User modelling by classification: a neural-based approach

Russell Beale, Janet Finlay, James Austin, Michael Harrison.

Department of Computer Science, University of York, Heslington, York, Y01 5DD.

ABSTRACT: A neural-based system is used for a classification task in the field of user modelling. The properties associated with neural methods such as learning by example and generalization are exploited to provide a system that overcomes the limitations of traditional approaches, removing the need for an extensive, explicit knowledge base, and exhibiting the desirable properties of domain-independence, noise tolerance and resource efficiency.

1. INTRODUCTION.
Interactive systems should take account of characteristics of each user in order to meet that user’s needs. However most systems are designed around a single canonical model of a user, based on the designer’s assumptions, and so are unable to do this. A solution to this is user modelling (Rich1983), where the system monitors the user’s interactions and adjusts the content or style of its output on the basis of perceived traits such as expertise, task objectives, and similar characteristics.

2. USER MODELLING - THE CURRENT APPROACH.
The underlying assumption in user modelling has been that it requires an explicit, exhaustive knowledge base, containing details of possible user characteristics and their realisation in system interactions, some notions of the objectives that are to be achieved, and so on (Rissland1984). Coupled to this knowledge base is a decision-making strategy which is usually rule-based(Jackson1984) or frame-based (Finin1983).

There are many problems associated with this approach, however. Knowledge acquisition is problematic and time consuming, since the important features present in the interaction may be difficult to identify, and the expert knowledge necessary to set up such a knowledge base may not be available. The system suffers from domain dependency, relying as it does on an explicit representation of knowledge about the specific application. This approach is also noise intolerant, where noise may be errors such as typing errors, incorrect responses to prompts, and so on. Neither the knowledge base nor the decision making strategy usually have the capability of catering for this interference: these approaches are effective only if the erroneous input is explicitly coded into the system.

In this paper we describe a novel approach to user modelling using a neural-based technique. This approach has many advantages; it is example-based rather than knowledge-based, so removing the need for the large knowledge base. The system is able to function in a variety of environments and so provides domain independence, and, due to the neural
system used, resource efficiency. It has the additional properties of other neural systems, such as generalization from the basic example set, which intrinsically gives the system tolerance to noise, as well as allowing it to recognize previously unseen interactions.

3. MODELLING WITH A NEURAL SYSTEM.

The task of user modelling can be viewed as a classification task, in which a user is assigned to a predefined category according to the content of an input or sequence of inputs. A user characteristic, such as expertise or task, forms the base of the classification, and the representation of the interaction must contain sufficient information to differentiate between the classes of user.

A transformation of the interaction into a bit pattern allows us to use a neural system to perform this classification, the network being trained on a set of examples of interactions from each class. Since an example set is relatively easy to identify, compared to the generation of rules about a system, it is possible to apply the method in any domain to act as a classification module within an adaptive or intelligent help system. Obviously the action required in response to a decision is application-specific, but the problem of acquiring knowledge about the system has been mainly overcome since the knowledge is derived from the examples.

The use of examples allows the network to recognize correctly all those interactions which match exactly with one of the taught examples, but the network is also able to generalize from the example set, enabling it to classify previously unseen patterns (Aleksander1979). Such generalization occurs since the network indicates to which example pattern the input pattern is most similar. In terms of pattern space, the example patterns provide centres around which the pattern space is partitioned, and the unseen pattern is classified as the class associated with the example pattern of the region of pattern space into which it is mapped. The generalization properties are a critical feature of the system, since from a few examples a whole range of interactions can be identified without the need for explicit representation. Noise tolerance is also provided by the generalization abilities of the system, since noise can be regarded as a deviation from an ideal ‘quiet’, i.e. non-noisy, interaction, and can be recognised as such.

4. ABSTRACTION.

In order to classify the user it is necessary to translate the interaction or sequence of interactions into a form that is essentially a bit pattern which can then be used as input to the neural system. Successful classification depends on the ability to recognize a feature or pattern within an interaction, and so the transformation process must retain this information. However, in a complete usage trace there is superfluous information which may serve to mask the true features. It is therefore necessary to abstract the relevant, useful information from the usage trace, and then transform this into a bit pattern, moving the problem from that of user modelling to that of pattern recognition.

