Bootstrapping Trust of Web Services through Behavior Observation

Hamdi Yahyaoui[1] and Sami Zhioua[2]

[1] Computer Science Department, Kuwait University
P.O. Box 5969, Safat 13060, State of Kuwait
hamdi@sci.kuniv.edu.kw

[2] Information and Computer Sciences Department, KFUPM
P.O. Box 958, Dhahran 31261, KSA
zhioua@kfupm.edu.sa

Abstract. We present in this paper a new Web services trust bootstrapping technique, which consists in observing several interactions of a user with a Web service. The obtained observations sequence is modeled as a Hidden Markov Model (HMM) and matched against pre-defined trust patterns in order to assess the behavior of the Web service under observation. The pre-defined trust patterns are specifications of possible behaviors of Web services. Based on the matching result, an initial trust value is assigned to the observed Web service. Our experimental results show that our technique has good precision and recall rates together with a fair distribution of trust values.

Keywords: Trust, Web services, Bootstrapping, Hidden Markov Model.

1 Introduction

Web services are becoming nowadays a powerful technology which allows users to invoke, in a transparent way, distant services through standard Web protocols. With a variety of functionalities, plenty of Web services are deployed in the internet. Consumers judge the performance of these services based on their non-functional quality attributes such as response time, availability, throughput, etc. Such judgement reflects how much a Web service is trusted for performing a certain task in the expected way. The non-functional quality attributes can be leveraged to derive a trust value for a Web service as suggested in [14]. Trust can be used as an indicator for the possible future behavior of a Web service. One of the important issues in establishing the trust for Web services is the assignment of a trust value to unknown Web services. This is known as the bootstrapping issue. Bootstrapping is even more challenging when there is an absence of user recommendations. Unfortunately, this issue did not receive much attention in the related work as will be shown later, while in reality it has an important impact on the robustness of the elaborated trust model. The aim of our work is to devise a new bootstrapping technique based on trust patterns and Hidden Markov Model (HMM) in the absence of recommendations. The proposed technique focus on the recognition of the behavior of a Web service rather than deriving a trust value from a simple interaction.

The rest of the paper is as follows. In section 2, we recall some definitions about HMM. Section 3 is devoted to the review of published bootstrapping techniques. Section 4 is dedicated to the presentation of our bootstrapping technique. In section 5, we provide experimental results and analysis. Finally, some conclusions and future works are drawn in Section 6.

2 Background

This section is devoted to the presentation of some definitions that are useful to have a clear understanding of our technique.

2.1 HMM Definition

A Hidden Markov model (HMM) [11] is a Markov chain where states are not completely observable. That is, the underlying mechanics corresponds to a Markov chain but the state of the system at a given moment is not exactly known. Instead, at each time step, a stochastic observation is generated based on the current state.

Definition 1. *Hidden Markov Model*
A HMM is a tuple $(\mathcal{S}, T, \mathcal{O}, Q, \pi)$ *where*

- $\mathcal{S}$ *is a set of* N *states* $\{s_1, s_2, \ldots, s_N\}$. *Besides,* s^t *refers to the state at time step t. Note the difference between the subscript and superscript versions* s_1 *and* s^t.
- $T : \mathcal{S} \rightarrow \Pi(\mathcal{S})$ *is a state transition function which maps each state* $\mathcal{S}$ *to a probability distribution over* $\mathcal{S}$. $T_{s_i \rightarrow s_j}$ *denotes the probability of making a transition from* s_i *to* s_j.
- $\mathcal{O}$ *is a set of* M *observations* $\{o_1, o_2, \ldots, o_M\}$. *Similarly to states, let* o^t *be the observation at time step t.*
- $Q : \mathcal{S} \rightarrow \Pi(\mathcal{O})$ *is an observation function.* Q_s^o *denotes the probability of observing o while in state* s.
- π *is the initial state distribution, with* $\pi(s)$ *denotes the probability of* s *being the initial state.*

For simplicity, we use the compact notation $\mathcal{H} = (T, Q, \pi)$.

