Journal of Universal Computer Science, vol. 18, no. 14 (2012), 1950-1966
submitted: 10/1/12, accepted: 25/7/12, appeared: 28/7/12 © *J.UCS*

Behavioral and Temporal Pattern Detection within Financial Data with Hidden Information

Doron Drusinsky
(Naval Postgraduate School, Monterey, CA, USA, and Time Rover, Inc., USA
ddrusins@nps.edu)

Abstract: This paper describes a technique for behavioral and temporal pattern detection within financial data, such as credit card and bank account data, where the required information is only partially visible.

Typically, transaction amount, transaction date, merchant name and type, and location of transaction are all visible data items, i.e., they are readily available in the financial institutions database. In contrast, the transaction status as a business transaction (using a personal card), a personal transaction, an investment related transaction, or perhaps a suspicious transaction, is information not explicitly available in the database. Our behavioral pattern detection technique combines well-known Hidden Markov Model (HMM) techniques for learning and subsequent identification of hidden artifacts, with run-time pattern detection of probabilistic UML-based formal specifications. The proposed approach entails a process in which the end-user first develops his or her deterministic patterns, s/he then identifies hidden artifacts in those patterns. Those artifacts induce the state set of the identifying HMM, whose remaining parameters are learned using standard frequency analysis techniques. In the run-time pattern detection phase, the system emits visible information, used by the HMM to deduce invisible information, and sequences thereof; both types of information are then used by a probabilistic pattern detector to monitor the pattern.

Keywords: Hidden Markov Models, UML, statecharts, monitoring, patterns, hidden data
Categories: F.4.1, F.1.1, D.2.4

1 Introduction

A Hidden Markov Model (HMM) can be considered a state machine in which state transitions and state outputs, or observations, are probabilistic. HMM's are used to learn and classify sequences of observables. HMM technology has been used successfully in a diverse set of applications, such as speech recognition [Davies 1952], [Pierce 1969], Gene prediction [Rabiner 1989], and Cryptanalysis [Singh 2000].

Because of the probabilistic nature of the underlying process being observed by HMM's, they are not used often to recognize long-periodic sequences. Rather, they are mostly used as discriminators, to determine whether one HMM is better than another. For example, an HMM-based speech recognition system may have each HMM represent a word, with run time voice recognition choosing the HMM that best fits the incoming sequence of speech features. This is in contrast with Deterministic Finite Automata (DFA) [Hopcroft, Motwani and Ullman (2006)], Finite State Machines (FSM's) [Kohavi and Jha (2009)], or Harel-Statecharts [Drusinsky (2006)],

[Drusinsky (2011)], [Harel 1987], which are often used to identify and clindividual sequences. Stated differently, because HMM's identify individual sequences of external observables with a relatively low probability, it is usually not perceived as convincing evidence of the occurrence of a particular sequence.

Run-time Verification (RV) of formal specification assertions is a class of methods for monitoring the sequencing and temporal behavior of an underlying application and comparing it to the correct behavior as specified by a formal specification pattern. Some published RV tools and techniques are: the TemporalRover and DBRover [Drusinsky 2000], PaX [Haveland and Rosu 2004] and RT-Mac [Sammapun, Lee and Sokolsky 2005], all of which use extensions and variants of Propositional Linear-time Temporal Logic (PLTL) as the specification language of choice, and the StateRover [StateRover] that uses deterministic and non-deterministic statechart diagrams as its specification language. In [Drusinsky 2011], Drusinsky describes the application of RV using statechart assertions to the verification of DoD and NASA applications, and to those of the Brazilian Space agency.

In this paper, we use HMM's to identify hidden events and sequences thereof. However, we will not be using the (rather small) probability of an observable sequence, but rather the probability of a hidden state being reached *given* a sequence of observables. Hence, the technique identifies hidden events with a relatively high probability.

This paper describes a pattern detection technique suitable for financial systems in which not all artifacts are necessarily observable. The technique is a novel combination of Hidden Markov Models (HMM's) with RV techniques for probabilistic pattern matching of statechart patterns. Throughout the paper, we will be using the Statechart assertion formal specification language of [Drusinsky (2006)] and [Drusinsky (2011)]. We will show a probabilistic variant of this formalism suitable for pattern detection within systems with hidden inputs.

The technique in this paper is not positioned as a method for achieving financial gains in financial markets. Various papers investigating and analyzing such statistical techniques can be found in the literature, using artifacts such as long-term memory [Muchnik, Bunde and Havlin 2009], self similarity [Ghosh, Manimaran and Panigrahi 2011], and fat-tailed distributions [So et al. 2008]; in fact, power laws are used to classify time series sequences in many other fields besides financial systems [Li and Zhao 2012], [Li 2010], [Li, Cattani, and Chen 2011], [Todorova and Vogt 2011].

Rather, the technique suggested in this paper is positioned as a hybrid pattern detection technique that combines patterns written by humans with statistical observations – manifested as HMM's. In other words, it is positioned as a hybrid between formal specification and run-time verification techniques (e.g., [Drusinsky (2006)], [Drusinsky (2011)], [Drusinsky 2000]) and statistical pattern detection. Section 8 addresses the possibility of extending our approach to utilize some of the above mentioned statistical techniques such as long-term memory, fat tailed distributions, and various other fractal properties.

