AN APPROACH TO FIND THE TRANSITION PROBABILITIES IN MARKOV CHAIN FOR EARLY PREDICTION OF SOFTWARE RELIABILITY

1Hitesh Rajput, 2Lalit Kumar Singh
1Dept. of Applied Mathematics, Indian Institute of Technology (BHU), India
2Dept. of Computer Science & Engineering, Indian Institute of Technology (BHU), India

Abstract- Early prediction of software reliability can help organizations to make informed decisions about corrective actions. Early prediction of software reliability is a challenging problem because of many uncertainties associated with components under development. Many researchers have addressed this problem by proposing the appropriate models based on Markov Chain but there is a major challenge to find the transition probabilities between the states. In this paper, we propose an approach to address this problem by making the use of Artificial Neural Network and Hidden Markov Model.

Keywords: Software reliability; DTMC; HMM; ANN

I. INTRODUCTION

Researchers have given / proposed many software reliability models to ensure the reliability of the software. But unfortunately till date no model fits in all kinds of software, means there is no generic model that can be used to predict the reliability of all kinds of software. These models are analytically derived from assumptions. The emphasis is on developing the model, the interpretation of the model assumptions, and the physical meaning of the parameters. These types of reliability models are known as “software reliability growth models”. These models attempt to statically correlate defect detection data with known functions such as an exponential function. If the correlation is good, the known function can be used to predict the future behavior. The reasons that these reliability models lack the needed strength to excel in eliminating errors in software environment are:

The misconception of fault & failure phenomena
Inaccurate modeling parameters
Difficulty in selecting the reliability models
Difficulty in building the software operational profile

So, different approaches have been started i.e. to predict the reliability based on the design parameters of software. The models based on this concept are known as “defect density” models. These models use code characteristics such as lines of code, design characteristics such as calculate the weight of a class or architectural characteristics. Many frameworks are based on Markov chain for early software reliability prediction. The biggest challenge in this framework to find the transition probabilities in between the states.

In this paper we have introduced an approach that can be used to find the transition probabilities, using Hidden Markov Model and Artificial Neural Network. The paper is organized as follows. Section 2, describes the approach to determine the transition probability by taking an already existing framework, for early prediction of the software reliability. Model computation and memory requirements are given in section 3. Section 4 describes the model limitations and section 5 concludes the paper.

II. RELATED WORK

2.1 Software Reliability prediction framework

Generally the software reliability prediction framework, based on Markov chain, consists of three phases, as shown in Figure 1.

Phase1- Any software consists of logical states which keep on changing during operational to produce some output. In this phase the software is divided into logical states. Identification of these states solely depends on the domain.

Phase2- In this phase, identification of the transition probabilities between the states (that are identified during phase1) is to be done, for which we have proposed an approach, which is described in next subsection.

Phase3- In this phase, the reliability is computed by solving the Markov chain, constructed in phase 1 and 2, as given in [5].

2.2 ANN & HMM based transition matrix prediction
2.2.1 DTMC Model Creation
As discussed, any software can be thought of collection of various logical states, and the control keeps on switching from one state to another state. We model the software using DTMC. This DTMC is characterized by a transition probability matrix \( P \).

\[
P = \begin{bmatrix}
S_1 & S_2 & S_3 \\
0 & P_{12} & P_{13} \\
0 & 0 & P_{23}
\end{bmatrix}
\]

Here \( S_1, S_2, S_3 \) are states of the software, which can be behavioral or fault state. Obviously, sum of all the probabilities in each row is equal to 1. Fault state represent that some error has occurred. A DTMC model has Markov property, namely that, given the present state, the future and the past states are independent:

\[
\Pr(X_{t+1} = x_{t+1} | X_t = x_t, \ldots, X_0 = x_0) = \Pr(X_{t+1} = x_{t+1} | X_t = x_t)
\]

2.2.2 Model Creation
The prediction of transition probabilities is regarded as causal forecasting problem: the mapping between inputs and outputs of ANN can be written as:

\[
f_i = f(\theta_i) \quad \text{for generalization training}
\]

\[
f_i = f(\theta_i-1) \quad \text{for prediction training}
\]

where \( \theta_i \) is the (updated) model parameter (transition probability) and \( \theta_i-1 \) is the set of last model parameter (transition probability).