The question of what constitutes the pertinent information is an important one, since the abstraction must provide identifiable, distinct examples of each class without including confusing, contradictory information. If the classification is to be made on the basis of expertise, errors and requests for help may provide a sound basis for the abstraction, but this would be less appropriate in the case of task identification where command sequences are more relevant. A degree of familiarity with the system can help in identifying a suitable
basis for abstraction, but it is possible to abstract all the major features of the trace in order to make no assumptions as to their relative importance; this will provide a solution, even if it is not an optimal one.

5. THE ARCHITECTURE.

The system architecture is shown in Figure 1.

![Figure 1. The ADAM system.](image)

The usage trace is first abstracted to provide a trace which reflects the pertinent information within the interaction. Each element of the abstracted trace can be represented by an individual bit pattern, and so a whole trace can be represented by a sequence of these bit patterns. These bit patterns are then passed into the neural system, which produces output indicating to which class the pattern belongs. It is possible for many sorts of neural architectures to be used for this purpose, but the one described here has the advantage of learning the example patterns without the need for iterative presentation of the example set, and is resource efficient.

The architecture used is called the ADAM system, and is based on a distributed mapping associative memory, with the addition of n-tuple preprocessing and a thresholding function on the output (Austin1987). The input vector, the bit pattern, is first passed through a constant random mapping function, and then into a tupling function. The tupling process involves selecting single bits from the presented pattern and combining them through a binary logic function to produce an output that is essentially a sparse encoding of the state of the input bits sampled. This tupling provides both a non-linear element in the processing of the pattern which enables patterns that are not linearly separable to be successfully classified, and also accounts for much of the generalization properties of the system. Each tuple state represents some pattern of bits set in the input vector and thus some feature in the interaction. Any input that has this feature will produce the same bit pattern and so will trigger the same response from the tuple. This implies that the tuple responds not to one
specific interaction but to any interaction that has the feature 'seen' by the tuple. The system is picking features from the example input and then classifying previously unseen patterns on the basis of the occurrence of these features.

The matrix is initially empty. This shows the appearance of the matrix after one pattern has been taught.

Appearance of the memory after many patterns have been taught. Presentation of the input pattern produces a response from the memory, which can be thresholded to recover the original pattern taught.

Memory response
Thresholded output
i.e. recalled class pattern

Figure 2. Inside the ADAM matrix.

The tupled input vector is itself a binary vector, and this is passed into a mapping memory. This can be visualised as a matrix of initially unlinked wires, one horizontal wire for each of the possible states of each of the tuples, and a number of vertical wires. In the teaching phase, each input example is presented along with a unique class bit pattern, which contains \( n \) randomly selected bits set to one. The class pattern appears on the vertical wires, whilst the tupled input appears on the horizontal wires. A 'link' is set in the memory matrix wherever an active vertical wire crosses an active horizontal wire. This process is repeated for the whole example set. In recall, the input pattern is presented as before, but this time the class pattern is calculated by summing the number of links in each column that are activated by the tupled input. These totals are then \( n \)-point thresholded to recover exactly the number of bits set in the original class pattern, \( n \). This is shown in Figure 2.
This can be expressed as follows:

Let memory matrix \( M_{ij} \)
Input vector = \( A_i \)
Class vector = \( C_j \)

During teaching,
\[
M_{ij} = \begin{cases} 
1 & A_i, C_j = 1 \\
0 & \text{otherwise} 
\end{cases} \quad \forall \ i, j
\]

During recall, the total output is given by
\[
\text{Recalled vector } R_j = \sum_{p=0}^{i} M_{pj} \cdot A_p \quad \forall \ j
\]

This vector \( R \) is then \( n \)-point thresholded to recover the class vector \( C \).

The response vector \( R \) can also be utilised in its raw form, providing levels of detail about the input vector and its relationship to the taught example set, indicating the degree of correlation between it and the exemplar patterns. This output can in turn be fed into further systems allowing the user to be modelled on a variety of interrelated characteristics that provide a more detailed analysis. This multi-level modelling is facilitated by the fact that the information required by the system can be obtained from the example set and so the problem of coding the relevant knowledge in the correct level of hierarchy is absent.