The rest of the paper is organized as follows. Section 2 provides an overview of behavioral pattern detection using deterministic UML statechart patterns. Section 3 provides an overview of HMM's and HMM related algorithms. Section 4 describes our proposed pattern detection architecture and process that uses a combination of

hidden and visible data, using an HMM connected to a behavioral pattern detection monitor. Section 4 also provides a workflow diagram that associates all remaining sections with tasks a developer will perform when applying the suggested technique. Section 5 describes HMM parameter estimation for the financial data HMM component, and section 6 describes the operation of the pattern detector. Section 7 describes the operation of the probabilistic pattern-matching monitor, and section 8 describes three techniques for computing the probability distribution used by that monitor.

2 Behavioral Pattern Detection using Deterministic UML Statechart Patterns – an Overview

Consider the following natural language (NL) patterns for a credit card (CC) system; the NL pattern is specified as being *flagged* when a scenario conforms to the pattern:

R1. *Flag a customer whose average expense, over three consecutive non-holiday weekend clothing related transactions is of a Dollar amount greater than his or her* $\mu+\sigma$*, where* μ *and* σ *are respectively, the mean and standard deviation of the customer's clothing expenses during the previous year*.

[Fig. 1] depicts a statechart-pattern for R1. As described in [Davies, Biddulph and Balashek 1952] and [Drusinsky (2006)], a statechart- pattern is a state-machine augmented with hierarchy, flowcharting capabilities, a Java action language, and a built in Boolean flag named *bFlag* whose default value is *false*, with a *true* value indicating that the pattern has been flagged (e.g., per pattern R1, flags that the input scenario conforms to R1).

The statechart-pattern of [Fig. 1] combines flowchart and state-machine elements.

Rectangular boxes and decision diamonds are flowchart elements – the statechart flows through them while executing their actions and conditions, eventually resting on a state machine state like *WaitForTransaction* where the statechart waits for an event. Hence, the statechart flows through the *Init* flowchart box, executes its actions and waits in the *WaitForTransaction* state. When a user transaction occurs (*newTransaction* event, with two arguments, a Transaction object and a User object) the statechart checks whether the argument is a holiday and clothing related transaction. If not, then the statechart waits for the next transaction. If it is, then the statechart checks whether the transaction is a weekend transaction. If it is, the statechart calculates the average amount spent on the most recent three such transactions this weekend (see the *CalculateAverage* flowchart box). If this average exceeds the user's $\mu+\sigma$ then the pattern detection flag is raised (*bFlag* = true).

Pattern matching is performed by comparing a trace of the financial system (e.g., a CC statement or bank log) to the behavior of the pattern set. The StateRover tool does so using a two step process. First, a transaction log, or statement, is converted into an equivalent JUnit test [JUnit], and the pattern is code-generated into an equivalent Java class (details about this code generator are available in [Drusinsky (2006)]). Next comes an RV step where the JUnit test is executed, thus checking that the transaction log conforms to the pattern. (Note that we assume that for an instance of the R1 pattern – i.e., an instance object of the Java class generated for statechart-pattern of [Fig.1], exists per user.)

The extended pattern matching technique suggested in this paper uses the same process for the development of patterns, i.e., patterns are developed as deterministic patterns. However, rather than performing deterministic RV by the virtue of using a code generator that generates a deterministic pattern implementation, our technique performs probabilistic pattern detection using a special pattern code generator that generates a probabilistic, weighted implementation. Specific details are provided in section 6.

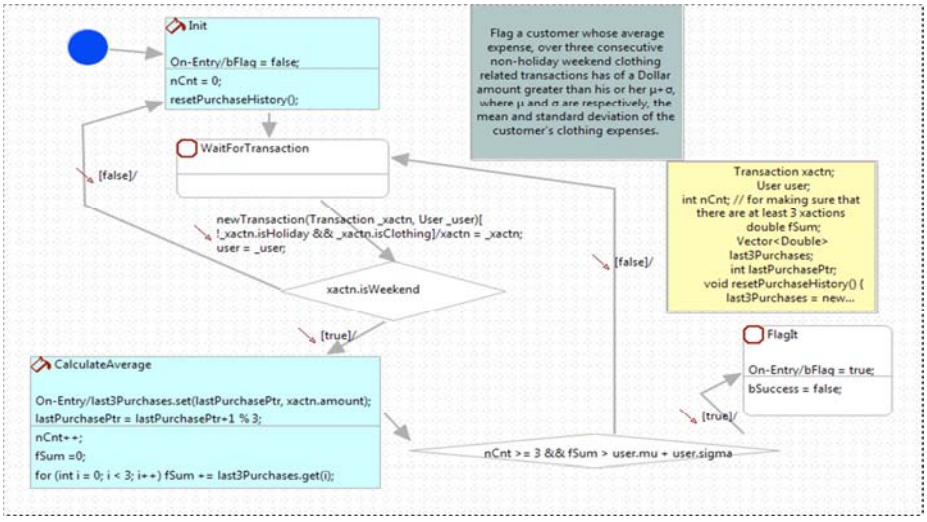

Figure 1: A statechart-pattern for requirement R1

3 Hidden Markov Models

A (discrete) hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved, or hidden states. While in a regular Markov model the state is directly visible to the observer, in a hidden Markov model the state is not directly visible, while the output, dependent on the state, is visible.

