**AUCS/TR9508**

# Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface

*Terry R. Payne & Peter Edwards*

Department of Computing Science
University of Aberdeen
King's College
ABERDEEN, AB9 2UE,
Scotland, UK.
E-Mail: {terry|pedwards}@csd.abdn.ac.uk
URL: http://www.csd.abdn.ac.uk/~pedwards/research.html

**Abstract**
In recent years, interface agents have been developed to assist users with various tasks. Some systems employ machine learning techniques to allow the agent to adapt to the user's changing requirements. With the increase in the volume of data on the Internet, agents have emerged which are able to monitor and learn from their users to identify topics of interest. One such agent, described here, has been developed to filter mail messages. We examine the issues involved in constructing an autonomous interface agent which employs a learning component, and explore the use of two different learning techniques in this context.

*Submitted to Applied Artificial Intelligence Journal.*

October 26, 1995

## 1 Introduction

Agents were once seen as anthropomorphic entities which would assist users with daily tasks. They could be used, for example, to locate information of interest to their user (Kay 1984). Ten years later, many definitions of agents have been proposed. The basic concept of a software agent is a system which can aid and assist a user with some common task. The agent may employ some degree of learning or adaptation to improve the quality of its assistance over time. It may be autonomous, in that the agent can learn without needing to be explicitly trained (such as an apprentice system). It may offer assistance only when requested, or act independently of the user. It may also be mobile, travelling in search of useful information or in order to fulfill tasks for the user.

Software agents have been employed to perform many tasks in recent years. These include calendar management and scheduling (Mitchell et al. 1994), information filtering (Sheth 1994; Payne 1994), and intelligent tutoring systems such as UNIX consultants (Nessen 1989; Chin 1991). Languages have been developed for writing agent based systems (Shoham 1993), such as PLACA (Thomas 1993) and Agent-K (Davies and Edwards 1994). Softbots (*Soft*ware Ro*bots*) are intelligent agents which interact with software environments to achieve goals for the user (Etzioni et al. 1994). Agents can be sent out across the Internet (Wayner 1994) to perform specific tasks, and report back to the user once the task has been completed.

Interface agents have been described as "...computer programs which employ Artificial Intelligence techniques to provide assistance to a user dealing with a particular computer application..." (Maes 1994). These *personal assistants* will often have to deal with filtering incoming information, such as electronic mail messages or USENET news articles. As well as acting as intelligent filters, they may assist the user with day to day tasks, such as the completion of personal leave report forms (Hermens and Schlimmer 1993) or arranging meetings as in *CAP* (Jourdan et al. 1993) and the Meeting Scheduling Agent (Kozierok and Maes 1993).

Two important issues must be considered when developing personal assistants which act as intelligent filters:

1. In order to identify information which could be of interest, concepts have to be extracted from the incoming data. Data may be in the form of structured data, where specific information can be found in labeled fields (such as the *From* or *Subject* field in a mail message header) or unstructured data, where different types of information appear together in the same field (as in a USENET news article or a mail message body). With structured data, the problem of extracting concepts is simplified by the fields in which the data is found. For example, the Information Lens (Malone et al. 1987) makes use of predefined templates for mail messages. These templates contain additional fields within the message body which can then be identified.

   Two approaches have been explored in the literature; those using syntactic information, such as keywords, and those using semantic information, as in natural language processing. Baclace (1992) developed a system based on an Agoric Open System approach (Miller and Drexler 1988) which identified specific keywords for personalised information filtering. Other probablistic methods exist based on keywords, such as vector space representations (Salton and McGill 1983) and Latent Semantic Indexing (Foltz and Dumais 1992). *PIES* (Ram 1992) and *SCISOR* (Jacobs and Rau 1990) are systems that make use of natural language processing techniques to identify concepts within articles.

2. Once concepts have been identified, some mechanism must be employed to determine whether the information is of interest to the user. The agent may maintain a knowledge base containing information about its domain, which could be in the form of user profiles, used in identifying articles of interest. By using these profiles, the agent can determine if concepts detected are relevant, and if so, how to act upon the information. The Information Lens (Malone et al. 1987) employs a user defined rule base to determine if an incoming mail message is of interest, and what actions should be performed if it is. Such actions include forwarding the message to another recipient or deleting it. The NewT system (Sheth 1994) uses a genetic algorithm based approach (De Jong 1988) to identify sets of features which can be used to identify USENET news articles of interest. Articles containing these features are then presented to the user. Other genetic techniques, such as mutation and crossover are applied to these features to explore new and possibly interesting USENET topics.

Lewis (1991) has examined issues in applying learning techniques to information retrieval. He identifies two types of learning: *short-term* learning, which occurs within a single session, and *long-term* learning which evolves over many sessions. Short-term learning can improve short term goals, such as specifications for the retrieval of documents, and differentiating between relevant and irrelevant documents. Long-term learning can be used to develop a profile of the information needs and preferences of a particular user over time. It is this second form of learning which has been embodied in many interface agents over recent years.

Traditionally, two approaches have been used to provide an agent with knowledge about a task domain. The first and most common approach is for users to provide their own rules. This is used in many filters, such as the Information Lens (Malone et al. 1987).

This approach possesses a number of inherent problems (Cohen et al. 1994). Many systems utilise a scripting language to specify rules. Learning and utilising a scripting language may discourage non-technical users from using the system. As well as understanding exactly how they require the system to behave, a user must appreciate how the filter will perform with the rules. The behaviour of rules may differ when used in isolation and when used with other rules. The user also has to be responsible for maintaining the rules over time as their interests change. Because of this complexity, the user may not trust the rules to perform as intended, and thus not use the system. One agent based mail system, *MAILTALK* (Cohen et al. 1994) addresses the problem of writing rules, by permitting users to specify rules through natural language. Internal testing environments can be created to simulate the behaviour of the rules and to test them with hypothetical messages.

The second approach is to make use of traditional knowledge engineering techniques (Sheth 1994) to identify background knowledge about the application and the user. Both the SINIX and UNIX consultants SC-UM (Nessen 1989) and UCEgo (Chin 1991) use this approach. They maintain a model of the user's competence, and make suggestions to correct user errors. Whilst this shifts the task of programming the agent from the user to the Knowledge Engineer, the agent will not be customised for a particular user. Thus the approach cannot be used for personalised tasks, such as information filtering.

The use of learning techniques to develop a profile of an individual user's preferences not only eliminates the need for programming rules, but allows the agent to adapt to changes. For example, a user may change their requirements for information filtering over time, perhaps as interests in USENET news topics change; certain users may change their meeting schedules at different times of the year. Recent work on calendar manager agents observed a drop in performance in predicting the user's meeting arrangements at the beginning of semesters in the academic year. This was due to old scheduling priorities being replaced by new ones (Mitchell et al. 1994).

Using an approach which employs machine learning techniques raises a number of issues, such as whether the agent should be specifically trained, or whether it can passively learn through observations; whether assistance is intrusive; and how the user learns to trust the agent. Systems currently learn from the user in a variety of ways (Gil 1994). Approaches such as programming by demonstration provide training examples. Apprentice systems acquire knowledge by observing and analysing the user's behaviour. User feedback can be used to refine the agent's user profile. Once the agent starts to develop such a profile, it may only be able to recognise certain situations where it can be of assistance. When assistance can be provided, it should be accurate, otherwise the user will fail to trust the agent in future situations. The user should be able to override agent decisions, and the agent should learn from this feedback. Approaches to these issues have been explored by various existing agents. Our work in this area has resulted in the development of a Mail Agent Interface, Magi, as a testbed to explore some of them.

## 2 Magi - A Mail Agent Interface

Magi is an agent which aids a user in sorting incoming electronic mail (Payne 1994). In essence, the system is an apprentice which autonomously observes and analyses user behaviour in dealing with mail. By interacting with a graphical user interface, a user can send and read mail messages, and organise their mailbox. Observations can be made and a profile of user preferences generated and refined. This profile, stored in the form of rules or weighted metrics, can be used to predict user behaviour with respect to filing incoming messages. Trust thresholds are used to determine how much confidence the agent has in its predictions, and a browser is provided to allow the user to explore the agent's predictions before actions are performed.