2.2 Probability of Accepting an Observation Sequence

One of the basic problems with HMMs is how to compute the probability of an observation sequence given a HMM model. Let $O = o^1 o^2 \ldots o^n$ be an observation sequence of length n and let $\mathcal{H} = (T, Q, \pi)$ be an HMM model. The goal is to compute the conditional probability $P(O|\mathcal{H})$.

The sequence O can be generated from $\mathcal{H}$ using different state sequences of length n. Hence, one way of computing $P(O|\mathcal{H})$ is to sum over all possible state sequences of length n. Let $S = s^1 s^2 \ldots s^n$ be such a sequence. The probability that O is generated using S is the probability of getting observation o^1 from state s^1, o^2 from s^2 and so on until o^n from s^n, that is,

$$P(O|S,\mathcal{H}) = Q_{s^1}^{o^1} Q_{s^2}^{o^2} \dots Q_{s^n}^{o^n} \tag{1}$$

However, to sum over all possible state sequences, one needs instead the joint probability of O and S, namely, $P(O,S|\mathcal{H})$. By the Bayes theorem we have:

$$P(O,S|\mathcal{H}) = P(O|S,\mathcal{H})P(S|\mathcal{H}) \tag{2}$$

Where $P(S|\mathcal{H})$ is the probability of going through the state sequence S:

$$P(S|\mathcal{H}) = \pi(s^1) T_{s^1 \to s^2} T_{s^2 \to s^3} \dots T_{s^{n-1} \to s^n} \tag{3}$$

We have now all the ingredients to illustrate the detailed computation of $P(O|\mathcal{H})$:

$$\begin{aligned} P(O|\mathcal{H}) &= \sum_{S=s^1 s^2 \dots s^n} P(O|S,\mathcal{H})P(S|\mathcal{H}) \\ &= \sum_{S=s^1 s^2 \dots s^n} \pi(s^1) Q_{s^1}^{o^1} T_{s^1 \to s^2} Q_{s^2}^{o^2} \\ &\qquad \dots T_{s^{n-1} \to s^n} Q_{s^n}^{o^n} \end{aligned} \tag{4}$$

Equation (4) is clearly not easy to compute. For one reason, it sums over all possible state sequences whose number is in the range of N^n. Adding the fact that for each such sequence $2n$ operations are performed, the total number of computations required to compute $P(O|\mathcal{H})$ is in the order of $2nN^n$. Hence, Equation (4) is not an efficient way of computing $P(O|\mathcal{H})$. A better approach is what is known as Forward-Backward procedure [1].

The Forward-Backward approach to compute $P(O|\mathcal{H})$ is based on defining the forward probability

$$f_t(i) = P(O, s_i|\mathcal{H}) \tag{5}$$

$f_t(i)$ is the partial probability of observing the sequence $O = o^1 o^2 \dots o^t$ and end-up in state $s^t = s_i$. The key point is that $f_t(i)$ can be defined inductively as follows:

$$f_t(i) = \begin{cases} \pi(s_i) Q_{s_i}^{o^t} & \text{if } t = 1 \\ \sum_{j=1}^{N} \big(f_{t-1}(j) T_{s_j \to s_i}\big) Q_{s_i}^{o^t} & \text{if } 1 < t \leq n \end{cases}$$

$P(O|\mathcal{H})$ can be expressed in terms of the forward probability simply as:

$$P(O|\mathcal{H}) = \sum_{i=1}^{N} f_n(i) \tag{6}$$

The Forward-Backward approach to compute $P(O|\mathcal{H})$ requires on the order of $N^2 n$ computations, which is clearly better than the $2nN^n$ of Equation (4). The key point is that since there are only N states, all the possible state sequences will remerge into these N states, no matter how long the observation sequence.

