Unsupervised Clustering of Symbol Strings

John A. Flanagan
Nokia Research Center, PO Box 407, FIN-00045 NOKIA GROUP, Finland
E-mail : Adrian.Flanagan@nokia.com

***Abstract*— The Symbol String Clustering Map (SCM) is introduced as a very simple but effective algorithm for clustering strings of symbols in an unsupervised manner. The clustering is based on an iterative learning of the input data symbol strings. The learning uses the principle of Winner Take All (WTA) and hence requires a similarity measure between symbol strings. A novel and efficient, average based, similarity measure is defined. Unsupervised generation of the data cluster structure results from the use of a lateral inhibition function applied to the update of adjacent nodes on the SCM lattice. A simple coding method to convert time sequences of symbols to simple symbol strings for use in the SCM is described. The SCM is shown to generate clusters for symbol string data sets**

I. Introduction

Unlike a real number or vector representation a symbol or symbol string representation is an abstract means of representing information. Methods for processing real vector data for feature extraction and pattern recognition are varied and well known [1], however, classical feature extraction and pattern recognition algorithms when applied to symbol string representations of data are not always so straight forward. One of the main hurdles is that for feature extraction, or clustering algorithms a distance or similarity measure is required which in the case of symbol string data is not so easy. As a practical example of the clustering problem consider a simplified version of the shopping basket problem . If each item or item category in the shop is represented by a symbol $\psi_i \in \Psi$, where $\Psi = \{\psi_1, \ldots, \psi_N\}$ is the total collection of items or categories of items in the shop, then the shopping basket of each customer i can be represented by a symbol string,

$$\mathbf{S}_i = \left(s_1, s_2, \ldots, s_{n(i)}\right) \qquad s_j \in \Psi,\ j = 1, \ldots, n(i). \quad (1)$$

Note the length $n(i)$ of the symbol string can and does vary from customer to customer. The problem addressed here, in terms of the shopping basket problem is ; given a large sample of customer baskets, to identify different subgroups of customers, the customers in each subgroup have a higher probability of having the same items in their basket. This corresponds to a classic clustering problem applied to symbol strings.

There are two distinct approaches to the clustering problem. In the first approach it assumed that there is a set of data points $\mathbf{S}_1, \ldots, \mathbf{S}_M$ to be divided into K clusters. A typical criterion is that the data points belonging to the same cluster are more similar to each other than they are to data points belonging to other clusters. Typically algorithms to perform this type of clustering include, nearest neighbor [2], normalized cuts [3]. The second approach can be formally defined by considering the common probability space $(\Omega, \mathcal{F}, \pi)$, with $\omega \in \Omega$ a sample of Ω and in this case the sample of data points on which the clustering is to be based, $\omega = \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$. Associated with the random variables $\mathbf{S}_i$ is the probability density function p_Ω with state space $\mathcal{S}$. In this case the idea is to associate clusters with local peaks in p_Ω. This approach requires either a parametric or non parametric estimation of p_Ω and then some means of detecting the local maxima. One requirement for both approaches is to define some form of metric on $\mathcal{S}$.

In the case of the shopping basket example, one approach to defining a metric is to represent each customer basket by an N dimensional vector, with a 1 in position m if the customer purchased item ψ_m and a 0 otherwise. However the number N in any real situation is very large and hence the computation is intensive, despite the use of projection methods as described in [4]. A different approach is to define a distance measure between symbol strings, for example the Levenshtein distance measure [5] or Hamming distance measures [6]. Based on this type of measure it is possible to define a similarity measure between symbol strings and perform a batch clustering, as Kohonen and Somervuo [7], [8] based on the Self-Organizing Map (SOM) [9]. This corresponds to the first approach to clustering described above. In their approach the cluster centers are represented by data symbol strings that best represent the average of each cluster. Another approach taken by Kaski et al [10] is to generate an M dimensional histogram of the frequency of occurrence of M words in a each document of a large collection of documents and use the histogram as a real M dimensional vector representation. Using the histogram the Euclidean distance between documents can be defined. In this case each word corresponds to a symbol. It is then possible to cluster the collection of documents using a learning algorithm based on the SOM. While this approach could be used in batch or sequential mode, corresponding to the second approach to clustering, a large value of M corresponding to a certain vocabulary is required and it remains computationally intensive even when using dimension reducing methods.

