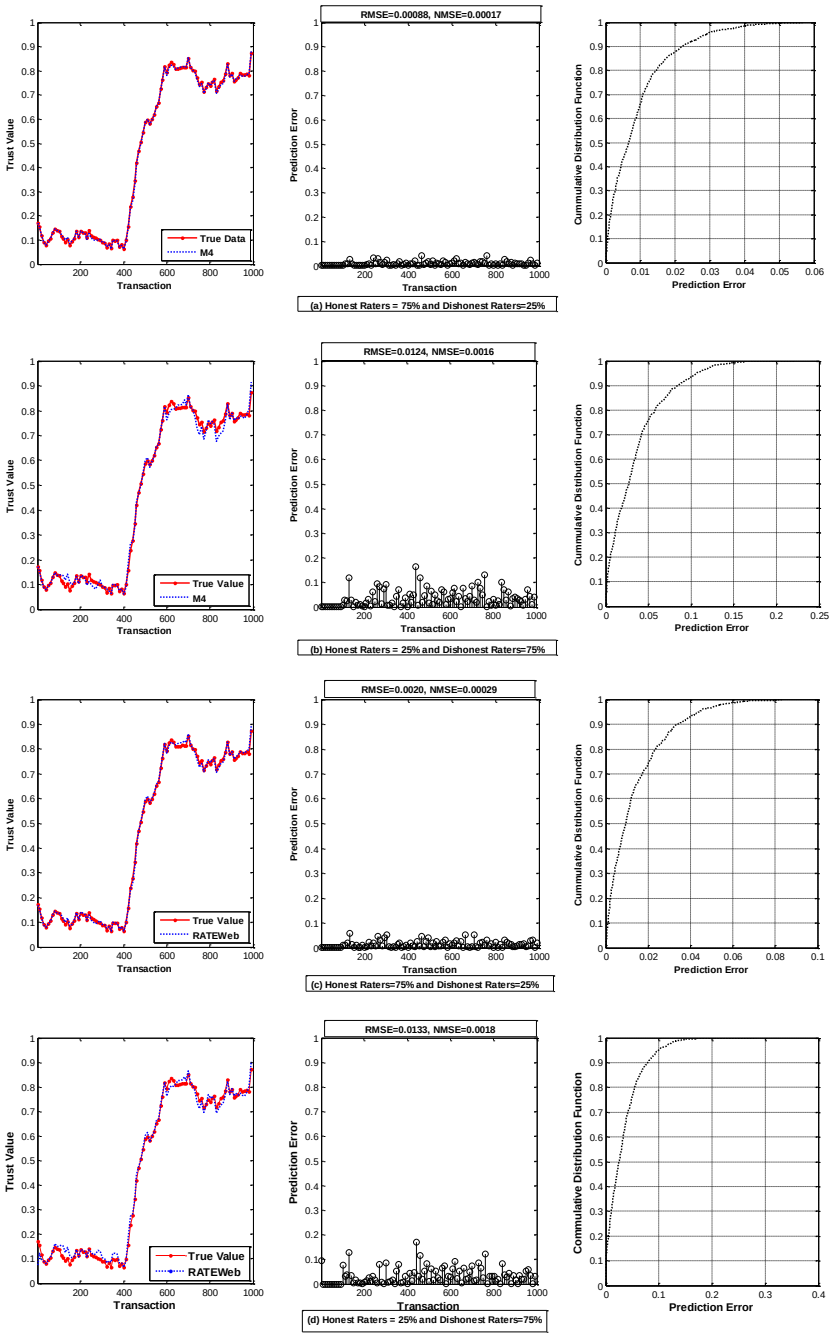

Figure 4.15: Performance of M4 and RATEWeb in prediction of behavior of Web service provider who swings from low to high performance

In all the figures, it can be observed that when the honest raters outnumbered the dishonest raters in the environment, the predicted trust values by all models are fairly consistent and close to the real trust values. However when the number

of dishonest raters is more than the honest raters, the predicted trust values deviate from the actual behavior of the Service provider. This is due to the reason that most of the reputation feedbacks are obtained from dishonest sources. However, in all the models, due to incorporation of the direct experience of the service consumer, the assessed trust values are still close to the actual values. The difference is around a margin of 0.2. Among the models, M4 showed better prediction accuracy compare to RATEWeb in their performance. This can be justified from the cumulative distribution plots of the prediction errors over 1000 transaction in all cases.