Magi was conceived as a testbed to explore some of the issues raised when machine learning techniques are employed within interface agents. To achieve this, its architecture has to be modular. This allows different learning algorithms to be compared, different prediction strategies to be explored and different feature extraction methods to be evaluated. Our work has concentrated on developing the agent and exploring the use of two different learning algorithms, CN2 (Clark and Niblett 1989) and IBPL1 (see below). The initial motivation for using CN2 was that this algorithm generated human comprehensible rules by performing induction over training examples containing specific features (Payne 1994). We then explored a sub-symbolic learning algorithm, IBPL1, to contrast with the symbolic approach. IBPL1 stores training examples as *exemplars*, and generates weights and distance metrics for the different features within the exemplars. Both algorithms are described in the next section.

### 2.1 Architecture

The architecture for the agent is shown in Figure 1. A graphical mail tool is provided as an interface to a UNIX based mail system. As the GUI is used to organise and file mail messages, observations are made from which the agent can induce a user profile.

These observations are used to generate training examples, by passing the observed mail messages to the feature extraction module. The resulting examples are then used by a learning algorithm to induce the user profile. Incoming mail messages are also passed to the feature extraction module.

Features extracted from the new messages are passed to the classification stage, where the user profile is used to classify them. The results are then evaluated by the prediction stage.

A classification in this context is an action which the agent believes should be performed on the message. The prediction stage evaluates the strength of the classification for each new message, and generates a *confidence rating* for it. The classification and *confidence rating* together form the agent's prediction. The incoming mail message is also placed in the user's mail box; it is on this that the predicted action will be performed.

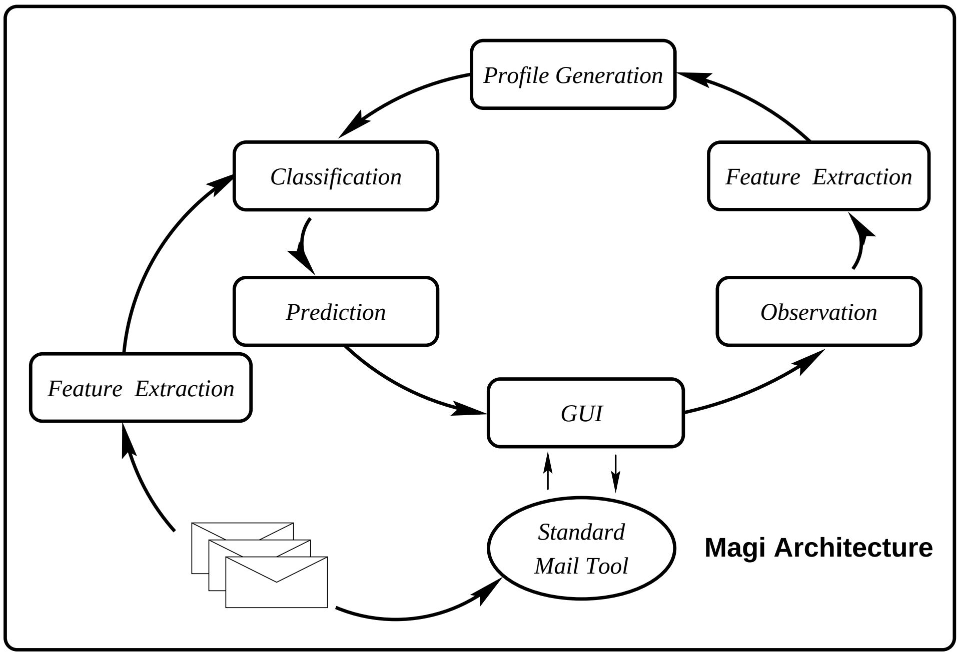



Figure 1: The Magi Architecture.

Predictions can be ignored, examined or executed by the user through the GUI. The user can browse through the predictions to verify the performance of the agent. As the agent improves its predictions, the user develops trust in them, and can instruct the agent to automatically perform the predicted actions on the new mail messages.

This agent architecture can be applied to other information filtering domains. UNA - USENET News Agent (Payne et al. 1995) employs it to identify interesting USENET news articles.

### 2.2 Feature Extraction

This module is responsible for extracting features from mail messages (see Figure 2). It is invoked to extract features from logged messages prior to inducing a new user profile, or to extract features from new messages prior to the agent suggesting an action.

The basic feature extraction module currently in use extracts features from the *From* field, the *Subject* field and the message body. All features identified in the *From* field are extracted. Common words which are held in a *stop-list*, such as *and, the*, etc. are removed from the *Subject* field and the message body. The remaining features in the *Subject* field are then extracted, as are the top $N$

(e.g. 10) most frequently occurring words in the message body.

The agent needs some way of identifying incoming messages for which a prediction is made. The *message-id* is extracted and used by the prediction module for this purpose. Figure 3 shows the features extracted from the mail message in Figure 2 ($N = 5$).

```
Message-Id: <199503031402.OAA23749@pigeon.csd.abdn.ac.uk>
From: terry@csd.abdn.ac.uk (Payne)
Subject: Here's more interesting material on agent research

Regarding your research on an information classifier agent,
here is more research on learning, learning within an agent,
information filtering and what makes a good classifier.
Enjoy,
        Terry.
```

Figure 2: An Example Mail Message.

To summarise, the feature extraction module generates sets of features for each of the three message fields. However, the two learning algorithms used in our study handle these feature sets in different ways. For each field, CN2 requires a single feature in a training example, whereas IBPL1 can handle feature sets. The way in which examples are generated for each algorithm is described below.

| | |
|---|---|
| *Message-Id* = *199503031402.OAA23749@pigeon.csd.abdn.ac.uk* | |
| *From* = { *terry@csd.abdn.ac.uk payne* } | $m = 2$ |
| *Subject* = { *agent interesting material research* } | $n = 4$ |
| *Body* = { *classifier learning research agent information* } | $o = 5$ |

Figure 3: An Example of the Feature Sets extracted from a Mail Message (the number of features in each set is shown to the right).

### 2.3 Classification and Prediction

The exact mechanism used to classify new messages depends on the learning algorithm used; this is discussed below. The mechanisms used in this study generate a *confidence rating* which is a measure of the certainty of the classification. Two threshold values are used by Magi to determine whether to make a prediction; the *predictive threshold* and the *confidence threshold*.

#### 2.3.1 A lower bound threshold - the Predictive Threshold

Once new messages have been classified, the agent has to determine if it can make a valid prediction. This is achieved by having a single lower bound threshold value within the classification stage, known as the *predictive threshold*. If a classification has a *confidence rating* greater than this threshold, then the action determined by the classification will be presented to the user.

#### 2.3.2 An upper bound threshold - the Confidence Threshold

The user can initiate predicted actions made by the agent from the GUI without browsing through them. Certain predicted actions, if incorrect, can be tolerated, such as storing a message in the

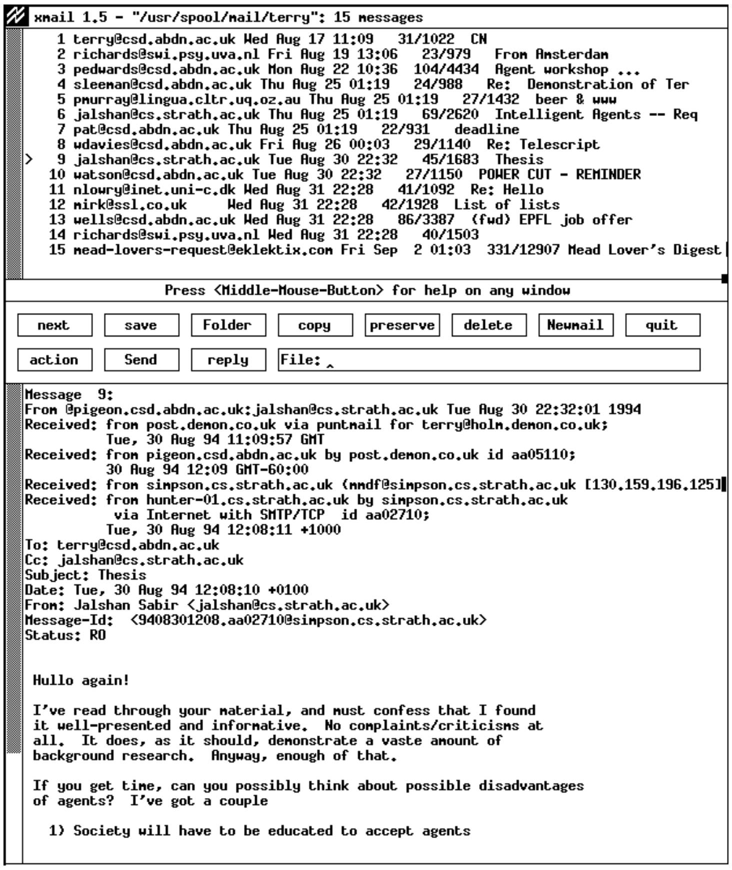



Figure 4: An Example of the Graphical User Interface, Xmail, in use.

wrong mailbox. However, actions such as those that delete mail or forward messages to other recipients are more critical. Thus the agent requires a higher level of certainty in such predictions before performing them. To tailor the level of certainty to the type of action being performed, a higher threshold value is employed by the GUI to determine what predictions require user confirmation before being performed. For each type of action that can be predicted, there exists a *confidence threshold*. If a prediction for a specific action has a *confidence rating* greater than the *confidence threshold* for that action class, then the prediction will be flagged as not requiring user confirmation. This is discussed in greater detail in the next section.