3 Related Work

Bootstrapping a new element in a system means assigning an initial trust/reputation value for it. Such value has an impact on the security of the whole system since there is no prior knowledge about the possible behavior of the new element. Henceforth, bootstrapping is paramount for the security of software and systems. The bootstrapping issue was studied in few research initiatives. In what follows, we present these initiatives and pinpoint their pros and cons. It is worth to mention that the terms reputation and trust have different meanings. Reputation denotes a group opinion about a peer while the trust denotes an individual opinion.

3.1 Default Value Technique

The default value technique was designed for peers collaborating in a Mobile Ad hoc Network (MANET). In this technique, the new peers that join the network get a default reputation value [7]. The value is generally considered as a threshold under which a peer is considered as malicious. One disadvantage of this technique is that, depending on the default initial value, it can either favor existing peers or new peers that join the network. If the initial reputation is high, existing peers are disadvantaged, since the new peer is assigned a value that can be higher than existing peers that collaborated a lot and strived to have a good reputation. This encourages malicious peers to leave and join again the network with new identities to avoid their bad reputation (this is known as white-washing). A low reputation value will discourage new peers from being involved in collaborations.

3.2 Punishing Technique

The punishing technique [2] is proposed as a solution to overcome the white-washing issue. By giving a low reputation value to the new peer, it ensures that a malicious peer that is trying to leave and join again the network will not gain that much. However, a very low value will make the new peers disadvantaged and perhaps no existing peer will collaborate with them, which makes them isolated in the network.

3.3 Adaptive Technique

The adaptive technique is part of a trust model for Web services, which is proposed in [5]. In this technique, a new Web service which is willing to collaborate is assigned a reputation value that depends on the rate of maliciousness, which is defined as the ratio of the number of collaborations (considered as transactions in that work) where the Web services defect, to the total number of collaborations. This requires tracking each collaboration quality. A Web service rater marks collaboration as acceptable if the Web service did collaborate as expected and defective in the opposite case. An unknown Web service is assigned a high initial reputation value when the rate of maliciousness is low and a low initial reputation value when that rate is high. The issue with this technique is that a leaving malicious Web service can try to rejoin the network with a new URL and get a better reputation value than his old value if he leaves the network with a low reputation value while the maliciousness rate is very low at that moment.

3.4 Prediction Technique

In [4], the author proposed a HMM-based prediction model to assess a provider reputation when adequate number of rater recommendations is not available. A service consumers HMM is trained using the recommendations provided by some evaluators. Once a reliable model is developed, the high and low reputations of the services are predicted. In the next step, the service consumer compares all the predicted provider reputations. The provider which has the highest predicted reputation for the next time instance is chosen for interaction. After each interaction, the observed behavior values and present recommendations are input to the HMM for the sake of refining the model. This model is used for predicting a single reputation value for a Web service and can't be used to classify a Web service behavior. Such classification is needed since at one time instance a service reputation value can't reveal accurately its long term behavior. Besides, it is very often to have a lack of recommendations about a Web service which is unknown to raters. Hence, assuming always the existence of recommendations is not always a realistic assumption.

4 A New Pattern-Based Bootstrapping Technique

The techniques mentioned in the previous section focus on a local evaluation of the reputation level of an unknown peer or Web service. This cannot provide an accurate assessment of the trustworthiness of a Web service. We follow a global strategy where a Web service is evaluated during a certain time frame in which a sequence of trust observations is built. Based on this sequence, we are able to judge, whenever possible, the kind of behavior a Web service has, i.e., the category to which a Web service belongs (trusted, malicious, redemptive, etc.). More precisely, our technique consists in first evaluating a Web service during a certain period of time. During that period, a sequence of trust observations is built. Each trust observation (T for Trusted and U for Untrusted) denotes the degree of compliance of the non-functional quality attributes of a Web service (e.g. response time, availability, reliability, etc.) with the announced quality attributes (as part of Service Level Agreement). The obtained sequence of trust observations is then matched against pre-defined trust patterns. The objective of the matching step is to have a global judgement about the behavior of a Web service and derive a trust value based on the recognized behavior. In what follows, we provide the ingredients of our technique.