Experiment with real data set-1

We performed one experiment on the real life dataset obtained from Cloud Armor Project. The same dataset is used in chapter 3. In this experiment, we make a comparison of our proposed models with one HMM based trust prediction model for Web services [161]. They modeled reputation as a time series and used a Hidden Markov Model (HMM) to predict future reputation. Reputation is derived from a set of Quality of Web Service (QoWS) parameters such as performance, availability, reliability and response time. Thus, the reputation of a service in their model is represented by a single value or a vector representing a value for each QoWS attributes. A HMM with a mixture of Gaussian distributions is used as the emission probability function of every state.

Training Data Preparation

We have selected a service called “BlueHost” for the study. The details of the QoS values of the selected Web service are given in the Table 3.4. There are 455 feedback sessions recorded for this service. Only the values for QoS attributes *availability, price, accessibility, ease of use and technical service* are recorded. The value of each QoS attribute is an integer in the range 1-5 while

the final trust value is real vale in the range [1,5]. In 43 sessions the value of one or more of the QoS attributes are found missing. We removed these sessions from the data set.

Table 4.7: Details of data set selected from Cloud Armor dataset

Cloud Service Name	Data points		Time		QoS Attribute Value (Integer)										Trust(continuous)	
					Availability		Price		Technology Support		Accessibility		Ease of Use			
BlueHost	Before Preprocessing	After Preprocessing	From	To	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
	455	411	August 8, 2006	May 24, 2012	1	5	1	5	1	5	1	5	1	5	1	5

Experimental Results

The number of states R of the HMM model is decided by using the Elbow method. The plot of the total within cluster sum of squares against the number of clusters is shown in the Figure 4.16. We used $R = 4$ for the experiment. 50% of the data points were used for initial training of all models.

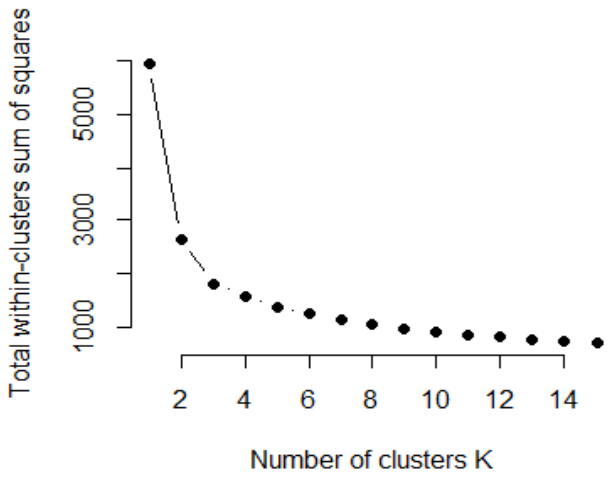

Figure 4.16: Selection of number of clusters by Elbow method.

The trained models are then used on the remaining 50 % of the reputation time series data to ascertain the accuracy of all models. The models are compared by using the percentage error given by the Eq. (4.24).

$$PerErr = \frac{predicted\ \ value - actual\ \ value}{actual\ \ value} \times 100 \tag{4.24}$$

Figure (4.17) shows the percentage errors in prediction using all the models. The results in all the figures show the comparison of actual trust data with predicted trust data. As can be seen, all the proposed methods are able to compete with the Gaussian Mixture HMM model. Except models M1 and M2, all other models performed better than Gaussian Mixture HMM model.

The experimental results clearly show that statistical techniques and machine learning techniques focus on providing a sound theory for trust management.