### 2.4 Graphical User Interface

The graphical user interface (GUI) is a platform through which the user interacts with the mail tool. A publicly available mail GUI has been modified to monitor the user performing mail actions, to browse the agent's predictions, and to instruct the mail tool to perform predicted actions. Figure 4 shows this interface, *Xmail*[1], in use.

If the user performs an action of interest to the agent, such as saving a message, a copy of the

[1]Xmail is copyright 1990,1991,1992 by National Semiconductor Corporation.

message and the action performed on it are logged. These logged messages are later used by the agent to induce the user profile. The GUI also handles the agent's predictions, by providing browsing facilities and a means by which the user can instruct the mail tool to perform those predictions which have been flagged for execution.

The prediction browser allows the user to monitor actions suggested by the agent, and thus establish trust in the agent's predictive ability. The browser displays a summary of the predicted actions and indicates those predictions not requiring user confirmation. As described earlier, if the *confidence rating* of a prediction for a specific action is greater than that action's *confidence threshold*, then that prediction will be flagged. Such predictions will be invoked by the agent when instructed by the user. The user can browse the predictions, and either confirm predictions which have not been flagged by the agent, or can reject flagged predictions, thus overriding the agent's decision. This feedback could be used to adjust the *confidence threshold* for each class of actions, such as deletion, etc.

### 2.5 Learning and Profile Generation

During our investigations, two different learning paradigms have been explored; a rule induction algorithm, CN2 (Clark and Niblett 1989) and a modified version of the $k$ - nearest neighbour algorithm ($k$-NN) *Memory-Based Reasoning (MBR)* (Stanfill and Waltz 1986), called IBPL1.

#### 2.5.1 CN2

CN2 uses a divide and conquer approach to induce an ordered rule set. The algorithm works in an iterative fashion, by applying a "best-set-so-far" beam search to a size limited set of *complexes*, where a complex is a conjunction of attribute tests. The complexes considered are then specialised to reduce the coverage of examples of other classes whilst maximising the coverage of a given class. Each new complex is evaluated and ranked using two evaluation functions, to determine its quality and significance. The first is a *Laplacian* error estimate function Eq.(1) which assesses the quality of the complex in making a classification:

$$Accuracy(n, n_c, k) = \frac{(n - n_c + k - 1)}{(n + k)} \tag{1}$$

where $n$ = total number of examples covered by the rule, $n_c$ = number of positive examples covered by the rule, and $k$ = number of classes in the problem.

The second evaluation determines whether the complex is *significant*. A complex is significant if it contains a regularity unlikely to occur by chance, and thus reflects a genuine correlation between attribute values and classes. CN2 measures this by comparing the *observed* distribution among classes of examples satisfying the complex with the *expected* distribution resulting from the complex selecting examples randomly.

This significance value is calculated by using the likelihood ratio statistic Eq.(2) where the distribution $F = (f_1, ..., f_n)$ is the observed frequency distribution, and $E = (e_1, ..., e_n)$ is the expected frequency distribution.

$$2 \sum_{i=1}^{n} f_i \log\left(\frac{f_i}{e_i}\right) \tag{2}$$

The resulting complexes are then combined to produce a set of ordered production rules which can be written to a file (see Figure 5).

```
. . .
IF feature = classifier
THEN action = "s %d +agents" [0 0 0 0 11 0 0 0 0 0 0 0 0]
ELSE
IF subjattr = agent
THEN action = "s %d +agents" [0 0.00 10 0 20 0 0 0 0 0 0 0 0 0.00]
ELSE
. . .
```

Figure 5: Sample Rules generated by CN2.

As mentioned earlier, to generate training examples, feature sets are generated from the mail message. Each such set contains features which could be used to predict an action. A number of examples are generated for CN2 from a given message, each with a single feature for each field. For example, in Figure 6, there are 2 features in the *From* field, a single feature in the *Subject* field and 3 features in the message body. This will result in $2 \times 1 \times 3 = 6$ examples. Figure 7 shows some of the training examples generated from the features shown in Figure 3. The last field in the examples shown contains the action performed on that mail message, i.e. saving the message in the *agents* mailbox.

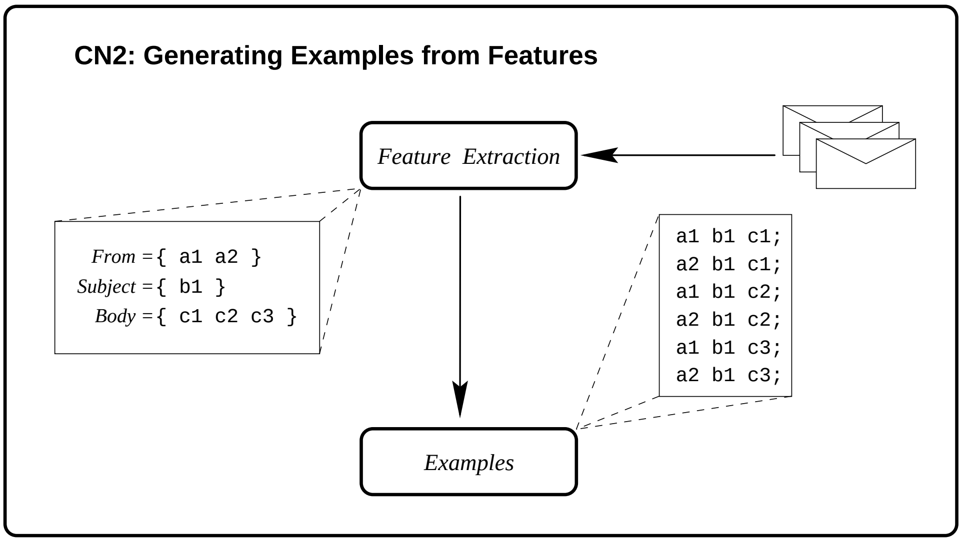



Figure 6: Generating Multiple Examples from Feature Sets.

This large number of examples could bias the algorithm towards features occurring within smaller feature sets over those within larger sets. In the example in Figure 6, there are two examples with the feature `c3` but all six examples contain the feature `b1`. This will affect the *Laplacian* error estimate function Eq.(1) in exploring new complexes.

This method does, however, provide a means of determining a measure of confidence in a classification of a new message. Feature sets are extracted from the incoming message, and from these a number of test examples generated. Each example may fire a rule, but each rule fired may lead to a

| | | | |
|---|---|---|---|
| terry@csd.abdn.ac.uk | agent | classifier | s %d +agents; |
| terry@csd.abdn.ac.uk | interesting | classifier | s %d +agents; |
| terry@csd.abdn.ac.uk | interesting | learning | s %d +agents; |
| payne | agent | learning | s %d +agents; |

Figure 7: A Sample of Training Examples for CN2.

different action (see Figure 8). As in earlier work (Payne 1994) a *confidence rating* is generated for each action by summing the number of rules which fire for that action. The highest rated action is then selected and its rating compared with the *predictive threshold* (determined empirically). If the rating is greater, the action is then suggested to the user.