4.1 Trust Patterns

In this section, we define trust patterns which refer to particular Web service behaviors. A trust pattern is a sequence of trust observations from which a clear conclusion about the behavior of a Web service can be drawn. We provide a formal characterization of trust pattern categories:

- Trustworthy: T^+
- Oscillating: $(T^+.U^+.T^+.U^+)^*$ Or $(U^+.T^+.U^+.T^+)^*$
- Malicious: U^+

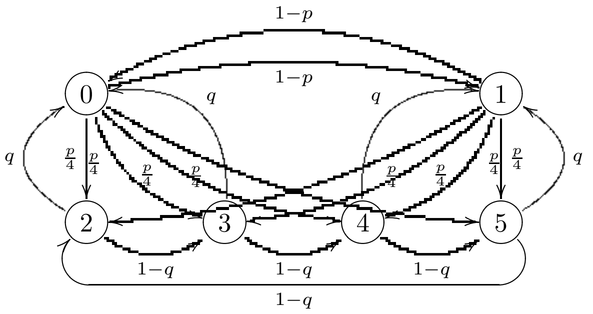
- Betraying: $(T^{+}.U^{+})^{*}$
- Redemptive: $(U^{+}.T^{+})^{*}$

Where T and U denote respectively a Trusted/Untrusted observation, while the operators $^{+}$ and * denote respectively a non-empty sequence and the repetition of a sequence. A trust observation is generated after one interaction with a Web service while a sequence of trust observations is constructed during a certain time frame. We restrict the description of patterns to the aforementioned categories but the extension of these categories is possible.

The sequence notation for a trust pattern is used for readability aims. However, we use another notation for specifying such patterns. Indeed, a trust pattern can be represented using a HMM. The advantage of this representation is that a HMM is more expressive than a static sequence of trust observations. Indeed, a HMM is more compact and it allows to represent a range of trust observation sequences falling in the same category. For example, consider the two following trust observation sequences which clearly fall in the category of an oscillating pattern:

1. T.U.T.U.T.U.T.U.T.U.T.U
2. T.T.U.U.T.T.U.U.T.T.U.U

The two trust observation sequences can be represented by a single HMM of the form:



Where states 0, 2 and 3 generate always the trust observation T and the states 1, 4 and 5 generate always the trust observation U. Another advantage of using HMMs in specifying trust patterns is that a HMM can capture patterns resulting from alternating between the two patterns above.

The five HMMs are trained given a single or multiple trust observation sequences. The training process is inspired by the seminal work of Rabiner[11]. Given a trust observations sequence $O = o^1 o^2 \dots o^n$, the training aim is to find how to adjust the model parameters $\mathcal{H} = (T, Q, \pi)$ to maximize $P(O|\mathcal{H})$. There is no optimal way for estimating the model parameters so that to maximize the probability of acceptance of a trust observations sequence. The best one can do is to choose the model parameters $\mathcal{H} = (T, Q, \pi)$ such that $P(O|\mathcal{H})$ is locally maximized using an iterative procedure such as the Baum-Welch method [1].

4.2 Trust Observation Generation

A trust observation (T or U) is generated during an evaluation period through the assessment of quality attributes of a Web service. Each quality attribute (such as response