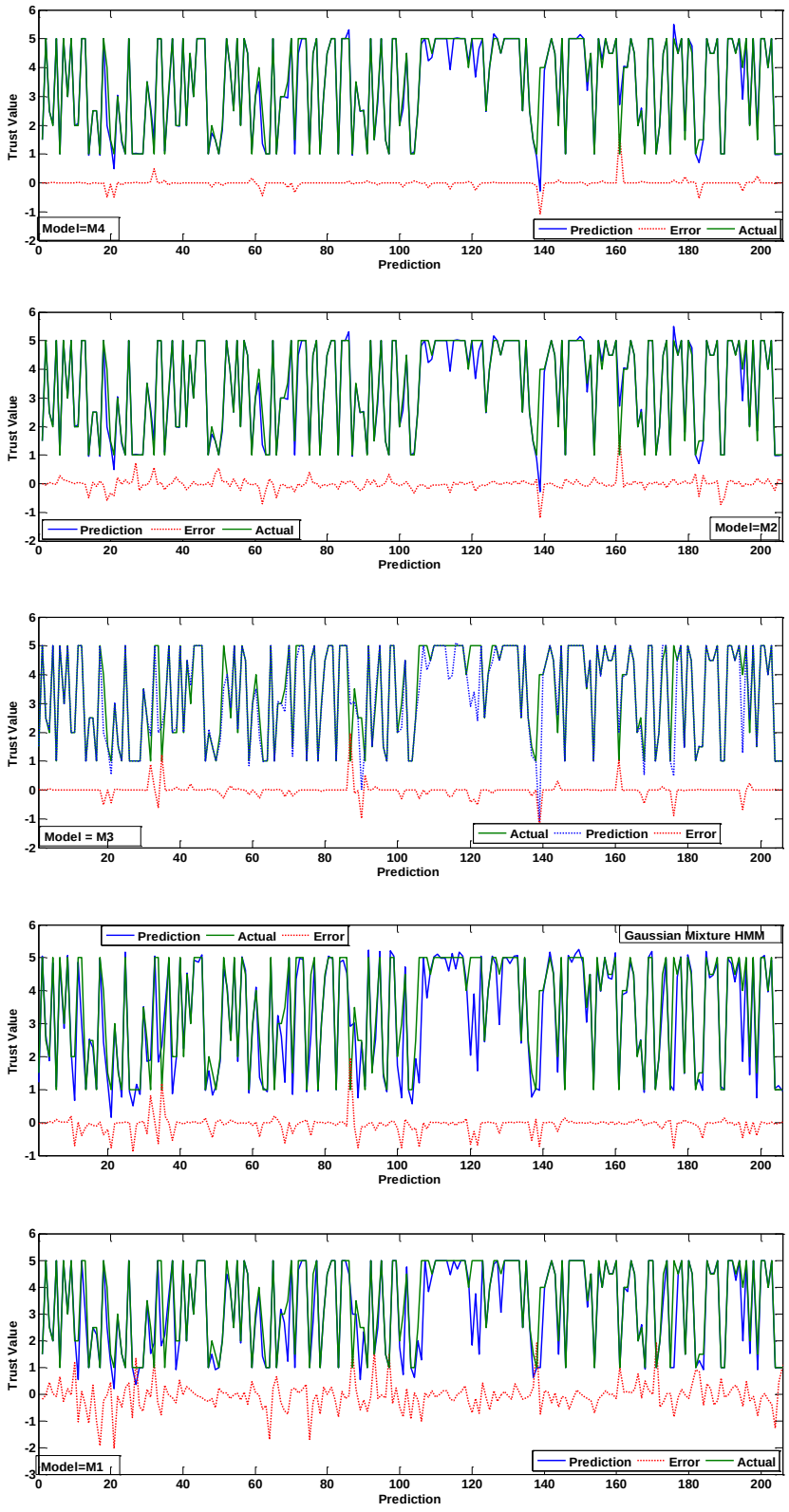

Figure 4.17 : Predicted Vs actual trust values and prediction performance in the form of percentage error for all models

Experiment with real dataset-2

Another set of experiments was conducted with the data set [161,162]. The dataset describes real-world QoS evaluation results from 339 users from 30 countries on 5,825 Web services in 73 countries. The data set consists of two 339 by 5,825 real-world matrices for response time and throughput, respectively. Each entry in the matrix represents the response-time or throughput value observed by a user on a web service. We used in this experiment the matrix of response-time to investigate the prediction quality and consistency of our proposed models. The mean and standard deviation of response time are 1.43 and 31.9 seconds, respectively. The large deviation indicates that response time has a wide range of values.