The parameters of a simple HMM are [Rabiner 1989]:

- N, the number of states in the model. Individual states are denoted $S = \{s_1, s_2,...s_N\}$, and the state at time t as q_t.
- M, the number of distinct observation symbols. Individual observations are denoted $V = \{v_1, v_2,...v_M\}$.
- The state transition probability distribution $A = \{a_{ij}\}$ where $a_{ij} = P[q_{t+1} = s_j | q_t = s_i]$, $1 \le i,j \le N$. Clearly, $\forall i, 1 \le i \le N, \sum_{1 \le j \le N} a_{ij} = 1$.
- The observation symbol probability distribution in state j, $B=\{b_j(k)\}$, where $b_j(k) = P[v_k \text{ at } t \mid q_t = s_j]$, $1 \le j \le N$, $1 \le k \le M$.

- The initial state distribution $\pi = \{\pi_i\}$, where $\pi_i = P[q_1 = s_i]$, $1 \le i \le N$.

Rabiner [Rabiner 1989] describes the following three primary problems associated with HMM's:

1. Given the observation sequence $O = O_1O_2...O_T$, and an HMM model $\lambda = (A,B, \pi)$, how do we efficiently compute $P(O|\lambda)$?
2. Given the observation sequence $O = O_1O_2...O_T$, and an HMM model $\lambda = (A,B, \pi)$, how do we choose an optimal state sequence $Q = q_1 q_2...q_T$?
3. How do we calculate the model parameters $\lambda = (A,B, \pi)$ to maximize $P(O|\lambda)$?

The most well known algorithms used to solve these problems are:

1. The *forward algorithm*, for calculating the *forward variable* $\alpha_t(i) = P(O_1O_2...O_t, q_t = s_i \mid \lambda)$. The forward algorithm is a dynamic programming algorithm based on the recurrence:

   $\alpha_{t+1}(j) = [\sum_{i=1..N} \alpha_t(i)\, a_{ij}\,]\, b_j(O_{t+1})$, $1 \le t \le T\text{-}1$, $1 \le j \le N$,

   with the initialization:

   $\alpha_1(j) = \pi_j\, b_j(O_1)$.

   Note that $P(O_1O_2...O_t|\lambda) = \sum_{i=1..N} \alpha_t(i)$.

   $\alpha`$ is the normalized version of α:

   $\alpha`_t(j) = P(q_t = s_i | O_1O_2...O_t, \lambda)$.

2. The *backward algorithm*, for calculating the *backward variable* $\beta_t(i) = P(O_{t+1}O_{t+2}...O_T \mid q_t = s_i, \lambda)$. The algorithm is a dynamic programming algorithm based on the recurrence:

   $\beta_t(i) = \sum_{j=1..N} a_{ij}\; b_j(O_{t+1})\, \beta_{t+1}(j)$, for $t = T\text{-}1, T\text{-}2, ..., 1$, and $1 \le i \le N$,

   with the initialization:

   $\beta_T(i) = 1$, for $1 \le i \le N$.

3. The *forward-backward algorithm*, for calculating the *forward-backward variable*

   $\gamma_t(i) = P(q_t = s_i \mid O_1...O_T, \lambda)$.

   γ is also:

   $\gamma_t(i) = (\alpha_t(i)\;\; \beta_t(i)) / P(O_1O_2...O_T|\lambda)$

4. The Viterbi algorithm, for calculating the best state sequence that explains an observation sequence, $\delta_T(O_1O_2...O_T \mid \lambda)$. The algorithm defines:

   $\delta_t(i) = \max[q_1, q_2, ... q_{t-1}]\, P(q_1, q_2, ... q_t = s_i, O_1O_2...O_t \mid \lambda)$,

   and uses the following recursive formula:

   $\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ij}]\, b_j(O_t)$

   along with the following formula, used to recover the actual most probable state sequence:

   $\psi_t(j) = \text{argmax}_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ij}]$, where $\psi_1(j) = 0$;

   The Viterbi algorithm is essentially the forward algorithm with a recurrence in which a *max* operator is used instead of the sum. The probability of best state sequence $\delta_T(O_1O_2...O_T \mid \lambda)$ is then the maximal $\delta_T(i)$, $1 \le i \le N$, and $q_T = \text{argmax}_i\, \delta_T(i)$, $1 \le i \le N$.

The most probable state sequence $q_1,q_2,...q_T$ is calculated in a backward manner, using $q_{t-1} = \psi_t(q_t)$.

3 The Suggested Technique: Detecting Hidden Data Patterns

The suggested pattern detection technique combines two otherwise separate techniques, namely: (1) deterministic behavioral pattern detection techniques for the detection of behavioral patterns written by human experts, and (2) well known HMM-based statistical techniques. It does so using a loosely coupled approach, where two respective components have little dependence, thereby easing their respective development and maintenance tasks.

[Fig. 2] describes the workflow for a developer developing such a hybrid pattern detection system. It consists of the following primary steps:

Box No. 1: In which the pattern is formally (and deterministically) specified, as described in Section 2.

Box No. 2: In which validation tests are written to assure the pattern accurately represents the natural language and cognitive requirements, as detailed in [Drusinsky Otani and Shing 2008].

Box No. 3: In which the validation testing is executed, as as described in Section 2. If a test fails then the developer either adjusts the pattern or the test and repeats the process.