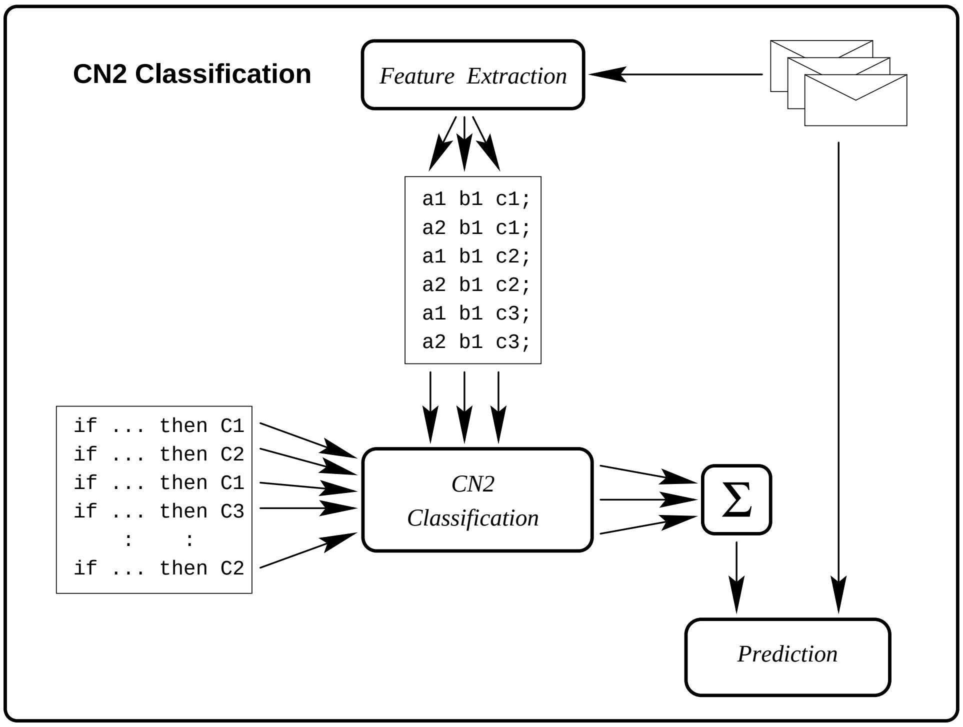



Figure 8: CN2 Classifying Incoming Mail.

A prediction consists of the action with the greatest rating, and a unique reference, so that the agent can associate the prediction with the corresponding message in the user's mailbox.

### 2.5.2 IBPL1

IBPL1 is an extended version of the Memory Based Reasoning algorithm (Stanfill and Waltz 1986). This algorithm compares new feature sets with previously stored exemplars. It can be described as a $k$ - nearest neighbour algorithm (Langley and Iba 1993), as it retrieves the $k$ closest exemplars and bases predictions on a weighted vote, incorporating the distance of each stored exemplar from the test example. Other approaches to nearest neighbour algorithms and instance based reasoning (Cover and Hart 1967; Aha et al. 1991) have been explored.

As described earlier, a large number of examples (*instances*) containing single values for each attribute were generated when the CN2 algorithm was used (see Figure 6). To overcome this, the original MBR algorithm was modified to learn from *sets* of values for each attribute, i.e. each attribute can contain one or more unordered values within a single instance. The new algorithm considers all the distances within two respective feature sets (Figure 9) when calculating the distance between two instances. These distances are then averaged to find an overall distance value. Figure 10 shows how incoming messages are passed to IBPL1 for classification.

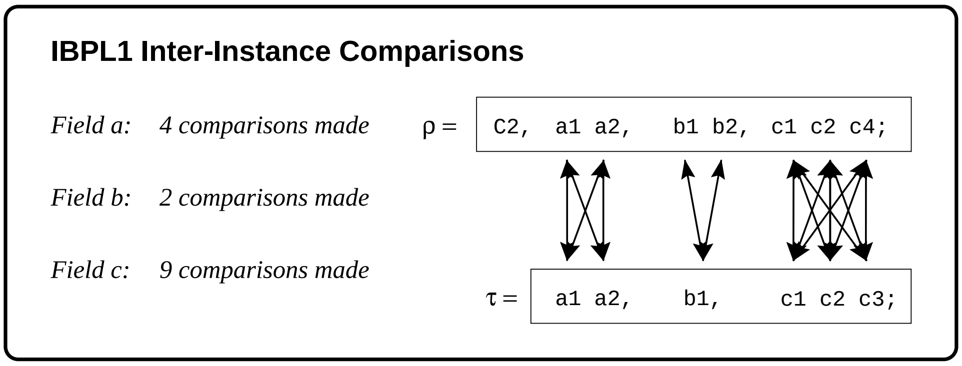



Figure 9: Comparisons Made between an Instance ($\tau$) and an Exemplar ($\rho$) by IBPL1.

Before the IBPL1 algorithm can be discussed in detail, some terms must be defined. Instances used in training, known as *exemplars*, and instances which are classified, known as *examples*, are denoted by the Greek letters $\rho$ and $\tau$ respectively. These instances contain *fields*, such as *From* and *Subject* used in our study. Each field contains a set of *features*, as shown in Figure 3. A field $f$ in an instance $\rho$ is written $\rho.f$. Likewise, a feature $i$ in that field would be $\rho.f.i$. The set of all possible values $\mathcal{V}$ for a field $f$ is written $\mathcal{V}_f$. The goal field, $g$, contains the classification of an exemplar $\rho$ or example $\tau$. An instance which contains the value $v$ in the goal field $g$ is represented as $[g = v]$. Likewise, an instance which contains a specific value in the feature set $i$ for the field $f$ is written as $[i = \rho.f.i]$. The set of memorised exemplars is known as the *database*, and is written $\mathcal{D}$. This can be restricted to only contain instances which contain a specific feature for a given field, as in $\mathcal{D}[i = \rho.f.i]$. This subset can then be counted ($|\mathcal{D}[i = \rho.f.i]|$).

The distance ($\Delta^g(\mathcal{D}, \tau, \rho)$) between a new example $\tau$ and a memorised exemplar $\rho$ with a goal field $g$ in the database $\mathcal{D}$ is computed as the sum of all partial distances for $f \in P_\rho$, i.e. the partial distance for each field $f$ in the exemplar $\rho$ Eq.(3).

For example, in Figure 9, there are three fields, $a$, $b$ and $c$. The goal field $g$ for the exemplar $\rho$ has the value `C2`. In field $a$, the exemplar $\rho$ contains 2 features, as does the example $\tau$. Hence, 4 comparisons are made for this field. Likewise, 2 comparisons will be made for field $b$, and 9 comparisons made for field $c$. The resulting distances will then be averaged for each field.

The partial distance ($\delta^g_f(\mathcal{D}, \tau.f, \rho.f)$) Eq.(4) is calculated by taking the average of the distances for each of the features in the two sets. This distance between $\tau.f.i$ and $\rho.f.j$ is the product of the weighting value of $i$ and the feature distance between $i$ and $j$, i.e. $w^g_i(\mathcal{D}, \tau.f.i)$ defined in Eq.(5) and $d^g_{ij}(\mathcal{D}, \tau.f.i, \rho.f.j)$ defined in Eq.(6). These values are based on calculating the proportion of exemplars in the database which contain a given feature for a given classification ($|\mathcal{D}[i = \tau.f.i][g = v]|$) over the total number of exemplars with that feature ($|\mathcal{D}[i = \tau.f.i]|$). This

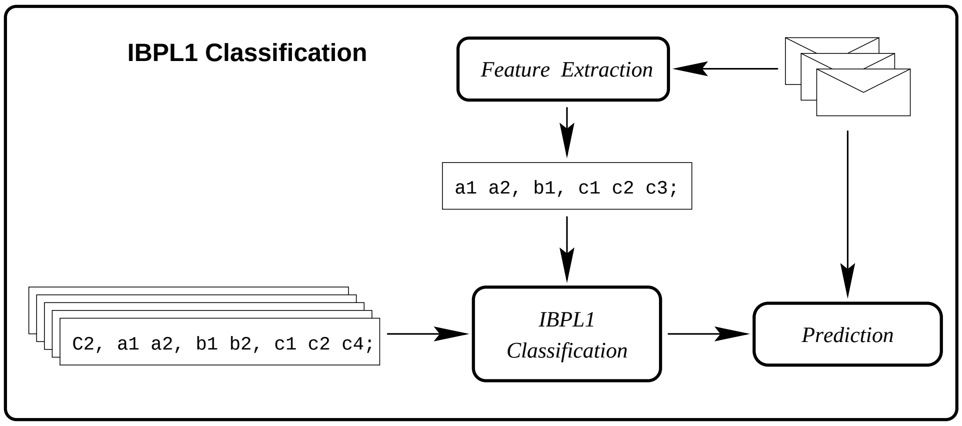



Figure 10: IBPL1 Classifying Incoming Mail.