For the purpose of our investigation, we have selected randomly 10 Web services. Each of these Web services exhibits different behavioral pattern with respect to response time values. The plots of response time values across 339 users are provided in Figure 4.18.

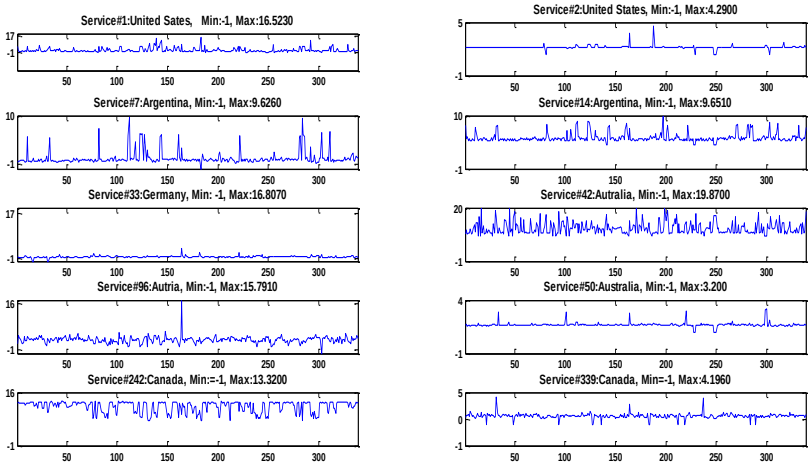

Figure 4.18: Response time values and country of origin of 10 randomly selected Web services

We used 50% of the 339 response time values in all the Web services to train our models and remaining points are used for the prediction. A value of $m=1$ is used for all cases. To measure the performance of our models, Mean Squared Error (MSE) (in Eq. (4.22)), Mean Absolute Error (MAE) , and Symmetric Mean Absolute Percentage Error (SMAPE) are used as the key performance indicators.

$$MAE = \frac{1}{N}\sum_{t=1}^{N}\left|T_t - \hat{T}_t\right| \tag{4.25}$$

$$SMAPE = \frac{100}{N}\sum_{t=1}^{N}\frac{\left|T_t - \hat{T}_t\right|}{T_t + \hat{T}_t} \tag{4.26}$$

Here, T_t and $\hat{T}$ are the real and predicted values, respectively. MSE and MAE represent the difference between the actual value and the predicted value. SMAPE is an accurate measure based on percentage error. Its value belongs to the range 0% and 100%. If the practical value of SMPAE is near 0%, the predicted results are quite accurate.

It can be seen from Table 4.8 that model M4 has consitently low values for the three indicators – MAE, SMAPE and MAE. SMAPE values of M4 are within the range of 3% to 5%. SMAPE values of M1, M2, and M3 are within the range of 4% to 60%, 2% to 7%, 7% to 25% respectively. Next to M4, model M2 has otperformed the other models. Finally M3 superceeds M1 in terms of prediction performance. We have ploted the indicator values for all models against the selected 10 Web services in Figure 41.19 to Figure 4.21.

This observation we have maded can be further verified from the three plots. The experiment clearly indicates that our proposed models can be used for prediction of Web service QoS or trust values under varied behavioral conditions.