Box No. 4: In which the developers analyzes the UML-statechart patterns developed in Box No. 1 to identify those events and conditions whose data domain is not visible. This activity is described in Section 4.

Box No. 5: In which the HMM state set is identified from the information gathered in Box No 4. This activity is described in Section 4.

Box No. 6: In which, HMM parameters are calculated based on the abovementioned state set. This activity is described in Section 5.

Box No. 7: In which the pattern is automatically implemented using a code generator. Rather than using the deterministic-pattern code generator mentioned in section 2, we use a special code generator that generates *weighted state changes* instead of deterministic state changes. This activity is described in Sections 6 and 7.

Box No. 8: In which the abovementioned implementation code is executed thereby performing probabilistic pattern matching. This activity is described in Sections 6 and 7.

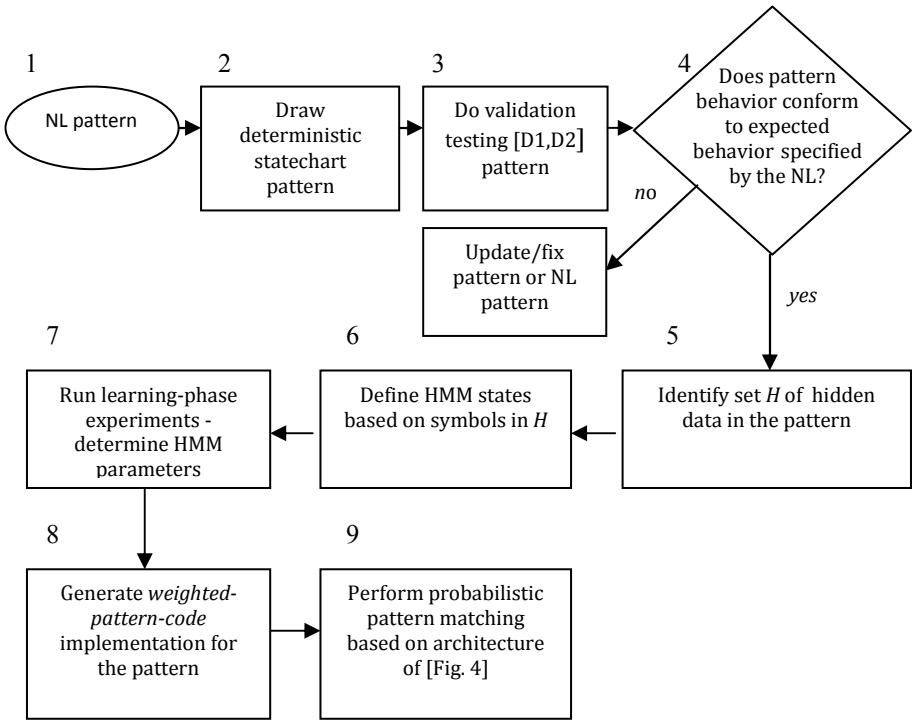

Figure 2: Workflow for developing pattern matching with hidden information

Suppose our financial system would like to detect patterns that assert about data that is not explicitly present in the list of transactions, such as whether a transaction is business related (albeit using a personal CC or bank account), is personal, is suspect (as fraudulent), or is investment related. More specifically, consider the NL for pattern R2.

R2. *Flag a customer that for a period of a week with at least two investment transactions, customer's investment transaction Dollar amount is 20% higher than customer's personal transaction Dollar amount.*

[Fig. 3] depicts a statechart-pattern for requirement R2. Note that it asserts about visible information (e.g., *newTransaction* event and transaction *amount* data item) as well as hidden information (*hmmType*, being PERSONAL or INVESTMENT).

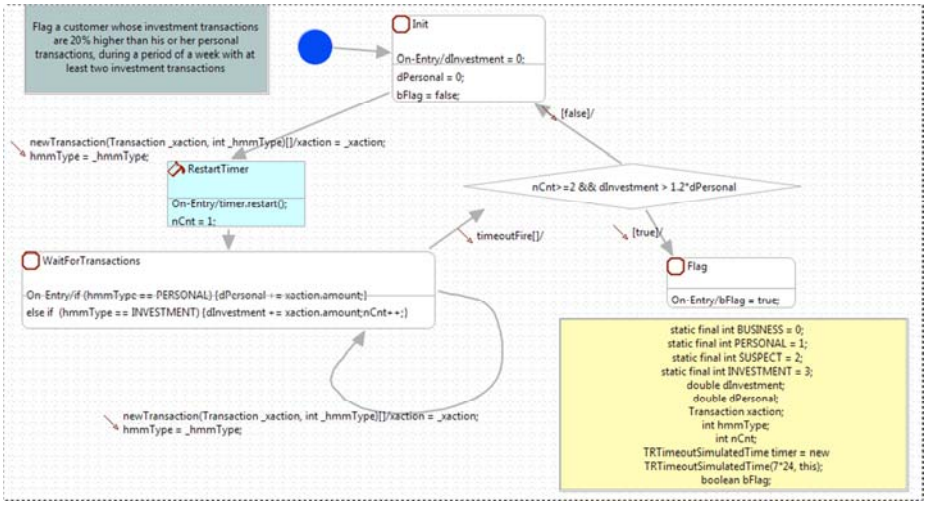

Figure 3: A statechart-pattern for requirement R2

To enable pattern detection of a transaction log with respect to R2 and its corresponding statechart-pattern, we apply the pattern detection architecture of [Fig. 4]. Key components of the architecture are:

1. It contains a Hidden Markov Model (HMM), used to decode the probability of occurrence of sequences of hidden states given sequences of the observable transactions. The states of the HMM are: *Business*, *Personal*, *Suspect*, and *Investment*, referring to the four *transaction types* discussed earlier. This HMM provides a plurality of weighted transaction type inputs to the statechart-pattern, weights reflecting the probability the corresponding HMM state was reached given the observed transaction sequence.
2. It uses a special code generator that generates a probabilistic implementation for the statechart-pattern, one that operates on the weighted inputs from the HMM.
3. It evaluates the pattern using a success score in the range [0,1].