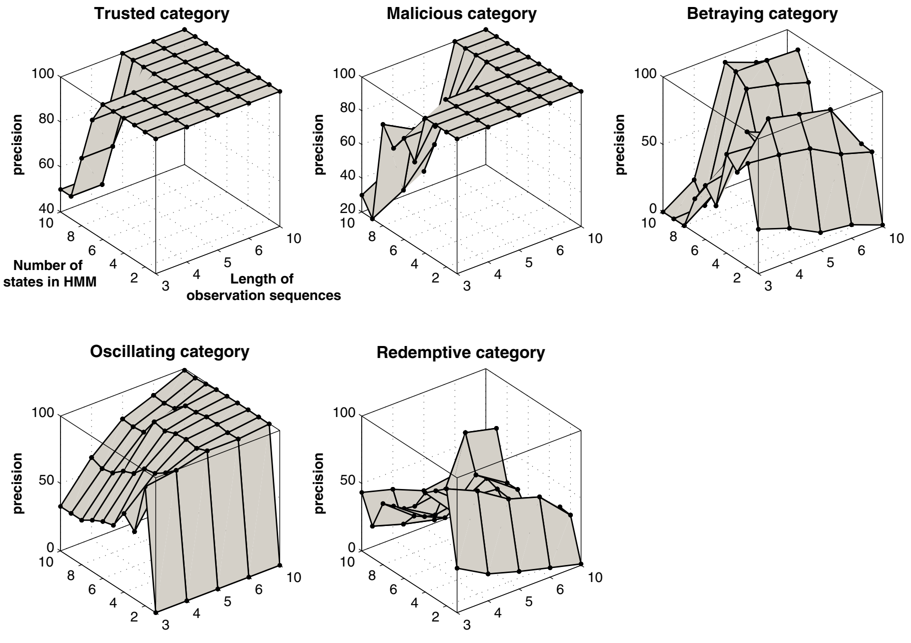


Fig. 1. The precision of classifying a bank of observation sequences as we increase the number of states of the HMM (y-axis) and the length of the observation sequences (x-axis)

time, availability, etc.) has a certain expected and actual value. The expected values are announced by Web service providers through a Service Level Agreement (SLA) with consumers. The actual values are computed during the evaluation period and matched against the expected ones. To assess the trustworthiness of a Web service during one interaction, we compute the Root Mean Square Error (*RMSE*):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(v_i - v_i')^2}{n}} \tag{7}$$

Where v_i denotes the actual value of a quality attribute q_i, v_i' the expected value of a quality attribute q_i and n the total number of assessed quality attributes. We assume that all values v_i and v_i' are normalized before computing *RMSE*. This means that *RMSE* has a value between 0 and 1. During an interaction i, a trust observation o^i is generated based on the following equation:

$$o^i = \begin{cases} T & \text{if } RMSE < \xi \\ U & \text{Otherwise} \end{cases} \tag{8}$$

Where ξ denotes a tunable threshold between 0 and 1. It reflects the maximal marginal error that a Web service is allowed to have and over which it is judged as not trusted during the evaluation of an interaction.

4.3 Bootstrapping Trust of Web Services Based on Trust Patterns

We present hereafter an algorithm for bootstrapping the trust of Web services based on matching a trust observations sequence with pre-defined trust patterns (Algorithm 1). The algorithm takes a sequence of trust observations (obtained during the evaluation period) and the HMMs corresponding to the pre-defined trust patterns as inputs and returns the category of the matching pattern. The algorithm associates the Web service to the category, which better matches the sequence, i.e., which maximizes the probability of generating the sequence.

Algorithm 1. PATTERN MATCHING ALGORITHM

```
inputs $O$: Trust observations sequence
1: for each $HMM_i$ corresponding to a trust pattern $p_i$ do
2:     $Pr_i$ = $GetProbAccept(O,HMM_i)$
3: end for
4: Compute $Pr_k = max\ Pr_i$ and determine the pattern $p_k$
5: return $G_k$: The category of pattern $p_k$
```

Based on the matching step, we propose a bootstrapping technique that takes into consideration two parameters: the category to which a Web service WS_i belongs and its behavior during the evaluation step. First, from the trust observations sequence O_i, an individual maliciousness rate is computed. Let O_i be an observation sequence and let $\sharp U_{O_i}$ and $\sharp T_{O_i}$ be the number of untrusted (U), respectively trusted (T), observations in O_i. The rate of maliciousness of observation O_i is defined as:

$$R_i = \frac{\sharp U_{O_i}}{\sharp U_{O_i} + \sharp T_{O_i}} \tag{9}$$

Intuitively, this rate indicates how many times a Web service was untrusted during the evaluation period. However, such rate is not sufficient to distinguish different Web service behaviors. For instance, it can't discriminate between a redemptive Web service who has the following behavior (e.g. $U.U.T.T$) and a betraying one (e.g. $T.T.U.U$). Distinction between those web services is possible given the category G_i each sequence is matched with.