metric is used to determine the quality of a feature as a good classifier.

$$\Delta_g(\mathcal{D},\tau,\rho) = \sum_{f \in P_\rho} \delta_f^g(\mathcal{D},\tau.f,\rho.f) \tag{3}$$

$$\delta_f^g(\mathcal{D},\tau.f,\rho.f) = \frac{\sum_{i \in I_{\tau.f}} \sum_{j \in J_{\rho.f}} d_{ij}^g(\mathcal{D},\tau.f.i,\rho.f.j) w_i^g(\mathcal{D},\tau.f.i)}{|I_{\tau.f}| \times |J_{\rho.f}|} \tag{4}$$

$$w_i^g(\mathcal{D},\tau.f.i) = \sqrt{\sum_{\upsilon \in \mathcal{V}_g} \left( \frac{|\mathcal{D}[i=\tau.f.i][g=\upsilon]|}{|\mathcal{D}[i=\tau.f.i]|} \right)^2} \tag{5}$$

$$d_{ij}^g(\mathcal{D},\tau.f.i,\rho.f.j) = \sum_{\upsilon \in \mathcal{V}_g} \left( \frac{|\mathcal{D}[i=\tau.f.i][g=\upsilon]|}{|\mathcal{D}[i=\tau.f.i]|} - \frac{|\mathcal{D}[j=\rho.f.j][g=\upsilon]|}{|\mathcal{D}[j=\rho.f.j]|} \right)^2 \tag{6}$$

Once the distances $\Delta^g$ Eq.(3) have been calculated between the example and every memorised exemplar, they are sorted to find the top $k$ closest ones. These are then used to generate a *confidence rating*. A score is determined for each of the closest memorised exemplars by taking the reciprocal[2] of the distance $\Delta^g$ Eq.(7). The top $k$ exemplars with a classification $\kappa$ are then summed for each $\kappa$ Eq.(8). The classification $\kappa$ for the example is determined by the class of the highest scoring exemplar. The confidence in this classification is the difference between the confidence of the resulting classification ($\kappa$), and the confidence in other classifications ($\neg\kappa$), as shown in Eq.(9).

$$Score^g(\mathcal{D},\tau,\rho) = \frac{1}{\Delta^g(\mathcal{D},\tau,\rho) + 0.01} \tag{7}$$

$$Confidence^{\upsilon \in \mathcal{V}_g}(\tau) = \sum Score^{\upsilon}(\mathcal{D},\tau,\rho) \tag{8}$$

[2] A small value, in this case 0.01, is added to avoid dividing by zero.

$$Confidence^{g}_{\kappa}(\tau) = Confidence^{[g=\kappa]}(\mathcal{D}, \tau, \rho) - \sum_{\upsilon \in \neg\kappa} Confidence^{[g=\upsilon]}(\mathcal{D}, \tau, \rho) \tag{9}$$

Unlike CN2, which induces rules for use later in performing classifications, IBPL1 builds weight tables. Each training instance consists of three fields (corresponding to the *From, Subject* and message body), each of which contains a feature set. Figure 11 shows an example of a training instance, for an action to file a mail message in the *agent* mailbox. The example consists of fields separated by a comma (,) and terminated by a semicolon (;). The first field contains the classification of the example, and the remaining fields contain a space separated list of features.

### 2.5.3 Adapting to changes

As new observations are made and new training examples created, the examples are time stamped. This way the training set can be pruned with respect to time. Not only does this help reduce the number of training examples used, but also removes old examples which may become out of date as the user's behaviour changes. Aha *et al.* (1991) discuss other methods of reducing the training set for nearest neighbour approaches.

*s %d +agents,*
*terry@csd.abdn.ac.uk payne,*
*agent interesting material research,*
*classifier learning research agent information;*

Figure 11: A Single Training Episode for IBPL1.

Both learning approaches we have considered differ in the way new training examples are integrated into the user profile. CN2 periodically (e.g. every night) induces a new ruleset, pruning out old examples and adding new examples. Hence new examples do not affect performance until new rules are induced. IBPL1, however, introduces new examples into the training set as soon as they are created, and thus the effects are immediate.

## 3 Method of Experimentation

The central component of Magi is the learning algorithm used by the agent to discover user preferences and thus make predictions. Our work has examined the issues involved in learning within an agent architecture, and has studied the applicability of the two learning paradigms described above to the mail filtering task.

### 3.1 Method and Dataset

In order to explore the use of learning within mail filtering, the two algorithms were tested with various numbers of training examples. The training examples were generated by extracting messages from existing mailboxes. These were then used to train the algorithm being tested. The remaining messages in the mailboxes were used to test the ability of the algorithm to recognise and classify new messages.

The data consisted of twelve different mailboxes, each containing different numbers of messages with different properties. These are listed in Table 1, with the number of messages in each mailbox and a description of the mail topic. The *From* and *Subject* fields are described as *various, similar* and *digest*. Mailboxes whose fields are marked as *various* contain various values such as different senders or different subject lines. Those with *similar* fields contain values which are consistent for the majority of messages, such as *cfp* where most of the messages originated from the same source. Mailing list digests, marked as *digest*, contain individual messages which have been grouped together by a moderator (either manually or automatically). Thus the message body features selected represent the digest, as opposed to any individual topic within the digest. The mailbox *tea* originates from a single source and contains very few entries.

Tests were performed by randomly dividing the messages from all the mailboxes into training and test data. The learning algorithm being tested was then trained with the training data. Two measurements were taken; the number of predictions made when testing with the test data, and the number of these predictions made which were correct. The proportion of messages in the training set was varied from 20% to 80% in 10% intervals. All the tests were repeated 30 times so that statistical measurements could be made. The experimentation was repeated for both learning algorithms described above.

| **Mailbox** | **Number** | **From** | **Subject** | **Notes** |
|---|---|---|---|---|
| agents | 61 | various | various | *Messages about the specific topic of 'agent' research.* |
| cfp | 11 | similar | various | *Messages containing 'calls for papers', with the majority of messages originating from the same source.* |
| cure | 53 | various | various | *Messages from a music related mailing list.* |
| dai | 9 | digest | digest | *Monthly periodic mailing list digest.* |
| jobs | 25 | various | various | *Messages relating to posts in academia from various sources.* |
| kdd | 27 | digest | digest | *Monthly periodic mailing list digest.* |
| mbox | 20 | various | various | *Messages covering a wide range of topics stored for later use.* |
| mead | 90 | digest | digest | *Regular mailing list digest (two to three per week). Two messages have been incorrectly saved in this box, thus acting as noise.* |
| personal | 32 | various | various | *Personal communications covering a wide range of topics.* |
| phd_pos | 64 | various | various | *Messages on a specific topic from a small number of users.* |
| phd_stuff | 13 | various | various | *Various messages relating to research.* |
| tea | 3 | similar | various | *Messages on a specific topic.* |

Table 1: Characteristics of the Different Mailboxes used in Experimentation.

The following tests were performed to identify factors which may influence the number of predictions made and their accuracy:

- Changing the number of message body features used.
- Omitting features from one of the three fields.

- Varying the lower bound *predictive threshold*, responsible for proposing predictions to the user.
- Using a dictionary to filter out incorrectly spelt or nonsense words before extracting features from the message body.

The rules generated by CN2 were also examined to determine the number of rules containing features from each field, and the number of rules generated for each action.

### 3.2 Hypotheses

By looking at the general characteristics of the different mailboxes, certain predictions can be made:

- Tests on digests should result in a high number of accurate predictions, due to the similar *From* and *Subject* field entries.
- Predictions on mailboxes such as *personal* and *mbox* are expected to be poor as these contain messages from a wide variety of users regarding a wide range of topics.
- The smaller mailboxes, *cfp, dai* and *tea* may not perform as well as the larger ones due to the limited number of messages used for training.
- IBPL1 should perform poorly on mailboxes where there are fewer than $k$ training examples, due to the voting strategy used by this algorithm.
- Due to the way training and testing examples are generated for CN2, the *From* and *Subject* features are expected to be more significant in inducing rules and consequently more important in making predictions.