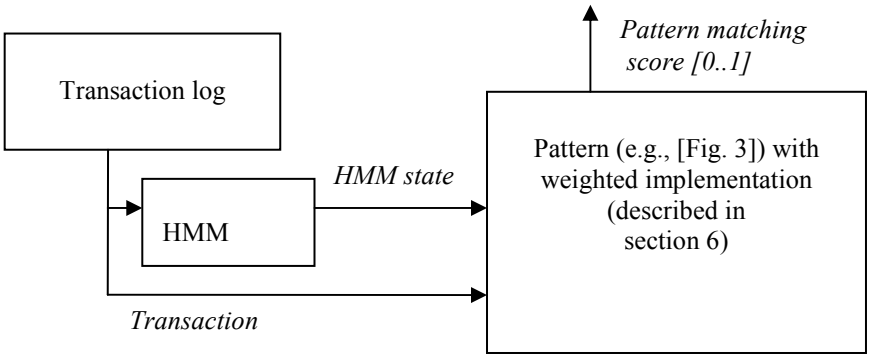

Figure 4: The pattern matching architecture for the transaction log and requirement R2

The HMM parameters for this example are determined in the learning-phase discussed in section 5. They are:

- The state set *Q* consists of the four states mentioned above: *Business*, *Personal*, *Suspect*, and *Investment*, denoted as states 0, 1, 2, and 3, respectively.
- An observable O, which is a triplet describing the data combinations required by the HMM to determine the next state. For example the Boolean conjunction:
- *isHoliday ^ isAutomotive ^ !isYouthExpense,* means the (visible) transaction occurred during a holiday, is not automotive related, and is related to an expense a young person typically makes. We represent each observable as a tuple of integers such as<2,6,0>, where each integer component represents a condition.
- Section 5 describes in greater detail the set of observable triplets for this example and the their associated learning process.
- State transition probabilities, given by the matrix *A* in [Tab. 1].

Transition Source\Target	Buss.	Pers.	Susp.	Inv.
Buss.	0.35	0.46	0.05	0.14
Pers.	0.25	0.60	0.06	0.09
Suspect	0.37	0.39	0.23	0.01
Inv.	0.35	0.38	0.05	0.22

Table 1: Matrix A of HMM state transition probabilities

- Matrix *B*, containing $b_s(O)$, the probability of an observable *O* being observed in state *s*, part of which is presented in [Tab. 2].

O\state	Buss.	Pers.	Suspect	Inv.
<0,1,0>	0.01	0.11	0.01	0.03
<0,2,1>	0.03	0.15	0.01	0.0001
<1,0,0>	0.01	0.22	0.03	0.01

Table 2: A part of Matrix B, of probability of observation O in HMM states

- The initial state distribution is [0.2, 0.65, 0.05, 0.1] for *Business*, *Personal*, *Suspect*, and *Investment*, respectively.

Pattern-detection now proceeds according to the process illustrated in [Fig. 4], as follows.

Transactions from the transaction-log are fed into the HMM, which then executes a probability estimation algorithm, such as the forward-algorithm, for the current iteration (section 7 discusses three probability estimation techniques). These probability values represent probabilities of the HMM being in states *Business*, *Personal*, *Suspect*, or *Investment*. This vector of symbols and corresponding probabilities is passed to the pattern's implementation code, which executes a weighted version of a state-machine state change, detailed in section 6. Finally, as discussed in section 6, the pattern detector announces the probability it flagged a pattern match.

4 From Patterns to HMM Parameter Estimation

HMM parameter estimation, i.e., estimating the transition probability and probability of state observations, is considered a difficult problem. In particular, it is difficult to estimate the number of HMM states, the extreme cases being using one state (i.e., reducing the HMM to a stationary process) or *n* states, *n* being the length of the observation sequence.

In our case however, HMM states are known; they are directly related to the hidden pattern artifacts. In our example, the four hidden symbols *Business*, *Personal*,

Suspect, and *Investment*, are derived from [Fig. 3] and its pattern specification R2, as well as from other patterns.

Our use-case for HMM's induces a simple method for calculating transition and observable probabilities. Because HMM states relate to real world artifacts, we can conduct learning-phase experiments, which measure relative frequencies using standard frequency analysis. The financial industry performs such experiments as a matter of business [Sesera 2008]. [Tab. 3] illustrates the learning-phase process with a list of transactions taken from the authors CC statement; the author annotated the transactions with the corresponding HMM state, listed in the right-most column.