The Symbol String Clustering Map (SCM) is introduced as a novel, efficient algorithm for the non parametric, unsupervised clustering of symbol strings. It falls within the second category of clustering algorithms as it is based on learning the probability of symbols in strings. The SCM is based on a regular lattice structure similar to the SOM lattice and associated with each node of the lattice is a symbol string and a weight vector. Each symbol of the node string is associated with one coefficient of the weight vector. The Winner Take All (WTA) principle is used during the learning and the winning node is defined as the node showing a maximum activation with respect to the current input. The node activation is a simple weighted average of the node weights. A learning rule is applied to each node and it is shown that the node weight vector elements converges towards an approximation of the probability of its corresponding symbol in the symbol string. Lateral inhibition in the form of a Mexican hat is used to form, in an unsupervised manner, the clusters.

In the next section some of the ideas behind learning symbol strings are described and some of the easier convergence properties are formally analyzed. It is shown that the learning rule can be derived from a cost function. In section III based on the learning rule and a related activation function the SCM algorithm is described. A simple example of its operation and expected result is described. Section IV describes how the SCM algorithm can be used to cluster time sequences of symbols, by simply using a hash coding of the time sequence that gives an input suitable for use in the SCM. It is suggested that this approach can be interpreted as a more general, non parametric, approach to Markov Models as applied to clustering time sequence symbol strings. An example of clustering using the SCM with user sessions collected from a WAP service is described. Finally section V is the conclusion.

II. A Learning Rule and Cost Function for Symbol Strings

The SCM algorithm is a symbol string learning algorithm which means it does not store input symbol strings as representative, typically average, symbol strings. To understand what is meant by learning consider the alphabet of possible symbols, $\Psi = \{\psi_1, \ldots, \psi_N\}$ and the sequence of symbol strings $\{\hat{\mathbf{S}}(0), \hat{\mathbf{S}}(1), \ldots, \}$, where $\hat{\mathbf{S}}(t) = (s_{i_1}(t), \ldots, s_{i_{\hat{n}}}(t))$ with $s_{i_j}(t) \in \Psi$ and the constant probability vector $P = (p_1, \ldots, p_N)$ where $p_j \in [0,1]$ is the probability that $\psi_i \in \hat{\mathbf{S}}(t)$. Define a weight vector $\mathbf{X} = (x_1, \ldots, x_N)$ where $\mathbf{X}(0)$ can have any finite random value. At time t update $\mathbf{X}$ as,

$$x_i(t+1) \;=\; \begin{cases} x_i(t) \;+\; \alpha(t)(1.0 - x_i(t)) & \text{if } \psi_i \in \hat{\mathbf{S}}(t) \\ x_i(t) \;+\; \alpha(t)(0.0 - x_i(t)) & \text{if } \psi_i \notin \hat{\mathbf{S}}(t) \end{cases} \tag{2}$$

where $\alpha(t) \in [0, 1.0]$ and $\alpha(t) \to 0, t \to \infty$ is a gain function. This means that if $\psi_i \in \hat{\mathbf{S}}(t)$ the value of $x_i(t)$ is increased towards 1 and if $\psi_i \notin \hat{\mathbf{S}}(t)$ the value of $x_i(t)$ is decreased towards 0. The following theorem can be stated and proved.

Theorem 1: Given $\alpha(t)$ such that $\sum_{t=0}^{\infty} \alpha(t) = \infty$ and $\sum_{t=0}^{\infty} \alpha(t)^2 < \infty$, if $x_i(t)$ is updated according to Eq. 2 then $\forall \; -\infty < x_i(0) < \infty$, $x_i(t) \to p_i$, $\forall \; i$ as $t \to \infty$ with probability 1.

Proof : The proof comes from stochastic approximation theory [11] in which an ordinary differential equation can be associated with Eq. 2 as,

$$\frac{dx_i}{d\tau} \;=\; p_i(1 - x_i) \;+\; (1 - p_i)(0 - x_i) \tag{3}$$

and at the stationary point, $dw_i/d\tau = 0$, then $x_i \;=\; p_i$. It is straightforward to show that the conditions required by the Kushner-Clark theorem (pg 39, [11]) are satisfied in Eq. 2 and as a result of the Kushner-Clark theorem $x_i(t) \to p_i$ as $t \to \infty$ with probability $1 \; \forall \; x_i(0)$. □

This theorem implies that the learning rule presented in Eq. 2 means x_i almost surely converges to p_i, the probability that symbol ψ_i is in $\hat{\mathbf{S}}$, independent of the initial value of x_i. Based on the same argument it is straightforward to show that the learning rule of Eq. (2) leads to a stochastic approximation minimization of the following cost function C,

$$C \;=\; \sum_{i=1}^{N} \; < (\delta_i^s - x_i)^2 > \tag{4}$$

where $' <>'$ is an ensemble average and $\delta_i^s = 1$ if $\psi_i \in \hat{\mathbf{S}}$ and $\delta_i^s = 0$ otherwise.