Table: 4.8 : Performance indicators values for for models M1,M2,M3 and M4

Model	Indicator	Web service									
		#1	#2	#7	#14	#33	#42	#50	#96	#242	#339
M1	MSE	0.2884	0..2630	1.2502	0.8088	0.4645	0.2650	1.1014	0.3545	1.2504	0.1925
	SMAPE	9.6461	10.6742	5.9185	59.1265	13.3595	10.7712	5.9035	11.4904	5.9210	5.9452
	MAE	0.9120	0.1252	0.2433	1.5555	0.1210	0.1223	0.2093	0.0834	0.2413	0.2087
M2	MSE	0.0985	0.0921	0.1284	0.1987	0.1042	0.0991	0.1284	0.1030	0.1284	0.1251
	SMAPE	3.7679	3.8099	5.7534	6.3373	3.7477	3.8815	5.6575	3.8011	5.6462	5.624
	MAE	0.0836	0.0974	0.03417	0.1925	0.1001	0.0941	0.2550	0.1141	0.2814	0.0724
M3	MSE	0.0776	0.0804	0.3622	0.3000	0.1523	0.0825	0.3543	0.0795	0.4547	0.0792
	SMAPE	11.4854	11.5154	15.3365	11.1517	8.1605	8.3654	12.9832	11.4780	14.7926	20.234
	MAE	0.1048	0.1031	0.1927	0.1925	0.1367	0.1021	0.1076	0.1048	0.1925	0.1854
M4	MSE	0.0150	0.0147	0.0186	0.0140	0.0137	0.0152	0.0187	0.0150	0.0187	0.0146
	SMAPE	3.6746	3.5931	3.3879	3.3390	3.3142	3.9078	3.4277	3.6755	4.4299	4.5421
	MAE	0.0144	0.0154	0.0154	0.0129	0.0151	0.0145	0.0155	0.0144	0.0154	0.0144

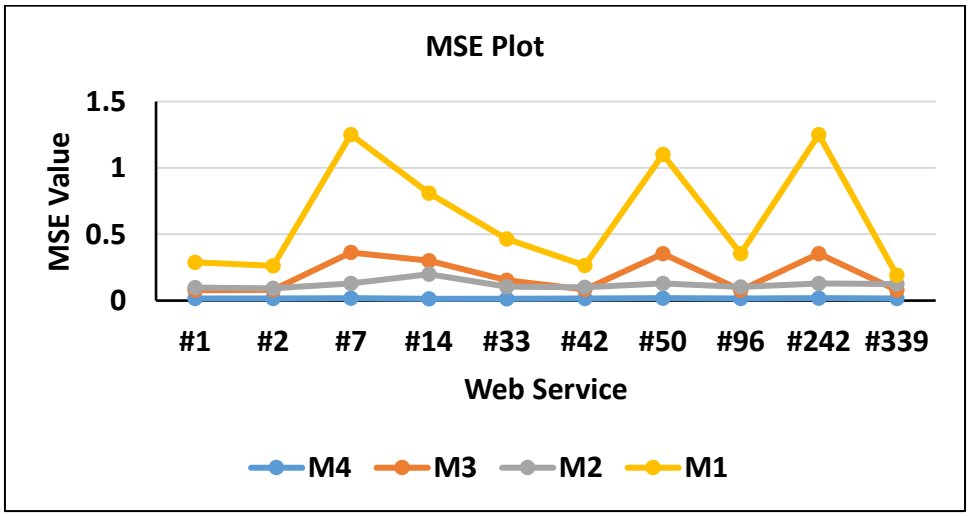

Figure 4.19: Plot of Mean Square Error (MSE) values for all models accorss 10 selected Web services.