These tests are designed to explore some of the issues regarding the use of learning algorithms in the context of filtering electronic mail messages. For empirical comparisons between instance based algorithms and rule based algorithms, see (Tan and Schlimmer 1990; Aha et al. 1991; Aha 1991).

## 4 Results

The following comparisons only make use of the lower bound *predictive threshold* (set to 0.3). No upper bound *confidence threshold* was used. This enables comparisons to be made between mailboxes. The results contained in the tables which appear on the following pages show:

- The number of messages for which predictions were made (coverage) for each learning algorithm (with standard deviation).
- The accuracy of the predictions made for each learning algorithm (with standard deviation).

### 4.1 Comparing IBPL1 & CN2

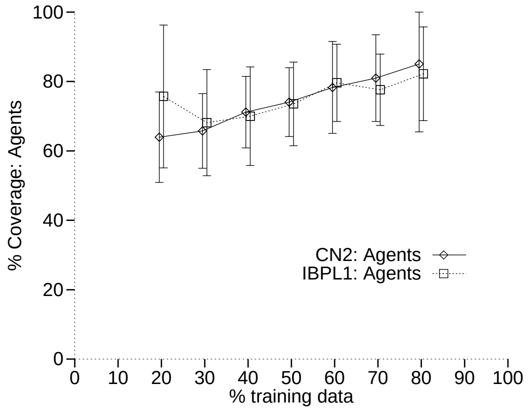
#### 4.1.1 Initial comparison

CN2 and IBPL1 were trained with examples containing features from all three fields (*From*, *Subject* and the 10 most frequently occurring words from the message body).

Both algorithms were able to make predictions on new messages (approximately 78% of messages overall had actions predicted when using CN2, compared to 74% when using IBPL1), although this varied depending on the mailbox tested. For example, when using CN2, approximately 57% of messages from the *jobs* mailbox generated predictions compared to 90% of messages from the *kdd* mailbox. Figure 12 shows a comparison of CN2 with IBPL1 when testing on messages from the *agents* mailbox.

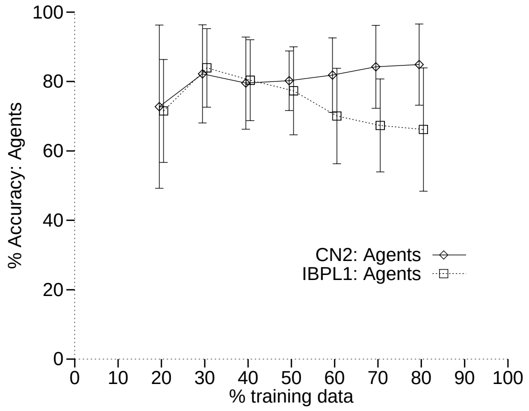
With most mailboxes, the number of predictions made by IBPL1 initially falls as the number of training examples increase, but then increases with larger training sets. This this would typically occur after about 50% of the data had been used to train the algorithm. The number of predictions steadily rises with CN2 as the number of training examples increases.

| % | Coverage when using CN2 | | Coverage when using IBPL1 | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 63.95 | 13.02 | 75.70 | 20.59 |
| 30 | 65.75 | 10.76 | 68.14 | 15.31 |
| 40 | 71.19 | 10.30 | 70.00 | 14.19 |
| 50 | 74.06 | 9.92 | 73.55 | 12.05 |
| 60 | 78.30 | 13.25 | 79.62 | 11.13 |
| 70 | 80.97 | 12.50 | 77.63 | 10.28 |
| 80 | 85.04 | 19.55 | 82.22 | 13.51 |




The Coverage of Predictions Made on the *Agents* Mailbox.

| % | Accuracy when using CN2 | | Accuracy when using IBPL1 | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 72.75 | 23.51 | 71.52 | 14.83 |
| 30 | 82.20 | 14.15 | 83.93 | 11.32 |
| 40 | 79.55 | 13.28 | 80.37 | 11.67 |
| 50 | 80.24 | 8.59 | 77.32 | 12.68 |
| 60 | 81.86 | 10.73 | 70.05 | 13.76 |
| 70 | 84.25 | 11.96 | 67.35 | 13.38 |
| 80 | 84.88 | 11.69 | 66.17 | 17.77 |



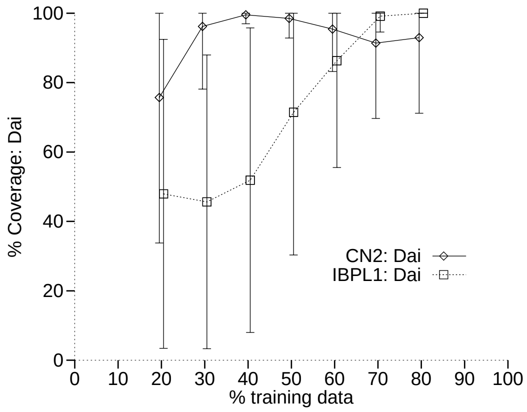

The Accuracy of Predictions Made on the *Agents* Mailbox.

Figure 12: Comparing CN2 with IBPL1 on the *Agents* Mailbox.

The number of predictions made by CN2 was only slightly higher than that made by IBPL1. However, approximately 65% of predictions made by CN2 were correct, compared to approximately 57% for IBPL1. The number of correct predictions made by IBPL1 tends to increase and peak at 50% of the data, then falls off as the number of training examples increases. This appears to correspond with the drop in the number of predictions made by IBPL1. This relationship between the number of predictions made and their accuracy is discussed below.

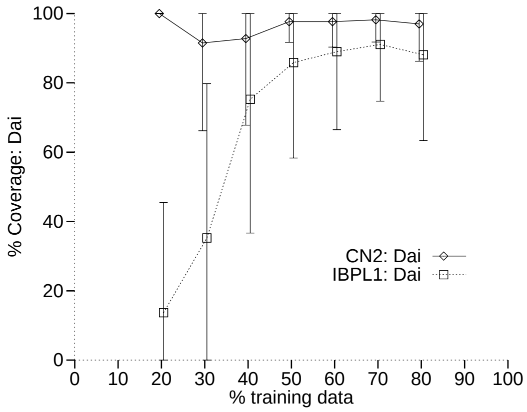
For the smaller mailboxes, such as *cfp, dai* and *tea*, a larger number of predictions are made by CN2 than IBPL1. The accuracy of the predictions made by CN2 is low for *cfp* (only approximately 10% of predictions were correct) but much higher for *dai* and *tea*, both of which result in prediction accuracies over 90%. The accuracy of IBPL1 predictions varied but overall was poor, with no accurate predictions being made for *tea* and only approximately 2% to 12% of *cfp* predictions being accurate. There is some evidence that as the number of messages used to train increases, so does the accuracy of predictions made by IBPL1 (see Figure 13).

| % | Coverage when using CN2 | | Coverage when using IBPL1 | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 75.71 | 41.92 | 47.94 | 44.51 |
| 30 | 96.19 | 18.05 | 45.63 | 42.34 |
| 40 | 99.52 | 2.56 | 51.88 | 43.88 |
| 50 | 98.50 | 5.65 | 71.44 | 41.10 |
| 60 | 95.44 | 12.25 | 86.33 | 30.79 |
| 70 | 91.38 | 21.72 | 99.14 | 4.56 |
| 80 | 92.95 | 21.77 | 100.00 | 0.00 |




The Coverage of Predictions Made on the *Dai* Mailbox.

| % | Accuracy when using CN2 | | Accuracy when using IBPL1 | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 100.00 | 0.00 | 13.68 | 31.82 |
| 30 | 91.52 | 25.34 | 35.25 | 44.54 |
| 40 | 92.78 | 24.98 | 75.24 | 38.59 |
| 50 | 97.67 | 5.97 | 85.87 | 27.58 |
| 60 | 97.67 | 7.36 | 88.99 | 22.50 |
| 70 | 98.21 | 6.44 | 91.09 | 16.37 |
| 80 | 97.00 | 10.77 | 88.10 | 24.74 |




The Accuracy of Predictions Made on the *Dai* Mailbox.