If the SCM algorithm is to have an advantage over some of the symbol string clustering algorithms described in the introduction then it is assumed that $\hat{n}(t) \ll N$ where $\hat{n}(t)$ is the length of a sample input string $\hat{\mathbf{S}}(t)$ at time t. This of course implies that $p_j \approx 0$ for a large number of $\psi_j \in \Psi$. If this is true then based on the arguments above $x_j \approx 0$ for a large number of the weights in $\mathbf{X}$. In this case the $\mathbf{X}$ is reduced so that $x_{i_j} \in \mathbf{X}$ if and only if $x_{i_j} > x_t$ where x_t is some threshold. This change requires a change of notation, where now, associated with $\mathbf{X}$ is a symbol string $\mathbf{S} = (s_{j_1}, \ldots, s_{j_{\hat{n}}})$ and $\mathbf{X} = (x_{j_1}, x_{j_2}, \ldots, x_{j_{\hat{n}}})$ with coefficient $x_{i_j} > w_t \; \forall x_{i_j} \in \mathbf{X}$ corresponding to symbol $s_{i_j} \in \mathbf{S}$. The cost function of Eq. (4) then becomes,

$$C \;=\; \sum_{i=1}^{N} \; < (\delta_i^s - x_i \delta_i^x)^2 > \tag{5}$$

where now, $\delta_i^x = 1$ if $\psi_i \in \mathbf{S}$ and 0 otherwise. Note that the update Eq. (2) still leads to a stochastic approximation minimization of C in Eq. (5).

The learning algorithm and the cost function C just described apply to a one node SCM algorithm but can be generalized to a $M \times M$ lattice node in the same manner. It has been found that in order to find the winner node for each input $\mathbf{S}(t)$, instead of defining an activation function directly in terms of the cost in Eq. (5), better results are achieved by defining the activation function $\mathcal{A}_k(t)$ for each node k as,

$$\mathcal{A}_k(t) \;=\; \frac{\left(\sum_{j=1}^{N} \delta_j^s \delta_j^x \; x_j(t)\right) \; * \; \left(\sum_{j=1}^{N} \delta_j^s \delta_j^x\right)}{\sum_{j=1}^{N} \delta_j^s \sum_{j=1}^{N} \delta_j^x}, \tag{6}$$

It is seen in the next section that this activation function is maximized by the learning rule used in the SCM algorithm.

III. The SCM Algorithm

The basic structure of the SCM algorithm consists of an $M \times M$ lattice of nodes. Associated with each node k of the lattice are two entities, a symbol string $\mathbf{S}_k$, from now on referred to as the ”node string” and an associated weight vector $\mathbf{X}_k$ where,

$$\begin{aligned} \mathbf{S}_k(t) \;&=\; (s_{i_1}(t), \ldots, s_{i_{n(k,t)}}(t)), \;\; s_{i_j}(t) \in \Psi, \;\forall\; j, t \\ \mathbf{X}_k(t) \;&=\; (x_{i_1}(t), \ldots, x_{i_{n(k,t)}}(t)), \;\; x_{i_j}(t) \in [0,1], \;\forall\; j, t \end{aligned} \tag{7}$$

with $n(k,t)$ the number of symbols in the node string $\mathbf{S}_k(t)$ which is the same as the number of weights in the weight vector $\mathbf{X}_k(t)$. Note that $x_{i_j}(t)$ of $\mathbf{X}_k(t)$ is directly associated with the symbol $s_{i_j}(t)$ of $\mathbf{S}_k(t)$. Initially the weight vectors $\mathbf{X}_k$ and node strings $\mathbf{S}_k$ are set to random values with $x_j(0) \in [0,1], \forall\ k, j$. At time step t there is an input symbol string

$$\hat{\mathbf{S}}(t) \ = \ \big(\hat{s}_1(t), \hat{s}_1(t), \ldots, \hat{s}_{\hat{n}(t)}(t)\big), \qquad (8)$$