As defined in Section 4.1, trust pattern categories exhibit different levels of trust. For instance, the level of trust in the Trustworthy category is clearly higher than the level in the Oscillating category. Similarly, the redemptive category reveals more trust than betraying category (kind of punishment). To account for these differences, a subjective category weight W_c (a value between 0 and 1) is given to each category c.

Based on the two aforementioned concepts, namely, the individual maliciousness rate and the category weight, we define the initial trust value of a Web service WS_i as follows:

Definition 2. *Initial Trust Value of a Web Service*

Let WS_i be a Web service and O_i a trust observations sequence representing its behavior during the evaluation period. Let G_i be the category associated to O_i according to Algorithm 1, R_i the maliciousness rate of O_i and W_{G_i} the category weight of G_i. The initial trust value of WS_i is defined as follows:

$$T_i = W_{G_i} \times (1 - R_i) \tag{10}$$

5 Experimental Analysis

The training process has been implemented using MATLAB. In addition, HMM Toolbox [10] has been installed and used to validate the results of our own implementation. As expected, both implementations (our and HMM Toolbox) returned exactly the same results. However, HMM Toolbox has been always faster due to some code optimizations. Therefore, we used HMM Toolbox in most of our experiments.

The goal of the experimental analysis is to assess the accuracy and completeness of Algorithm 1. Recall that Algorithm 1 is a classification algorithm to decide which trust category a given Web service belongs to. It is well known that each classification process is characterized by a level of accuracy, which we measure through the metrics *precision* and *recall*.

5.1 Precision and Recall

The main idea of our technique is to represent each trust pattern category using a HMM. Hence each of the 5 different trust pattern categories, namely, Trusted, Malicious, Betraying, Oscillating, and Redemptive, is associated with a different HMM. Every HMM is trained using some typical observations of the associated category. For instance, the Oscillating category HMM is trained using sequences of the form: *T.U.T.U.T.U.T.U.T.U* and *T.T.U.U.T.T.U.U.T.U*.

After the HMMs are constructed, the experiment consists in generating a bank of trust observation sequences and see how accurate Algorithm 1 can classify them. In a nutshell, the idea is to confront the outcome of Algorithm 1 with the true category of each sequence. For the sake of conducting this experiment, the actual category of a given sequence is determined based on an expert judgement.

For each category, we define three variables:

- True Positives (tp): tracks the number of correctly classified sequences;
- False Positives (fp): tracks the number of incorrectly classified sequences;
- False Negatives (fn): tracks the number of sequences which are of the current category but classified into a different category.

These variables are updated as follows. Given an observation sequence O, Algorithm 1 is used to classify it into some category. Assume that this category is X. If the classification is correct, that is, X coincides with the actual category, the variable true positives *tp* for the category X is incremented. If O does not actually belong to the category X, i.e., should be classified into a different category Y, the variable false positives *fp* for X is incremented and the variable false negatives *fn* for Y is incremented.