The observations are visible to the observer who interacts with the software system. In an abstract level, a software system consists of 3 nodes- input node, processing node and output node, see Figure 2, which is popularly known as the context diagram or Data Flow Diagram Level-0 (DFD Level-0). Processing node consists of the various states and there transition takes place in between these states. As only the observations are visible to the observer, it resembles HMM. The probability to switch to a particular state is not known to an observer, for which we have devised an approach which is described in section 2. Our model consists of the following:

1. Initially the transition probabilities are assigned, say, by using the domain knowledge, experts or similar existing functional software.

2. During testing time, we train our ANN. In this phase we shall try to find out the state sequence given the observed sequence of featured vectors. In this phase, we will find out the updated parameters of the model.

3. Lastly we will use the Expectation-maximization (EM) algorithm to compute and update the model parameters.

Notations:
1. \( N \) = total number of states.
2. \( T \) = total number of observations.
3. \( \pi_{ij} \) = transition probability from state \( i \) to state \( j \)
4. \( \pi_i = 1 \)

\[
\sum_{j=1}^{N} \pi_{ij} = 1
\]

5. \( \theta_i \) = Initial probabilities of states

\[
\sum_{i=1}^{N} \theta_i = 1
\]

6. \( \hat{O} \) = Observation vector, set of independent observations in sequence.

\[
\hat{O} = \{ \hat{O}_1, \hat{O}_2, \ldots, \hat{O}_T \}
\]

We can also term it as feature vector.

7. \( y_{ij}(\hat{O}) \) = conditional probability that at present time, the control is in state \( i \) at position \( t \), while \( \hat{O} \) is given.

\[
y_{ij}(\hat{O}) = \Pr(\hat{O}_t = t, s_t = j | \hat{O} = \hat{O})
\]

8. \( y_{ij}(\hat{O}) \) = conditional probability that at present time, the control is in state \( i \) at position \( t \) and in state \( j \) at position \( t+1 \), while \( \hat{O} \) is given.

\[
y_{ij}(\hat{O}) = \Pr(\hat{O}_{t+1} = j, \hat{O}_t = t, s_t = i | \hat{O} = \hat{O})
\]

9. \( p_{ij}(\hat{O}) \) = Probability law of \( \hat{O} \) for its generation

2.2.3 Parameter Estimation
The transition matrix \( \pi_{ij} \) needs to be updated iteratively and this is to be done by ANN. We can estimate this parameter through EM algorithm, which consists of 2 steps:

Step 1- In this step we will compute the values of \( \hat{O} \) and \( \hat{O}_j \) for \( i, j = 1, 2, \ldots, N \) and \( t = 1, 2, \ldots, T \). This is known as E step.

Step 2- In this step we will update the parameters. This is known as M step.

Let the software is under the testing phase. The probability of the state sequence can be written as:

\[
\Pr(\hat{O}_1, \hat{O}_2, \ldots, \hat{O}_T) = \Pr(\hat{O}_1) \Pr(\hat{O}_2 | \hat{O}_1) \Pr(\hat{O}_3 | \hat{O}_2, \hat{O}_1) \ldots \Pr(\hat{O}_T | \hat{O}_{T-1}, \ldots, \hat{O}_1)
\]

The observation \( \hat{O}_t \) is independent of other observations as well as states, for a given state \( s_t \). The observation \( \hat{O}_t \) is generated with a fixed probability law. Here we have assumed that the observation data is discrete. Let \( p_{ij} \) specifies the probability law of vector \( \hat{O}_t \). Then \( \hat{O}_t \) will contain many but finite value, so \( p_{ij} \) is specified by probability mass function (pmf). The joint probability of the observation vector and state vector is given by:

\[
\Pr(\hat{O}, \hat{S}) = \Pr(\hat{S}) \Pr(\hat{O} | \hat{S})
\]