The similarity between $\hat{\mathbf{S}}(t)$ and each of the nodes k, is calculated and given by an activation function $\mathcal{A}_k(t) \in [0,1]$ defined in Eq. (6). Based on Eq. (6) the activation function for each node k is a weighted average of the weights of $\mathbf{X}_k$, where the weighting for weight j of $\mathbf{X}_k$ is 1 if symbol j is in the node string and the input symbol string. The weighting is 0 otherwise. This weighted average is then multiplied by a second term, given by the ratio of the number of matched symbols between the node string and the input string and the product of the total number of symbols in the input string and the total number of symbols in the weight vector. It is clear that this second term increases as the number of symbols in common between the input symbol string and the node symbol string increases relative to the total number of symbols in both strings. The winner node $v(t)$ at time t is chosen as the one with maximum activation, hence

$$v(t) \ = \ \underset{1 \le k \le M \times M}{\arg\max}\ \mathcal{A}_k(t). \qquad (9)$$

This represents the first step or WTA of the clustering algorithm, deciding on the winner node that best represents the input. The second stage is the updating of the nodes. For each node $k = 1, 2, \ldots, M \times M$ the following is the update rules for the node string $\mathbf{S}_k(t)$ and the node weight vector $\mathbf{X}_k(t)$,

1) if $\psi_i \in \hat{\mathbf{S}}(t)$, $\psi_i \in \mathbf{S}_k(t)$ and $h_m(v(t),k) > 0$ then
$$x_i(t+1) = x_i(t) + \alpha(t)\ h_m(v(t),k)\ \Big(1.0 - x_i(t)\Big).$$
2) if $\psi_i \in \hat{\mathbf{S}}(t)$, $\psi_i \in \mathbf{S}_k(t)$ and $h_m(v(t),k) < 0$ then
$$x_i(t) \ = \ x_i(t) + \alpha(t)\ |h_m(v(t),k)|\Big(0.0 - x_i(t)\Big).$$
3) if $\psi_i \notin \hat{\mathbf{S}}(t)$ and $\psi_i \in \mathbf{S}_k(t)$ then
$$x_i(t) \ = \ x_i(t) + \alpha(t)\ |h_m(v(t),k)|\Big(0.0 - x_i(t)\Big).$$
4) if $\psi_i \in \hat{\mathbf{S}}(t)$, $\psi_i \notin \mathbf{S}_k(t)$ and $h_m(v(t),k) > 0$ then, $\mathbf{S}_k = \{\mathbf{S}_k, \psi_i\}$, $\mathbf{X}_i = \{\mathbf{X}, x_i(t)\}$ with
$$x_i(t) = \alpha(t)\ h_m(v(t),k).$$
5) if $x_i(t) < x_t$ then remove ψ_i from $\mathbf{S}_k(t)$ and $x_i(t)$ from $\mathbf{X}_k(t)$.

In these rules, $\alpha(t) \in [0,1]$ is a gain that decreases towards 0 as t increases, $h_m(v(t),k)$ is the Mexican hat as a function of the lattice distance d_L between the winner $v(t)$ and the node k being updated. The variation of h_m over time is indicated in figure 1. In rule 1 if ψ_i is in both the input and the node k symbol string with $h_m(v(t),k) > 0$ then weight coefficient $x_i(t)$ is increased towards 1. $h_m(v(t),k)$is positive for nodes close to $v(t)$ on the lattice. For the same case, in rule 2, with $h_m(v(t),k) < 0$, then $x_j(t)$ is decreased significantly towards 0. $h_m(v(t),k) < 0$ if $v(t)$ and k are separated by a greater lattice distance d_l. This corresponds to a form of lateral inhibition. In rule 2 if $s_j(t) \notin \hat{\mathbf{S}}(t)$ then $x_j(t)$ is decreased towards 0 irrespective of the sign of $h_m(v(t),k)$. In rule 3, if $\psi_i \in \mathbf{S}_k(t)$ but is not in $\hat{\mathbf{S}}(t)$ then independent of the polarity of h_m the coefficient $x_i(t)$ is decreased towards 0.0. Rule 4 adds symbol ψ_i to the node k symbol string and a corresponding weight coefficient $x_i(t)$, if the symbol does not already appear in $\mathbf{S}_k(t)$. Finally rule 5 removes any symbols from the node symbol string if the coefficient $x_i(t)$ is below the threshold x_t. Overall the idea is to increase the weights of symbols that are frequently matched between the node string and the input symbol string while eliminating less commonly occurring symbols in the node strings. At the same time the length of the node symbol strings is made to converge to the length of the input symbol strings. The effect of the lateral inhibition is to drive to 0 the weights of symbols in node strings, close to but not immediate neighbors of winning nodes. If the weights of a node are decreased then from Eq. (6) it is less likely to be a winning node later on during training. This enhanced negative effect drives the node strings of nodes that are not often winner to null symbol strings. On the other hand the nodes that are quite often winner and their neighbor's tend to have their weights increased towards 1 and from Eq. (6) are more likely to be winners later in training. The null nodes then serve the purpose of separating the clusters which are represented by sets of neighboring non-null nodes with weight vector coefficients close to 1. Hence the unsupervised formation of clusters can be understood in terms of the probability distribution of the input symbol strings. Frequently occurring symbol strings that represent clusters then appear in the lattice separated from other frequently occurring but quite different symbol strings by null nodes. For the moment the only proof of this behavior is given from simulation. The result of the learning algorithm is best understood from example.