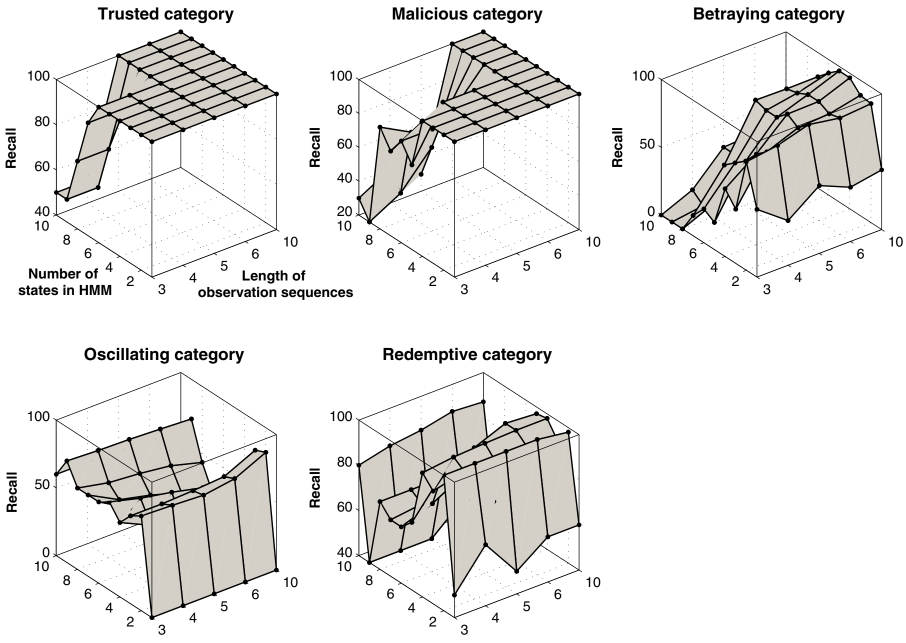


Fig. 2. The recall resulting from classifying a bank of observation sequences as we increase the number of states of the HMM (y-axis) and the length of the observation sequences (x-axis)

The accuracy of the classification for a given category can be expressed as:

$$precision = \frac{tp}{tp + fp} \tag{11}$$

On the other hand, the completeness of the algorithm for a given category is estimated by the recall metrics which is defined as:

$$recall = \frac{tp}{tp + fn} \tag{12}$$

The recall metrics estimates the percentage of sequences belonging to a certain category which has been correctly classified by the algorithm.

In order to reduce the noise factor, we repeated the experiments 10 times and we plotted the average of the 10 experiments. The results are shown in Figures 1 and 2.

Figure 1 shows, for each category, how the precision of the algorithm behaves as we increase the number of states of the HMM and the length of the observation sequences. For almost all categories, the precision increases as the length of observation sequence increases. This is normal because long sequences provide more data and consequently allow to have more accurate HMM models. As of the number of states of the HMM, Figure 1 shows that every category has an optimal number of states. For instance, Trusted and Malicious categories clearly need HMMs with small number of states (one or two). Therefore, HMMs with larger number of states will need

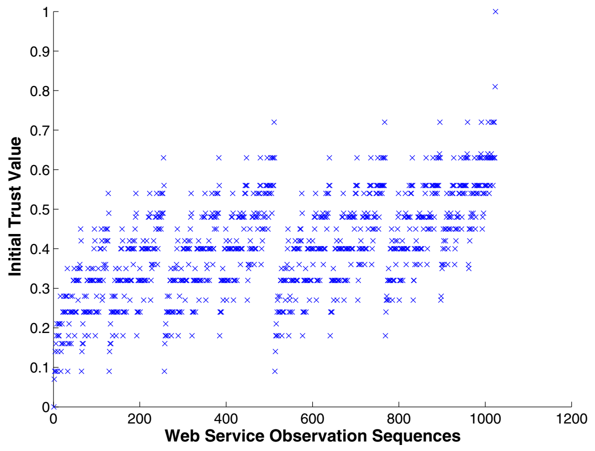


Fig. 3. Distribution of Trust Values

lengthier observation sequences to reach a precision of 100% as shown in Figure 1. For the Betraying and Redemptive categories, the optimal number of states is not very obvious. The plots suggest that the precision reaches its maximum with HMMs having four or eight states. For the Oscillating category, the plot shows clearly that the number of states of the corresponding HMM should be strictly larger than one. Overall, Figure 1 shows that for all categories except Redemptive, Algorithm 1 reaches a significantly high level of precision. For Redemptive category, this can be explained by the fact that it has a significant intersection with the Oscillating category. This is confirmed by the recall values of Oscillating category in Figure 2.