	Date	Merchant	Amt	
1	9/22/11	WHOLEFDS	57.25	P
2	9/22/11	TRADER JOE'S	113.35	P
3	9/23/11	IKEA	975.86	P
4	9/23/11	IKEA	68	B
5	9/23/11	UNION 76	38.46	P
6	9/25/11	M. LI DDS	38.8	P
7	9/30/11	UNION 76	25.6	P
8	10/5/11	DOG EAR PUBLISHING	175	B
9	10/6/11	UNITED AIR	1666.8	P
10	10/7/11	MCAFEE	45.99	B
11	10/8/11	GREAT TANS	49.25	S
12	10/9/11	CVS PHARMACY	30.27	P
13	10/11/11	K APT HOME OWNERS ASSOC	303	I
14	10/13/11	RADIOSHACK	13.04	B

Table 3: A sample segment of the author's transaction log. The rightmost column indicates the HMM state as added by the author in the learning phase

With this information the process continues by identifying the combination of visible transaction-log data, in the form of Boolean conditions, that according to the training user induces the states in the rows of [Tab. 3]. [Tab. 4] presents this information.

	Condition	Condition	Condition	
1	Food			P
2	Food			P
3	Furniture	Weekend		P
4	Furniture	!Weekend	!Holiday	B
5	Automotive			P
6	Health			P
7	Automotive			P
8	Publishing			B
9	Travel	Amt>1000		P
10	ITSecurity	Amt > 40		B
11	Leisure	Youth Expense		S
12	Health			P
13	Residential	Non Local		I
14	Electronics	Weekday		B

Table 4: Conditions that induce the HMM states in [Tab. 3]

The conditions of [Tab. 4] represent data items required by the HMM to determine the next state. In our example these items are: *isWeekend*, *isHoliday, isFurniture, isPublishing, isResidential, isElectronics, isHealth, isAutomotive, isLeisure, isYouthExpense, isITSecurity,* and *Amt* (Dollar amount). All data items with the prefix *is* are Boolean conditions. According to [Tab. 4], the *Amt* data can be divided into 3 mutually exclusive segments: less than $40, between $40 and $1000, and above $1000, denoted as $Amt_{<40}$, $Amt_{[40,1000]}$, and $Amt_{>1000,}$ respectively.

Note that the number of combinations of these data items is large: $3\times2^{11}=6144$. However, most conditions are not necessarily orthogonal, but are often mutually exclusive. We identify three bins of mutually exclusive conditions:

1. Temporal – containing *isWeekend*, and *isHoliday* (when a certain day is both we say its *isHoliday*).
2. Type of purchased object – containing *isFurniture, isPublishing, isResidential, isElectronics, isHealth, isAutomotive, isLeisure*, and *isITSecurity*.
3. Age group for purchased object – containing *isYouthExpense*.

Using these three bins we encode observables as triplets, such as: <2,6,0> being:

isHoliday ^ isAutomotive ^ !isYouthExpense.

HMM parameters follow from this information in a straight-forward manner. For example, the probability of a transition from state *Personal* to state *Business* is the ratio of number transactions with a *P* state whose next transaction is *B* state to the total number of transactions with a *P* state, being 0.375 for the data in Tables 3 and 4. Similarly, the probability of observable <2,6,0> in state *Personal* is the ratio of number transactions with a *P* state and observable <2,6,0> to the total number of transactions with a *P* state, being 0.25 for the data in Tables 3 and 4.

5 Behavioral Pattern Matching in the Presence of Hidden Data

Using the architecture of [Fig. 4], the pattern-matching module observes sequences that consist of visible as well as hidden artifacts; in [Fig. 3] for example, *newTransaction* is a visible event, while *hmmType* is hidden. Hidden artifacts have an associated probability distribution which we call the *probability-of-occurrence distribution (POD)*, such as *POD*-1: *hmmType=BUSINESS, PERSONAL, SUSPECT or INVESTMENT at time 5 occurs with probability 0.1, 0.8, 0.05, 0.05, respectively*. Section 7 describes three techniques, called α`, γ, and δ``, for computing the cycle-by-cycle POD, based on α, γ, and δ, respectively. We consider a visible artifact to have a probability of occurrence of 1.

A weighted/probabilistic implementation of the statechart-pattern module of [Fig. 4] responds to an input sequence $I = <S_1, P_1>, <S_2, P_2>,.., <S_T, P_T>$, where S_t is a visible or hidden artifact (i.e., event such a *newTransaction*, or data artifact, i.e., variable, such as *hmmType,* both in [Fig. 3]), and P_t is the *POD* of S_t.

We use the UML notation for S_t, S_t=$event_t$[$condition_t$], where $condition_t$ is optional; $event_t$ and $condition_t$ can either or both be visible or hidden.

A pattern implementation consists of a collection ***C*** of instances, or copies, of the pattern, called *configurations.* Each configuration executes as a standalone pattern and preserves its own present-state. Each configuration *Con* has a probability measure *P*(*Con*), called the *Configuration Probability Measure* (CPM), that measures the probability the pattern is behaving as suggested by *Con*, i.e., that its present-state is *Con*'s present state. Upon startup, ***C*** consists of a single configuration $Con_{default}$ whose present-state, denoted $PS(Con_{default})$, is the pattern's default state (e.g., state *Init* in [Fig. 3]), and having $P(Con_{default})=1$.