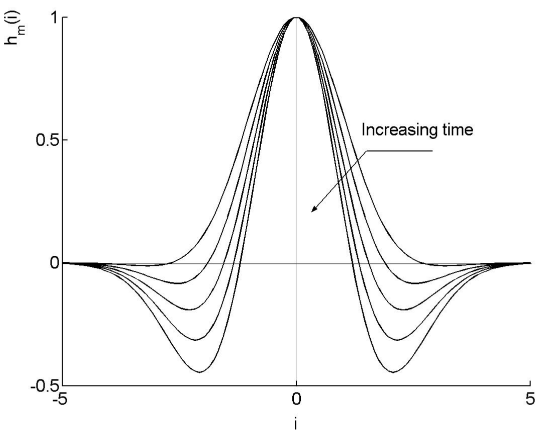


Fig. 1. Time varying Mexican hat function.

A randomly initialized, SCM of lattice size 15×15 nodes, was trained using the following 10 basic symbol strings made up from an alphabet of 50 symbols, $\Psi = \{1, 2, \ldots, 50\}$.

$$\begin{array}{ll} (15, 21, 23, 32) & (17, 28, 29, 32, 34, 35) \\ (7, 9, 13, 29, 32) & (10, 19, 33, 35) \\ (11, 23, 31) & (9, 21, 22, 28, 29, 31) \\ (7, 9, 13, 22, 26, 28) & (28, 34, 35) \\ (14, 17, 32) & (7, 8, 10, 13, 16, 26) \end{array}$$

At each iteration a *noisy* input symbol string was generated by randomly choosing one of the 10 basic symbol strings and making random variations to it. In this case at each time c the number of variations made, is the total length of the basic symbol string multiplied by a random number uniformly distributed in $[0, 0.5]$. Each of the c variations was chosen randomly as either, 1) the insertion of a randomly chosen $\psi_i \in \Psi$ in $\hat{\mathbf{S}}(t)$, 2) the deletion of a randomly chosen symbol from $\hat{\mathbf{S}}(t)$, 3) the replacement of a randomly chosen symbol ψ_i in $\hat{\mathbf{S}}(t)$ by another randomly chosen symbol $\psi_j \in \Psi$. The expected result is an input with 10 distinctive clusters. The gain function $\alpha(t)$ was of the form

$$\alpha(t) \;=\; 10000/(20000 + t) \qquad (10)$$

for $t = 0, \ldots, 40000$ and satisfies the conditions on α in Theorem 1. The value of the threshold in step 4 was a function of time, $x_t(t) = 0.2 * t/40000$. The time variation of h_m is the same as that in figure 1, given as follows.,

$$h_m(i) \;=\; (1.0 - a \star b \star i^2) exp(-a * i^2) \qquad (11)$$

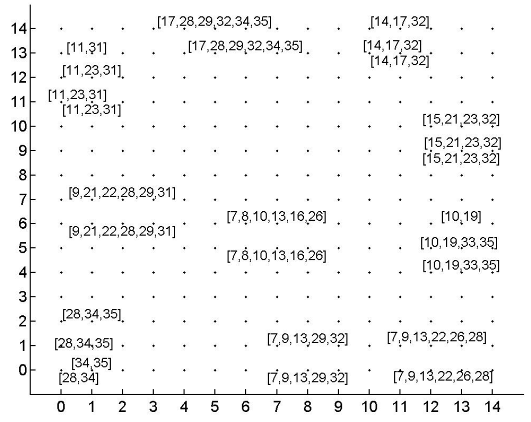
where $a = 0.35$ is a constant determining, the width of the Mexican hat and b varies with time as,