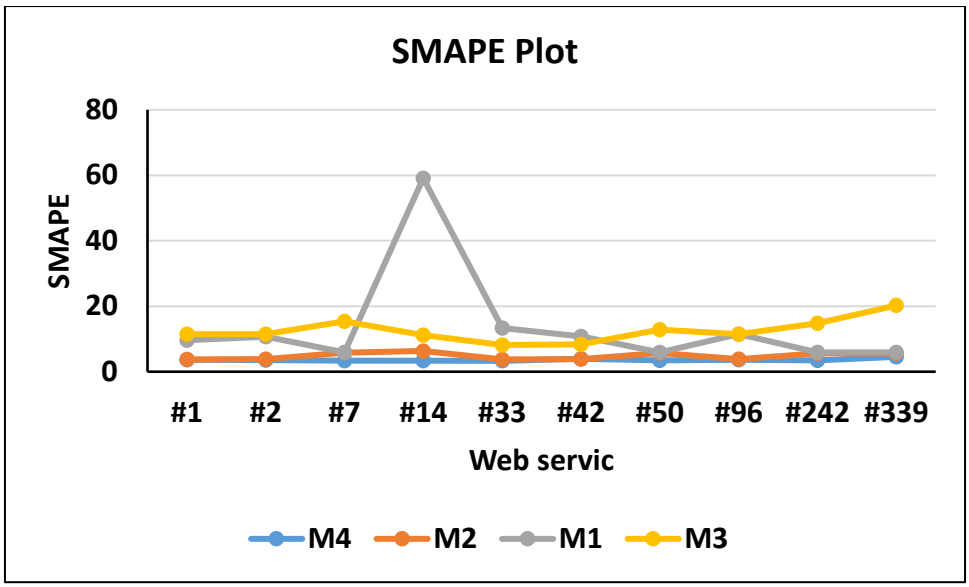

Figure 4.20: Plot of Symmetric Mean Absolute Percentage Error (SMAPE) values for all models accorss 10 selected Web services.

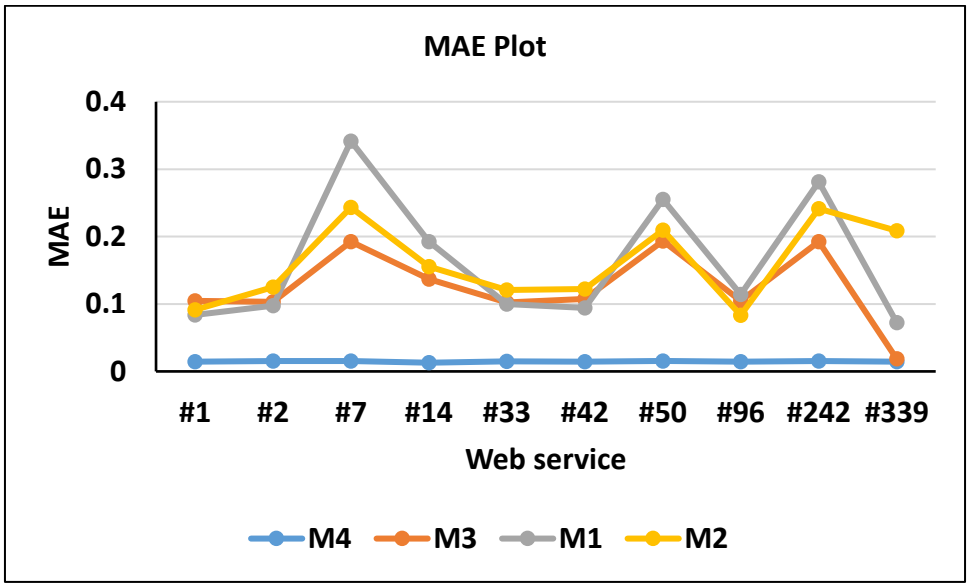

Figure 4.21: Plot of Mean Absolute Error (MAE) values for all models accorss 10 selected Web services.

4.5 Conclusion

In this chapter, we have proposed an ensemble prediction using different GP models as its members. While using GP model, the selection of the kernel is very important as it is our prior knowledge about the input data. Using our model selection approach based on the likelihood measure, we can select the ensemble members to participate in the next prediction step. One of the main concern about GP model is its computation time, which is $O(n^3)$. To reduce the computational cost, a novel sleep and recovery mechanism and online update technique of the covariance matrix of a GPR are used. The application of our ensemble models and as well as the other two models proposed in Chapter 3 in the sequential prediction has been experimentally shown. All the models are tested in dynamic environment, where the Web service provider and the third party information sources exhibit different behaviours over the transactions. The dynamic characteristics of trust and reputation values over time have also been considered in the experiment by using a time decay factor.

The experimental evidences show that the proposed framework can compete with the existing models Except for one model, all proposed models are found to have better prediction accuracy than the state of the art trust model. Further, the superiority of the framework of our ensemble model over the prediction model in Chapter 3 has also been experimentally established.