We can easily find out the probability of the Observation vector i.e. probability of sequence of observations, with the help of total probability formula:

\[
\Pr(\hat{O}) = \sum_{\hat{S}} \Pr(\hat{O} | \hat{S}) \Pr(\hat{S})
\]

\[
= \sum_{i} \pi_i \sum_{j} \pi_{ij} \sum_{k} \pi_{jk} \sum_{l} \pi_{lk} \ldots \pi_{lN} \cdot p_{ij} \cdot \ldots \cdot p_{lj} \cdot \ldots \cdot p_{ln} \cdot p_{ln}
\]

Suppose that presently, the control is in state \( i \), at position \( t \) and the Observation vector is given as \( \hat{O} = \{ \hat{O}_1, \hat{O}_2, \ldots, \hat{O}_T \} \). So the probability that every time a particular state is at position \( i \), irrespective of the observation vector is given by:

\[
I_{ij}(\hat{O}) = \Pr(\hat{O}_t = t | \hat{O})
\]

\[
= \sum_{i} \Pr(\hat{O}_t = t, s_t = i | \hat{O})
\]

Similarly,
Now, to calculate the transition probability in between the states, we can work with the following DTMC model:

![Figure 3. DTMC model](image)

Given the observation sequence, suppose the probability that the present state is S1 at position 1 is \( q \). Also given the same observation sequence, suppose the probability that the present state is S1 at position 1 and in state at position 2 is \( r \). So the transition probability in between S1 and S2 will be \( \frac{r}{q} \).

So, in general, the transition probability in between state \( i \) to \( j \) is given by:

\[
\pi_{ij} = \frac{\text{sum of the conditional } P_{ij} \text{ of being in state } i \text{ at all positions and in state } j \text{ at all subsequent positions}}{\text{sum of the conditional } P_{ij} \text{ of being in state } i \text{ at all positions}}
\]

\[
\pi_{ij} = \frac{\sum_{t=1}^{N} y_{ij}(t)}{\sum_{t=1}^{N} y_{ii}(t)}
\]

The efficient solution for \( \pi_{ij}(t) \) and \( y_{ij}(t) \) can be computed with the Forward-Backward Algorithm. Forward probability is the joint probability of observing the first \( t \) vectors \( \hat{O}_{1:t} = 1, \ldots, t \) and being in state \( i \) at time \( t \). Let us denote the forward probability by \( \alpha_i(t) \). Therefore,

\[
\alpha_i(t) = \Pr(O_{1:t}, s_t = i) = \sum_{s_{t-1}} \alpha_i(t-1) \Pr(O_t | s_t = i) \Pr(s_t = i | s_{t-1})
\]

Similarly Backward probability is the conditional probability of observing the vectors after time \( t \). \( \hat{O}_{t+1:T} = 1, \ldots, T \), given that at time \( t \), the state is \( i \). Let us denote the forward probability by \( \beta_i(t) \). Therefore,

\[
\beta_i(t) = \Pr(s_t = i | O_{t+1:T})
\]

Also, \( \beta_i(T) = 1 \). From 1,

\[
\pi_{ij}(t) = \frac{\Pr(O_{1:t}, s_t = j)}{\Pr(O_{1:t})} = \frac{\Pr(s_t = j | O_{1:t}) \Pr(O_{1:t})}{\Pr(O_{1:t})} = \frac{\beta_j(t)}{\alpha_i(t)}
\]
Using 4, \[ \frac{e_{ij}(t+1)}{e_{ij}(t)} = \frac{e_{ij}(t+1)}{e_{ij}(t)} \] 