Figure 13: Comparing CN2 with IBPL1 on the *Dai* Mailbox.

This could be explained by considering the voting strategy used by IBPL1. The top $k$ closest matches are considered when determining the confidence and prediction. If the number of training instances is less than $k$, then any similar matches, however poor, will be taken into consideration when determining these values. Tests on smaller values of $k$ showed an increase in performance for these two mailboxes. Cover (1968) demonstrated that a larger value of $k$ results in an improvement in the behaviour of large samples, at the expense of small sample behaviour.

Both CN2 and IBPL1 were able to generate predictions for a high proportion (85%+) of messages from two of the digest mailboxes, *kdd* and *mead* with near 100% accuracy. The third, smaller digest mailbox, *dai*, resulted in a slightly lower number of predictions when CN2 was used. When using

IBPL1, training with smaller proportions of training examples (e.g. 30%) resulted in a small number of predictions. The accuracy of these predictions was also low. As the number of training examples increased, the performance of IBPL1 improved. As this mailbox originally contained 9 messages, this supports the explanation given above regarding $k$ - nearest neighbour voting strategies. Similar performance was exhibited when recognising messages from the *tea* mailbox.

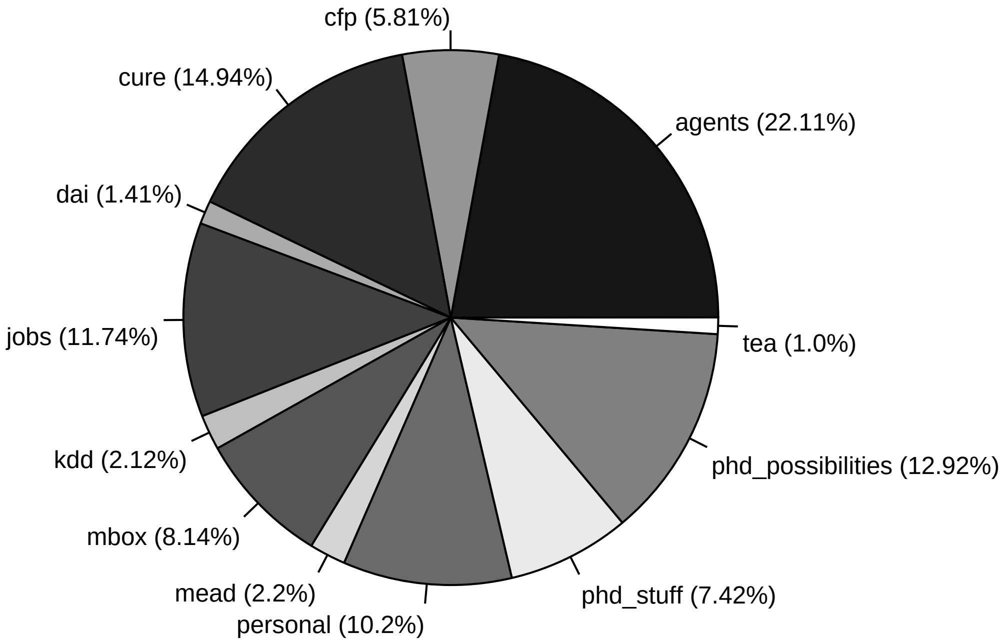



Figure 14: Percentage of Rules used for different classifiers.

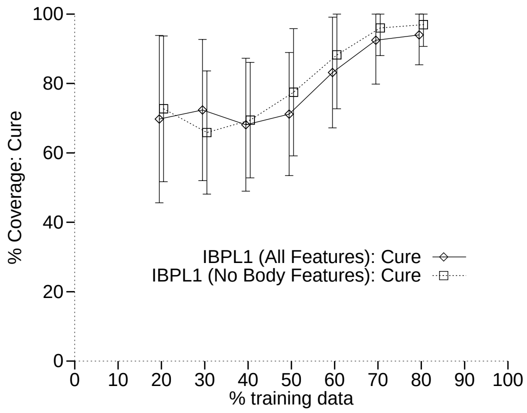
Figure 14 shows the proportion of rules generated for each classification by CN2. The digest mailboxes rely on a very small number of rules. This is due to the *From* and *Subject* fields being similar across all messages within a digest. However, for other mailboxes which have different *From* and *Subject* features, there is a steady increase in the number of rules as the number of messages in the mailbox increases. This indicates that there are few features which are common to all messages within a given mailbox. Because of this, a larger number of rules are needed to cover the larger number of less common features found in the messages.

### 4.1.2 Changes in body features

Tests on reducing the number of message body features used, and omitting such features altogether indicate that there is no significant gain in employing these features when the *From* and *Subject* fields are employed. Previous work (Payne 1994) demonstrated that whilst predictions could be made on mail messages based purely on message body features, a significant improvement in performance was achieved by including the *From* and *Subject* fields. The work here expands on this previous study by exploring the effects of changing the number of message body features used whilst employing these extra fields.

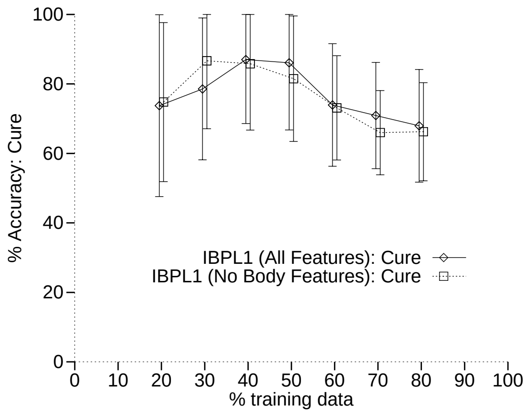
With CN2 there is a slight but not significant degradation in performance overall by reducing or

| % | Coverage with Body Features | | Coverage with No Body Features | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 69.74 | 24.13 | 72.70 | 20.99 |
| 30 | 72.36 | 20.35 | 65.88 | 17.76 |
| 40 | 68.11 | 19.16 | 69.45 | 16.63 |
| 50 | 71.19 | 17.75 | 77.49 | 18.34 |
| 60 | 83.17 | 15.94 | 88.24 | 15.54 |
| 70 | 92.45 | 12.62 | 95.99 | 7.94 |
| 80 | 94.01 | 8.63 | 96.98 | 6.29 |




The Coverage of Predictions Made on the *Cure* Mailbox.

| % | Accuracy with Body Features | | Accuracy with No Body Features | |
|---|---|---|---|---|
| | mean | s.d. | mean | s.d. |
| 20 | 73.75 | 26.16 | 74.77 | 22.88 |
| 30 | 78.56 | 20.43 | 86.65 | 19.56 |
| 40 | 87.00 | 18.39 | 85.80 | 19.07 |
| 50 | 86.07 | 19.30 | 81.51 | 18.05 |
| 60 | 73.95 | 17.64 | 73.11 | 15.01 |
| 70 | 70.90 | 15.30 | 65.98 | 12.12 |
| 80 | 67.95 | 16.21 | 66.25 | 14.12 |




The Accuracy of Predictions Made on the *Cure* Mailbox.

Figure 15: Comparing the Use and Absence of Message Body Features on the *Cure* Mailbox.

omitting message body features. A drop in the number of predictions occurs for all non-digest mailboxes with the exception of *personal* and *phd_possibilities*. The accuracy of the predictions made is also slightly lower. The digest mailboxes were unaffected by changes in the use of message body features, as predictions for these messages are based on the contents of the *From* and *Subject* fields only.

If features from the message body are omitted when using IBPL1, then the number of training examples required to achieve the same level of accuracy is reduced. Figure 15 shows the number of messages for which predictions were made and the accuracy of these predictions when message body features are present and absent.

We have also explored the use of a dictionary with IBPL1. This examined the effect of restricting the message body features used to those found in common English. This does of course result in technical terms being filtered out that could be used in identifying certain topics. The results showed that in all cases the number of predictions fell, but the accuracy of the predictions increased.

### 4.1.3 Omitting other fields (*Subject* and *From*)

If the *Subject* features are omitted when using IBPL1, then the effect is similar to omitting message body features (see Figure 15). However, whilst the number of predictions made increased, the accuracy of those predictions fell. This indicates the importance of *Subject* features in making accurate predictions when using IBPL1.