$$b \;=\; 2.0 \times t/40000 \qquad (12)$$

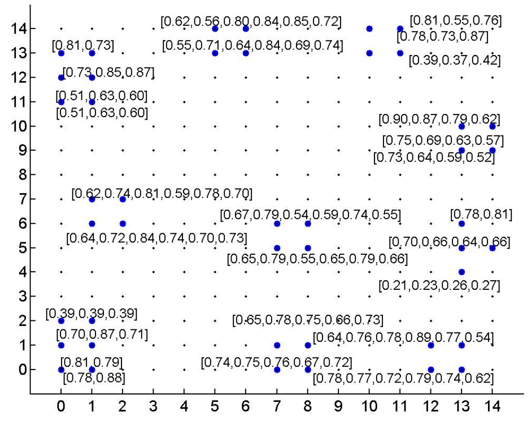
The value of b determines the minimum negative value of the Mexican hat function. Figure 2 (a) shows after training the symbol strings of some of the nodes. Figure 2 (b) shows the corresponding node weight vectors after training. For the sake of illustration not all node strings and weight vectors are shown, however, in 2 (b) the nodes with a null weight vector and symbol string are marked by a '.' while those with a non null weight vector and symbol string are marked by a $'\bullet'$. It is clear that there are 10 distinct clusters each cluster separated from other clusters by neighboring sets of null nodes. On examination each cluster corresponds to one of the original inputs. From the discussion earlier the coefficients of the weight vectors should correspond to the probability of the symbol appearing in the input symbol string associated with that cluster. While the results presented here are based on one simulation, repeating the simulation many times from random initial conditions, in a very robust manner, the SCM algorithm converges to a final state representing the 10 clusters.

IV. Clustering Time Sequence Symbol Strings with the SCM

The discussion so far has concentrated on symbol strings where the ordering of the symbols in the string is unimportant.



(a)



(b)

Fig. 2. (a) Symbol strings for nodes of the SCM. (b) The corresponding weight vectors associated with the symbol strings of each node of the SCM. The nodes '.' have null node string and the $'\bullet'$ nodes have non-null node strings.

There is however another form of symbol string where the ordering of the symbols in the string is important. For example a user who logs into a web site and visits different WEB pages. If each page of the web site is denoted by a distinct integer value then for each user session a symbol string of integers describes the user session. Clustering data from many different user sessions should result in K clusters with each cluster corresponding to a different type of distinctive user session. This type of problem can be analyzed using Hidden Markov Models (HMMs) and a clustering algorithm based on a mixture model as in Cadez et al [12]. However the SCM algorithm provides a non parametric alternative and more general approach than HMMs to clustering this type of sequence.

The SCM algorithm as described in the previous section, that is the update rule and the activation function are very specific to symbol strings where the order of the symbols is not important. However it is quite simple to convert a time sequence symbol string to a symbol string where the order of the symbols is not important allowing the SCM algorithm to be used. The coding is a very simple hash coding [9] of the time sequence symbol string.

Consider 3 consecutive symbols $\tilde{s}(t-2), \tilde{s}(t-1), \tilde{s}(t)$ from a time sequence symbol string where the symbols $\tilde{s} \in \Psi$ the finite alphabet and $N = \mathcal{C}(\Psi)$ the cardinality, or total number of elements in Ψ. If the alphabet is finite and countable then there is always a one-one correspondence between each element of the alphabet and the positive integers. The coding method is easier explained if the symbols are considered to be positive integers, and hence $\Psi = \{1, 2, \ldots, N\}$. Generate a new symbol $s \in \Psi'$ as,

$$s(t) \ = \ \tilde{s}(t-2) * N^2 \ + \ \tilde{s}(t-1) * N \ + \ \tilde{s}(t). \qquad (13)$$

Using this type of encoding, for each triplet of symbols $\big(\tilde{s}(t-2), \tilde{s}(t-1), \tilde{s}(t)\big) \in \Psi \times \Psi \times \Psi$ there is a one to one mapping with an element $s \in \Psi'$. It is also clear that $\mathcal{C}(\Psi') = N^3$. $s_m(t)$ corresponds to the hash coding of an 3-gram used in [9]. This coding can be extended in a general way and the coding is defined to be of order m which gives

$$s_m(t) \ = \ \sum_{j=0}^{m-1} \ \tilde{s}(t-j) * N^j. \qquad (14)$$

In more general terms such an approach can be used to code not just adjacent symbols but also next to adjacent. For example from the triplet $\tilde{s}(t-2), \tilde{s}(t-1), \tilde{s}(t)$, the pair $\tilde{s}(t-2), \tilde{s}(t)$ could be coded to order 2 as $s(t) \ = \ \tilde{s}(t-2) * N \ + \ \tilde{s}(t)$. It is seen that the hash coding is effectively coding the transitions between consecutive symbols in the time sequence symbol string into a new set of symbols. This is essentially the aim of HMMs, however in HMMs traditionally only transitions between adjacent states are analyzed. In general a time sequence symbol string can be coded as,