Figure 2 illustrates the recall rates for each category as we increase the number of states of the HMM and the length of observation sequences. Similarly to precision, recall increases as the observation sequences get longer. The only exception is the Oscillating category which is due to the tight intersection it has with the Redemptive category. Overall, it is easy to see that, for a particular combination of number of states in the HMM and observation sequence length, all categories reach an almost perfect recall rate of 100%.

5.2 Distribution of Trust Values

As shown earlier, the weight given to each classified trust observations sequence allows to discriminate between Web services either having the same pattern or in different patterns. In our experiments, we use the following weights: 1 for the trusted pattern, 0.6 for the malicious pattern, 0.7 for the betraying pattern, 0.8 for the oscillating pattern, and 0.9 for the redemptive pattern.

Figure 3 depicts the distribution of trust values for the classified Web services. It is worth to mention that those Web services having the same pattern are close (but still different thanks to the maliciousness rate) to each others and together they form a kind of a cluster.

6 Conclusion

We proposed in this work a new bootstrapping technique for Web services based on trust patterns and hidden markov models. The main benefits of our technique compared to the related work consist in a fair assignment of trust values to unknown Web services and a global view of possible behaviors of these services. The experimental analysis carried out showed that our HMM based classification technique is very promising and features a high rate of precision and completeness provided that the number of states of HMMs are well chosen and the observation sequences used for learning are long enough. The only concern, however, remains the difficulty of the algorithm to cope with the intersection between categories, in particular, the redemptive and oscillating categories. Another interesting research initiative is to apply our technique on real world Web services. These research directions are part of our future work.

References

1. Baum, L.E., Egon, J.A.: An Inequality with Applications to Statistical Estimation for Probabilitsic Functions of a Markov Process and to a Model of Ecology. Bull. Amer. Meteorol. Soc. 73, 360–363 (1967)
2. Moukas, A., Zacharia, G., Maes, P.: Collaborative Reputation Mechanisms in Electronic Marketplaces. Decision Support Systems 29(4), 371–388 (2000)
3. Grandison, T., Sloman, M.: A survey of trust in internet applications. IEEE Communication Surveys & Tutorials 4(4), 2–16 (2000)
4. Malik, Z.: Reputation-based Trust Framework for Service Oriented Environments. PhD thesis, Virginia Polytechnic Institute and State University (October 2008)
5. Malik, Z., Bouguettaya, A.: RATEWeb: Reputation Assessment for Trust Establishment among Web services. Very Large Data Bases (VLDB) 18(4), 885–911 (2009)
6. Malik, Z., Bouguettaya, A.: Reputation Bootstrapping for Trust Establishment among Web Services. IEEE Internet Computing 13(1), 40–47 (2009)
7. Marti, S., Garcia-Molina, H.: Taxonomy of Trust: Categorizing P2P Reputation Systems. Computer Networks 50(4), 472–484 (2006)
8. Maximilien, E., Singh, M.: Reputation and Endorsement for Web Services. SIGecom Exchanges 3(1), 24–31 (2002)
9. Maximilien, E., Singh, M.: Toward Autonomic Web Services Trust and Selection. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, NY, USA, pp. 212–221 (2004)
10. Murphy, K.: Hidden Markov Model (HMM) Toolbox for Matlab (2005), `http://www.cs.ubc.ca/murphyk/Software/HMM/hmm.html`
11. Rabiner, L.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE 77(2), 257–286 (2000)
12. Doshi, P., Paradesi, S., Swaika, S.: Integrating Behavioral Trust in Web Service Compositions. In: Proceedings of the Seventh International Conference on Web Services (ICWS 2009), Los Angeles, CA, USA, pp. 453–460 (2009)
13. OASIS Web Service Secure Exchange TC. Ws-trust 1.3 (2007), `http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html`
14. Yahyaoui, H.: Trust Assessment for Web Services. In: IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, pp. 315–320 (2010)