6. EXPERIMENTAL EVALUATION.

The system was tested using a functional programming environment as its domain. *Glide* is an exploratory functional programming environment at York that supports a lazy functional programming language comparable to Lisp, but with major differences; it is pure, having no assignment, syntactically richer, and supports lazy evaluation (Toyn1987). It is command based with three modes; system, edit, and help. The total command set is small; 24 at the time of data collection. The system is used by staff and research students and is used to teach functional programming to undergraduates.

The characteristic chosen to be modelled was expertise; distinguishing between expert and novice users. This coarse classification was chosen since it is a well-established characteristic for which there is precedent (Chin1986), and it is relatively easy to identify example users in each category. Glide usage was unobtrusively and automatically logged over a period of three months. The tasks performed were of the users’ choice and not explicitly recorded. The logs of the sessions had to be abstracted into a form that reflects the patterns of usage; the features chosen for the basis of the abstraction were command use, help requests, and errors. Features ignored were function definitions since they are unique to each user and therefore difficult to represent, and temporal information because free use of the system makes this too unreliable.

A typical glide log and its corresponding abstraction is shown in Figure 3.

From the novice and expert groups eight exemplar abstractions were chosen at random, and
the system was then asked to classify the remaining data set (176 other traces) on the basis of the example set. The results are shown in Figure 4.

7. DISCUSSION OF GLIDE RESULTS.

The results show that the method succeeds in classifying experts and novices with reasonable success. Comparison with a knowledge-based technique is difficult since no breakdown of results has been reported; significantly, however, the results are better than those obtained using a decision tree technique (Finlay 1989). It is important to note that certain sets of results are better than others; obviously, some training sets will provide more distinct examples of experts and novices, but the variation within the same training sets shows that the tuple mapping used plays an role in determining the success of the classification, and so there should be some optimal, non-random mapping solution.

The domain of the glide experiment is an exceptionally noisy one. Successful classification is regarded as categorising the user into his original class; this does not take into account the fact that novice users may become more advanced as they use the system more, or the fact that some experts were involved in teaching novices and debugging novice actions, all of which are recorded in their traces. Categorization into original classes may not then reflect
the information in the trace; for example an expert who teaches novices may have a novice-type trace and the features within that trace will be recognised and classified as a novice interaction However this is recorded as an error since the user’s original class was expert. Thus the results are worst-case due to the nature of the base classification. It is should be noted that a rule-based system would not be able to distinguish experts interacting in novice manner or vice-versa.

8. CONCLUSION.

The ADAM system has shown itself capable of generating a sufficiently powerful knowledge base by generalizing from an example set. The full potential of the system requires further investigation, but the results obtained so far, as worst-case in a noisy environment, suggest that it is able to better the performance of an inductive learning technique and that respectable results can be obtained without the need for a large implicit knowledge base.

Specific areas have been identified as requiring refinement. The accuracy of the classification is affected most by the position dependence of the features to be recognised; this is one of the major problems inherent in pattern recognition systems as a whole. One prospective solution is to extend the ideas of multi-level modelling to include a hierarchy of feedback systems, which will provide context sensitivity and so a degree of position invariance.
The system at present simply provides a classification of the user based on the example set of classes, but it is an obvious extension to analyse the interaction after classification to determine if it represented an optimal solution to a problem, or if it contained ineffectual and spurious intermediary states. This is obtained by ensuring the example set contains optimal solutions; the interaction is then classified and compared to the ideal approach to the task.

The idea that neural techniques are able to successfully perform user recognition tasks can be extended to encompass other areas such as security, where temporal information reflecting a users typing style may be used as a feature worthy of abstraction and subsequent recognition. It is also challenging to realise that since the system is interpreting the salient features of the interactive abstraction, it can be used as an analysis tool for interface design by highlighting the elements of an interaction that are critical to the successful use of a system.

ACKNOWLEDGEMENTS.
We would like to thank colleagues at York for many discussions on these ideas. This research was funded by grants from the United Kingdom Science and Research Council.

REFERENCES.