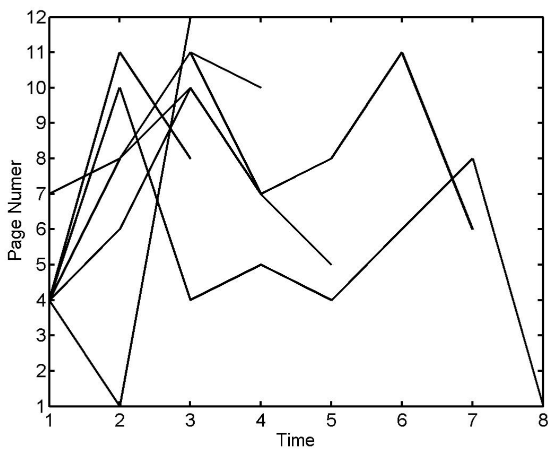
$$\mathbf{S}_{p,q,d} = \Big(\sum_{j=1}^{m} \ \tilde{s}(t - \theta_k(j)) \ N^{j-1} \Big) \qquad (15)$$

for $k = 1, \ldots, (m-1)^{d+1}, m = p, \ldots, q$,where p is the minimum order of coding, q is the maximum order of coding and d is the maximum displacement in the sequence between the symbols that can be coded together and $\theta_k(j)$ is the displacement in the string from the current symbol $\tilde{s}(t)$. $\theta_k(j) = \sum_{i=1}^{j} \ \phi_k(i)$ where ϕ_k is an m dimensional integer vector with $\phi_k(i) \in [1, d+1], i = 2, \ldots, m$ and $\phi_k(1) = 0, \forall \ k$. For example the time sequence symbol string, $(1, 2, 3, 4, 5)$ with $N = 10$, coded to orders $p = 2, q = 3$ and $d = 1$ results in the coded symbol string

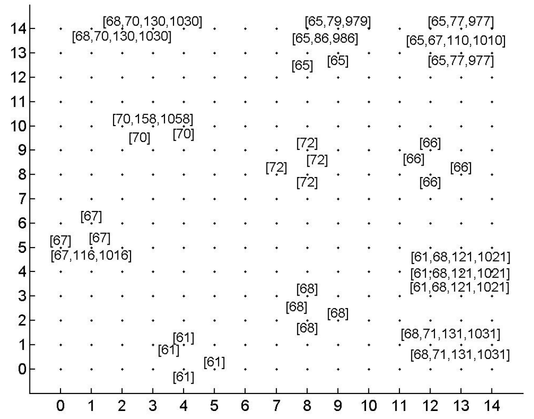
$$(12, 123, 23, 13, 234, 134, 124, 34, 24,$$
$$345, 245, 235, 135, 45, 35)$$

In the coded symbol string it is assumed that the ordering of the symbols is not important as the short range ordering of the original time sequence symbol string has been encoded. While the length of the coded symbol string is longer than the original, the increase in complexity for the similarity measure is proportional to the length of the symbol string for small p, q, d as opposed to the complexity of the similarity measure such a s the Levenshtein measure between two time sequence strings strings that is proportional to the product of the lengths of the two strings.