All configurations of ***C*** respond to a pair $<S_t,P_t>$ of *I*, as follows. If $P_t = 1$ then the configuration performs a conventional state machine state change upon input S_i. Otherwise, either $event_t$ or $condition_t$ are hidden. In this case the configuration *Con* is replaced with two configurations: *Con1* and *Con2,* whose present-state probabilities are calculated as follows:

- If $event_t$ is hidden (as discussed in section 7) then $P(Con1) = P(Con)*P_t$ and $P(Con2) = P(Con)*(1-P_t)$. The calculation of the probability of hidden events is described in a companion paper
- If $condition_t$ is hidden, then we calculate $P(condition_t)$, the probability of the condition, as a function of the probabilities of its constituent variables using standard probability calculations. For example, if $condition_t$ is *hmmState = BUSINESS || hmmState = INVESTMENT* then $P(condition_t) = P($*hmmState =*

BUSINESS) + *P(hmmState = INVESTMENT*), where each term is taken from the POD at time *t*, such as 0.1 and 0.05 respectively, using *POD*-1.

- We set $P(Con1)=P(Con)P(condition_t)$, and $P(Con2)=P(Con)(1-P(condition_t))$.

Let *PS*(*Con*) denote *Con*'s present-state. *PS*(*Con1*) and *PS*(*Con2*) are determined as follows:

- If $event_t$ is hidden then *PS*(*Con1*) is the next state determined by the pattern's transition out of *PS*(*Con*), under the assumption that the event fired, and *PS*(*Con2*)= *PS*(*Con*).
- If $condition_t$ is hidden (e.g., *hmmState*==PERSONAL condition in [Fig. 3]), then *PS*(*Con1*) is calculated assuming $condition_t=true$ and *PS*(*Con2*) is calculated assuming $condition_t=false$,

For the sake of simplicity we disallow patterns in which both $event_t$ and $condition_t$ are hidden.

C configurations are routinely (i.e., every cycle *t*) managed as follows. All configurations *Con`* with the same present-state are merged into a single configuration $Con_{merged}$, using the sum of all *P*(*Con`*) as $P(Con_{merged})$. (More accurately, *PS*(*Con*) is an extended state vector, that includes the state variable and the states of all local variables, such as the timer state and the *bFlag* flag.)

The statechart-pattern declares a *probability of flag (POF)*, i.e., the probability its corresponding NL requirement has been flagged, on a cycle by cycle basis, being the sum of all *P*(*Con*) for all configurations *Con* such that *PS*(*Con*) is an flag state.

Note that statechart-patterns typically use sink-states as flag states, sink-states being states with no outgoing transitions. For such patterns, the POF is monotonically increasing with time.

6 Calculating the POD of a Hidden Artifact

We propose three techniques for estimating the POD at time *t*: the *alpha*, *gamma*, and *delta* methods, as follows.

- The *alpha* method, which uses *N* values of $\alpha`_t(i)=P(q_t=s_i|O_1O_2...O_t, \lambda)$, one per symbol $s_i$, $1\le i\le N$. Note that $\sum_{1\le i\le N}\alpha`_t(i) = 1$.
- The *gamma* method, which uses *N* values of $\gamma_t(i)=P(q_t=s_i|O_1O_2...O_T, \lambda)$, one per symbol s_i, $1\le i\le N$. Note that $\sum_{1\le i\le N}\gamma_t(i) = 1$.
- The *delta* method, which uses *N* values of:
  $\delta_t``(i) = \delta_t`(i)/\sum_{1\le i\le N}\delta_t`(i)$, where
  $\delta_t`(i) = \max[q_1,q_2,... q_{t-1}]P(q_1,q_2,...q_t=s_i| O_1O_2...O_t,\lambda)$, where
  $P(q_1,q_2,...q_t=s_i| O_1O_2...O_t,\lambda)=\delta_t(i)/P(O_1O_2...O_t)$. In other words, $\delta_t``(i)$ is a normalized version of $\delta_t`(i)$, which in turn is the probability of the HMM generating symbol s_i at time *t*, via the most probable state sequence, given the observation.

The gamma method is a backward-forward algorithm; it therefore requires the entire observable sequence $O_1O_2...O_T$ for the evaluation of $\gamma_t(i)$ for $t \le T$. The alpha

and delta methods on the other hand, are forward algorithms and therefore do not require future-time information.

When the HMM contains transitions with probability 0, then all three methods might induce sequences of symbols that cannot be physically generated. For example, consider an an HMM with N=3 and $a_{1,2}$=0, and suppose $\gamma_t(1)$=0.3 and $\gamma_{t+1}(2)$=0.2; The pattern then considers the sequence s_1, s_2 as possible, having a positive probability of 0.06.

7 Conclusion and Future Research

We have demonstrated a technique for performing financial pattern detection in the presence of hidden financial data.

In future research we plan to incorporate known statistical methods within the suggested monitoring technique as part of either the visible data or the hidden data. On the visible data side, rather than using raw data, such as transaction data, it is possible to refer to statistical artifacts, such as maxima and minima obtained in successive time intervals of fixed length R, such as manifested NL for pattern R3.

R3. *Flag a sequence in which the maxima is persistently at least 50% greater than the minima for a period of a week or more.*

On the hidden side, rather than using a random variable with Categorical distribution it is possible to extend the technique to cater for random variables with various distributions, ranging from a Normal distribution to distributions defined by more complex power laws [Li and Zhao 2012], [Li 2010].