From 2,
\[ \begin{align*}
\gamma_{ij} &= Pr(s_i = l, s_{t+1} = j | \tilde{O}) \\
&= \frac{Pr(s_i = l, s_{t+1} = j | \tilde{O})}{Pr(\tilde{O})} \\
&= \frac{Pr(\tilde{O} \cup \ldots \cup \tilde{O}, s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \frac{Pr(\tilde{O} \cup \ldots \cup \tilde{O}, s_{t+1} = j, s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \frac{\alpha_t(\tilde{O})Pr(\tilde{O} \cup \ldots \cup \tilde{O}, s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \frac{\alpha_t(\tilde{O})Pr(\tilde{O} \cup \ldots \cup \tilde{O}, s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \frac{\alpha_t(\tilde{O})Pr(\tilde{O} \cup \ldots \cup \tilde{O}, s_{t+1} = j, s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \frac{\alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j)}{Pr(\tilde{O})} \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
&= \alpha_t(\tilde{O})Pr(s_{t+1} = j | s_i = l, s_{t+1} = j) \\
\end{align*} \]

So, here Step1 gets completed. Now for Step2, update the parameter \( \gamma \) using equation 3, with the help of equation 6.

\[ p_{ij} = \frac{\sum_{t=1}^{T} \gamma_{ij}(t)}{T} \]

2.2.4 Prediction Procedure
In this process the \( \gamma \) keeps on updating with new \( \gamma \) along with the testing data. The following steps shall be followed to train the ANN:
1. Assign the initial transition probabilities by using expert’s knowledge, domain knowledge, standard documents and similar project, which is currently in use.
2. Compute the transition probability \( \gamma \) between the states using equation 3.
3. Cycle \( \gamma \) through net, for each state.
4. Calculate error derivative between output transition probability \[ \gamma \] and target output transition probability (actual) \[ \gamma \].
5. Back propagate the summed product of the weights and errors in the output layer to calculate the error on the hidden units.
6. Update weights according to the error on that unit.
7. Repeat the steps until error is low and the net settles.

There exist many available algorithms for ANN training.

2.2.5 Prediction Procedure for a similar kind of application
The prediction can be much faster and more accurate, if similar kind of application exists. Suppose there are two similar projects: \( \text{proj}_1 \) and \( \text{proj}_2 \) and \( \text{proj}_1 \) is finished already and \( \text{proj}_2 \) is currently under testing phase. Let \( \gamma_{ij}^{\text{proj}_1} \) is the set of the transition probabilities in between state i and j (for all states), of \( \text{proj}_1 \) collected during its testing / running process, from beginning to current time. Also, let \( \gamma_{ij}^{\text{proj}_2} \) is the set of the transition probabilities in between state i and j (for all states) of \( \text{proj}_2 \) up to current time \( \gamma_{ij}^{\text{proj}_2} \). This data set is very small, for accurate prediction, in the early phase of testing and ANN cannot be trained well with this limited data set. But because these two projects are similar, these data sets can be combined and same prediction procedure can be followed to predict the transition probabilities in the future.

III. MODEL COMPUTATION AND MEMORY REQUIREMENTS
Let the total number of states are \( N \) and order of \( N \times L \) hidden nodes are required. Also maximum order of \( N \times N \) amount of computation is needed as no equations in section 2.2.3 is taking more than this order.

IV. MODEL LIMITATION
Every hidden node of ANN processes information of one state. The transition probabilities will be \( N \times N \), can be verified from section 2.2.1. So, if the number of states will be more the number of hidden layers will be huge which will slow down the performance of computation. Also the validity of the approach given in section 2.2.5, for similar kind of application will keep on deteriorating if the number of logical states of both the projects differs and which is quite possible if different heads have been put to architect the software.

V. CONCLUSION AND DISCUSSIONS
In this paper we explored the topic of early prediction, on the basis of software architecture, in which the major challenge was to find out the transition probabilities in between the states of the DTMC model. We have resemble our DTMC model as an HMM because only the state emissions are known to the observer. Then we have explained the process of model creation and to estimate and update the model parameter (transition probability) and after that the procedure to predict the transition probabilities. We have also proposed an approach to predict the transition probability using similar projects.
probabilities for a similar kind of application, in which the prediction will be much accurate and faster. Adaptive prediction could help to correct the predictors. Finally we stated our model’s computation and memory requirements and its limitations.

In the future, we shall try to develop a tool based on our approach and will use that tool to one of the live projects to come out with the results.

REFERENCES