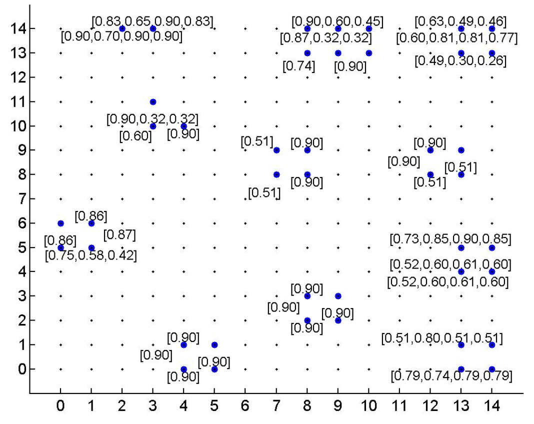
As an example of coding and clustering time sequence symbol strings, user sessions from a real WAP server were used. Each user session corresponds to a time sequence symbol string where each page of the WAP service available was assigned a unique integer value, 1 = Business, 2 = Entertainment, 3 = Feedback, 4 = Front page, 5 = Fun, 6 = My mail, 7 = My page, 8= News, 9 = News send, 10= Sports, 11 = Today, 12 = Weather. Figure 3(a) illustrates some of the sessions which for a WAP service are typically quite short. Note that all the sessions shown begin on the 'Front page' which is typical of the data. The data was coded using $p = 2, q = 3, d = 1$ with $N = 15$ and an SCM with the same parameters as the previous example in section III for $t = 0, \ldots 40000$. There are a total of 25601 user sessions in the data base and at each time t one of these was chosen at random as the input. Figure 3(b) shows the resulting coded node symbol strings on the SCM lattice and figure 3(c) shows the corresponding node weight vectors. Once again for ease of illustration not all node symbol strings and weight vectors are shown, but the nodes with a non null weight vector and symbol string are indicated by a '$\bullet$'. There are 11 distinct clusters on the lattice. Examples of the decoded or time sequence symbol strings associated with each cluster moving from left to right and top to bottom of the SCM lattice are $1030 = [4, 8, 4]$, $986 = [4, 5, 11]$, $1010 = [4, 7, 5]$, $1058 = [4, 10, 8]$, $72 = [4, 12]$, $66 = [4, 6]$, $1016 = [4, 7, 11]$, $68 = [4, 8]$, $1021 = [4, 8, 1]$, $61 = [4, 1]$ and $1031 = [4, 8, 11]$. The different categories of user associated with each cluster are evident, for example cluster $[4, 8, 11]$ users go from the front page to the news to the today page whereas cluster $[4, 10, 8]$ go from the front page to the sports and then to the news, on the other hand cluster $[4, 8, 1]$ users go from the front page to the news to the business page. Using more nodes in the SCM



(a)



(b)



(c)

Fig. 3. (a) Illustration of sample sequences from the WAP sessions. (b) The SCM lattice with some of the node symbol strings. (c) The corresponding SCM and the node weight vectors for nodes marked with "●". Nodes marked by "." have a null weight vector.

lattice it is possible to distinguish more detailed clustering, however the example used here is sufficient for illustration.

V. Conclusions

Symbol strings are a simple means of representing information. The simplest symbol string is one for which the order of the symbols is not important. This is also the easiest type to process. The SCM algorithm is a simple and efficient method for the unsupervised clustering of such strings. It is possible to show in the simple case that the update rule used in the SCM algorithm on the node weights converges to the probability of a symbol being in a symbol string. Using a simple hash coding, symbol strings for which the order of the symbols contains information, can be converted to the simple symbol string and clustered in the SCM. The coding method followed by clustering in the SCM provides a simpler non parametric description of the data than HMMs. The SCM is widely applicable because it makes few if any assumptions about the data and how it is generated except that it contains clusters. This means the SCM does not try to estimate model parameters for a model but rather through learning it gives a non-parametric description of the data based on the most frequently occurring patterns in the data. It provides an efficient, robust and general method for processing very different types of information based on a symbolic representation.

Acknowledgment

The author would like to thank Urpo Tuomela and Heikki Anttila for their continued support of this work.

References

[1] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*, ser. Adaptive Computation and Machine Learning. The MIT Press, 2001.

[2] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[3] J. Shi and J. Malik, "Normalized usts and image segmentation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[4] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, "Fast algorithms for projected clustering," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*. ACM Press, 1999, pp. 61–72.

[5] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, Feb. 1966.

[6] R. Hamming, "Error detecting and error correcting codes," *Bell Sytem Techical Journal*, vol. 9, pp. 147–160, April 1950.

[7] T. Kohonen and P. Somervuo, "Self-organizing maps of symbol strings," *Neural Processing Letters*, vol. 10, pp. 151–159, 1999.

[8] P. Somervuo, "Self-organizing maps for signal and symbol sequences," Ph.D. dissertation, Helsinki University of Technology, 2000, acta Polytechnica Scandinavia, Mathematics and Computing Series No. 107.

[9] T. Kohonen, *Self-Organizing Maps*. Berlin, Heidelberg: Springer, 2001, (Third Extended Edition).

[10] S. Kaski, T. H. K. Lagus, and T. Kohonen, "WEBSOM- self-organizing maps of document collections," *Neurocomputing*, vol. 21, pp. 101–117, 1998.

[11] H. Kushner and D. Clark, *Stochastic approximation methods for constrained and unconstrained systems*. Springer-Verlag, 1978.

[12] I. V. Cadez, S. Gaffney, and P. Smyth, "A general probabilistic framework for clustering individuals and objects," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM Press, 2000, pp. 140–149.