We also plan on applying this technique to automatic pattern detection within large volumes of cyber data, in an effort to identify malicious or dangerous behavioral patterns.

We are currently building a special StateRover code-generator that generates weighted/probabilistic implementation code for statechart patterns.

Acknowledgements

This research was funded by a grant from the U.S. Defense Threat Reduction Agency (DTRA).

References

[Davies, Biddulph and Balashek 1952] Davies, K.H. Biddulph, R. Balashek, S.: "Automatic Speech Recognition of Spoken Digits"; J. Acoust. Soc. Am. 24, 6 (1952), 637–642.

[Drusinsky (2006)] Drusinsky, D.: "Modeling and Verification Using UML Statecharts – A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking"; Elsevier (2006).

[Drusinsky (2011)] Drusinsky, D.: "Practical UML-based Specification, Validation, and Verification of Mission-critical Software"; DogEar Publishing (2011).

[Drusinsky 2000] Drusinsky, D.: "The Temporal Rover and the ATG Rover"; Proc. SPIN 2000 Workshop, LNCS 1885, Springer-Verlag, (2000), 323-329.

[Drusinsky Michael and Shing 2008] Drusinsky, D., Michael, J.B., Shing, M.: "Visual Tradeoff Space for Formal Verification and Validation Techniques"; IEEE Systems Journal, 2, 4 (2008), 513-519.

[Drusinsky Michael and Shing 2008a] Drusinsky, D., Michael, J.B., Otani, T., Shing, M: "Validating UML Statechart-Based Assertions Libraries for Improved Reliability and Assurance"; Proc. of the Second International Conference on Secure System Integration and Reliability Improvement *(SSIRI)*,) (2008), 47-51. Best paper award.

[Harel 1987] Harel, D.: "Statecharts: A visual formalism for complex systems"; Science of Computer Programming, 8, 3 (1987), 231–274.

[Ghosh, Manimaran and Panigrahi 2011] Ghosh, S., Manimaran, P., Panigrahi, P.K.: "Characterizing multi-scale self-similar behavior and nonstatistical properties of fluctuations in financial time series"; Physica A, 390, 23-24 (2011), 4304-4316.

[Havelund and Rosu 2004] Havelund, K., Rosu, G.: "An Overview of the Runtime Verification Tool Java PathExplorer"; Formal Methods in System Design, 24 (2004), 189-215.

[Hopcroft, Motwani and Ullman (2006)] Hopcroft, J.E., Motwani, R., Ullman, J.D.: "Introduction to Automata Theory, Languages, and Computation*";* Addison Wesley (2006).

[JUnit] JUnit, http://www.junit.org

[Kohavi and Jha (2009)] Kohavi , Z., Jha, N.K.: "Switching and Finite Automata Theory"; Cambridge University Press (2009).

[Li and Zhao, 2012] Li, M., Zhao, W.: "Visiting power laws in cyber-physical networking systems"; Mathematical Problems in Engineering, 2012 (2012).

[Li 2010] Li, M.: "Fractal time series — a tutorial review"; Mathematical Problems in Engineering (2010), Article ID157264.

[Li, Cattani and Chen 2011] Li, M., Cattani, C., and Chen, S.Y.: "Viewing Sea Level by a One-dimensional Random Function with Long Memory"; Mathematical Problems in Engineering (2011).

[Muchnik, Bunde and Havlin 2009] Muchnik, L., Bunde, A., and Havlin, S.: Long term memory in extreme returns of financial time series"; PhysciaA , 388, 19 (2009), 4145-4150.

[Pierce, 1969] Pierce, J.: Whither Speech Recognition. *Journal of the Acoustical Society of America* (1969).

[Rabiner 1989] Rabiner, L.W.: "A Tutorial on Hidden Markov models and Selected Applications in Speach Recognition"; Proc. of the IEEE, **77**, 2 (1989).

[Rätsch et al., 2007] Rätsch, G., Sonnenburg, S, Srinivasan, J., Witte, H., Müller, K.R., Sommer, R.J., Schölkopf, B.: "Improving the C. elegans genome annotation using machine learning"; PLoS Computational Biology, 3, 2 (2007**)** e20. doi:10.1371/journal.pcbi.0030020. PMC 1808025. PMID 17319737.

[Sammapun, Lee and Sokolsky 2005] Sammapun, U., Lee, I., and Sokolsky, O.: "RT-MaC: Runtime Monitoring and Checking of Quantitative and Probabilistic Properties"; Proc. 11th IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications, IEEE, (2005), 147-153.

[StateRover] The StateRover, http://www.time-rover.com

[Singh 2000] Singh, S.: "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography"; London: Fourth Estate (2000), 143–189. ISBN 1-85702-879-1.

[Sesera 2008] Sesera, L.: "Fundamental Panking Patterns"; Proceedings of the 15th Conference on Pattern Languages of Programs (PLoP) (2008).

[So et al., 2008] So, M.K.P., Chen, C.W.S., Lee, J.-Y., Chang, N.D.Y.-P.: "An empirical evaluation of fat-tailed distributions in modeling financial time series"; Mathematics and Computers in Simulation, **77**, 1 (2008), 96-108.

[Todorova and Bogt, 2011] Todorova, L., and Vogt, B.: "Power law distribution in high frequency financial data? An econometric analysis"; Physica A, 390, 23-24 (2011), 4433-4444.