With CN2, there was a reduction in the number of predictions made for some of the mailboxes when *Subject* features were omitted, namely *agents, cure* and *personal.* However, the accuracy of the predictions made for messages in these mailboxes increased. Otherwise there was no significant difference in the number of predictions made or the accuracy of those predictions for messages from the other mailboxes.

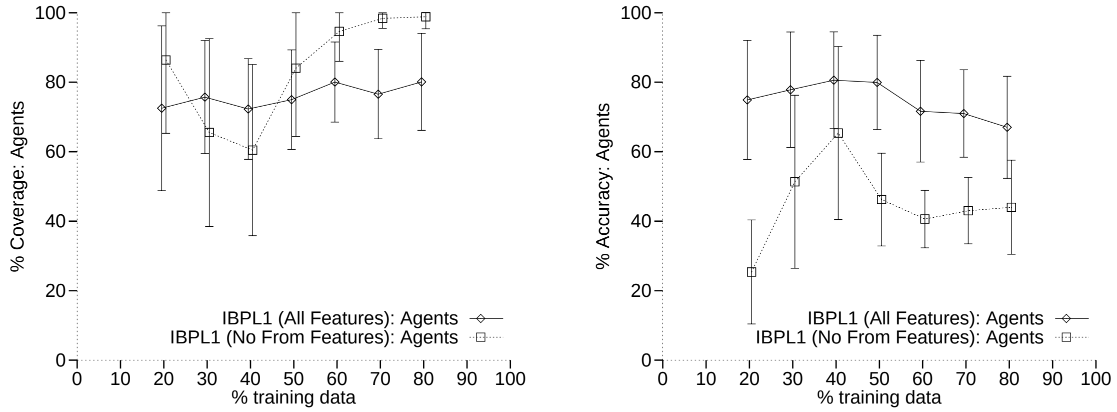



Figure 16: Comparing the Use and Absence of *From* Features with IBPL1 on the *Agents* Mailbox.

From our investigation, CN2 appears to be more dependent on the *From* features than IBPL1. In most cases, omitting the *From* features when using CN2 resulted in a reduction in the number of predictions made. This drop was more pronounced for certain mailboxes, such as *cfp, mbox, phd_stuff* and *phd_possibilities.* The change in accuracy of the predictions varied depending on the mailbox. For example, there was no noticeable change in the accuracy of predictions for messages from the *phd_possibilities* mailbox, yet the accuracy of predictions for *phd_stuff* increased by approximately 15-20%. Some mailboxes (such as *jobs*) resulted in an increase in accuracy without an overall increase in the number of predictions made.

Figure 16 shows the behaviour of IBPL1 for non-digest mailboxes when *From* features are omitted. This behaviour can be explained by considering the performance of IBPL1 before and after 40% training data:

- As the proportion of the data in the training set increases to 40%, the number of predictions made decreases. However, the accuracy of these predictions increases.
- As the training data increases beyond 40%, the number of predictions made increases again. There is an initial reduction in the accuracy which then levels out at approximately 47%.

The indications here are that with smaller training sets, IBPL1 depends heavily on the *From* features to make predictions. When they are omitted, many incorrect predictions are initially made. As the training set increases in size this number drops, whilst the number of accurate predictions made increases.

## 4.2 Other issues

### 4.2.1 Nature of rules generated by CN2

Figure 14 shows the distribution of rules across the different mailboxes. On examining the rules, only 2.5% of the rules incorporate message body features, compared to 34.24% incorporating *Subject* features and 63.26% incorporating *From* features. This is reflected in the results obtained when certain features are omitted. Little change occurs when message body features are omitted, while a significant drop is observed in the number of predictions made when *From* features are omitted.

The feature extraction mechanism extracts a small number of features from the *From* field, while a similar numbers of features are extracted from both the *Subject* field and message body. There is a difference between the type of features extracted from the *Subject* field and message body however. Of all the features extracted by the feature extraction module, 10.11% are extracted from the *Subject* field compared to 72.00% extracted from the message body. With the smaller number of features, CN2 is able to induce rules which cover a greater number of examples than the rules which employ message body features, i.e. *Subject* features are used to cover most of the examples, and those examples not covered by these features are covered by features from the message body.

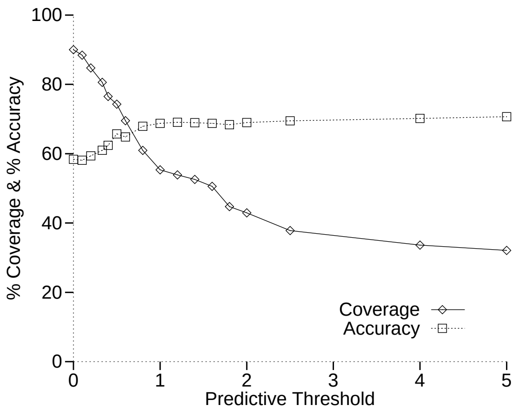



Figure 17: Comparing Changes in Predictions Made with Respect to the Predictive Threshold.

Table 2 shows how the percentage of features generated from the *From* field compares to the number of rules induced by CN2 for these features. The small number of *From* features appear more frequently in training examples than other features (see Figure 3). This may bias the algorithm, resulting in a higher number of rules which incorporate them. This effect becomes more pronounced when features from other fields are omitted.

### 4.2.2 Examination of coverage vs accuracy

Many of the results indicate a trade-off between the number of predictions and the accuracy of these predictions. By increasing the lower bound *predictive threshold*, we would expect the number of predictions made to decrease, but the accuracy of the remaining predictions would increase. Figure 17 shows the results of varying the *predictive threshold* on the number of predictions made

| **Fields used to induce rules with CN2** | **% of all features which are *From* features** | **% of induced rules containing *From* features** |
|---|---|---|
| *From, Subject, Message Body* | 17.89% | 63.26% |
| *From, Subject* | 32.64% | 72.53% |
| *From, Message Body* | 12.27% | 82.61% |

Table 2: Proportion of Features Extracted and Rules Induced from the *From* field.

and their accuracy after training IBPL1 with 60% of the data.

These results show that as the threshold increases, there is an initial drop in the number of predictions and a resulting increase in accuracy. However, the accuracy of predictions appears to stabilise at about 68%. This indicates that some incorrect predictions are made with a high *confidence rating*. For each of the individual mailboxes, this leveling out occurs at different thresholds; values between 0.6 and 1.4 have been observed for the mailboxes described here.

These results support the use of variable upper thresholds. It is very difficult to predict actions for some mailboxes, such as *cfp*, yet every *kdd* message can be correctly predicted. By using an upper threshold (the *confidence threshold*) for each different action, the accuracy of predictions for messages from different mailboxes can be improved. Figure 18 shows the effect of incorporating these thresholds on prediction accuracies for messages from each mailbox.

## 5 Discussion

The purpose of this study was to investigate issues related to the application of machine learning algorithms to interface agents. Tests using two learning algorithms across a number of mailboxes have shown that the performance of the algorithms vary with respect to their ability to accurately predict user actions for incoming messages. However, the characteristics of different message types vary, so whilst one learning approach performs better with certain types of messages, another approach may perform better with other types of message. For example, the performance on messages from the *phd_stuff* mailbox was superior when using IBPL1, yet predictions for messages from the other mailboxes was in general superior when using CN2. The different learning approaches also appear to differ with respect to the types of features they depend on. As a result, it is difficult to state any definitive strategy for extracting good features for any specific learning algorithm.

The results indicate that the *From* and *Subject* fields help to reduce the number of incorrect predictions made, depending on the learning algorithm used. For example, let us consider the changes in performance for each algorithm when the *From* features are used: with CN2, the number of accurate predictions made increases; however, with IBPL1, the number of accurate predictions appears to be constant, but the number of incorrect predictions made is reduced. The use of a dictionary with IBPL1 also appears to reduce the number of incorrect predictions.

One important difference between the two algorithms is the time taken to induce and apply user profiles to new mail messages. The instance based approach builds a sub-symbolic representation in the form of weights and distance metrics. Unlike rule induction in CN2, these calculations do not involve searching through a large space of possible solutions. The search performed by CN2 is compounded by the large number of features generated by the message body. It was found that tests involving CN2 took significantly (30 to 40 times) longer than tests involving IBPL1.

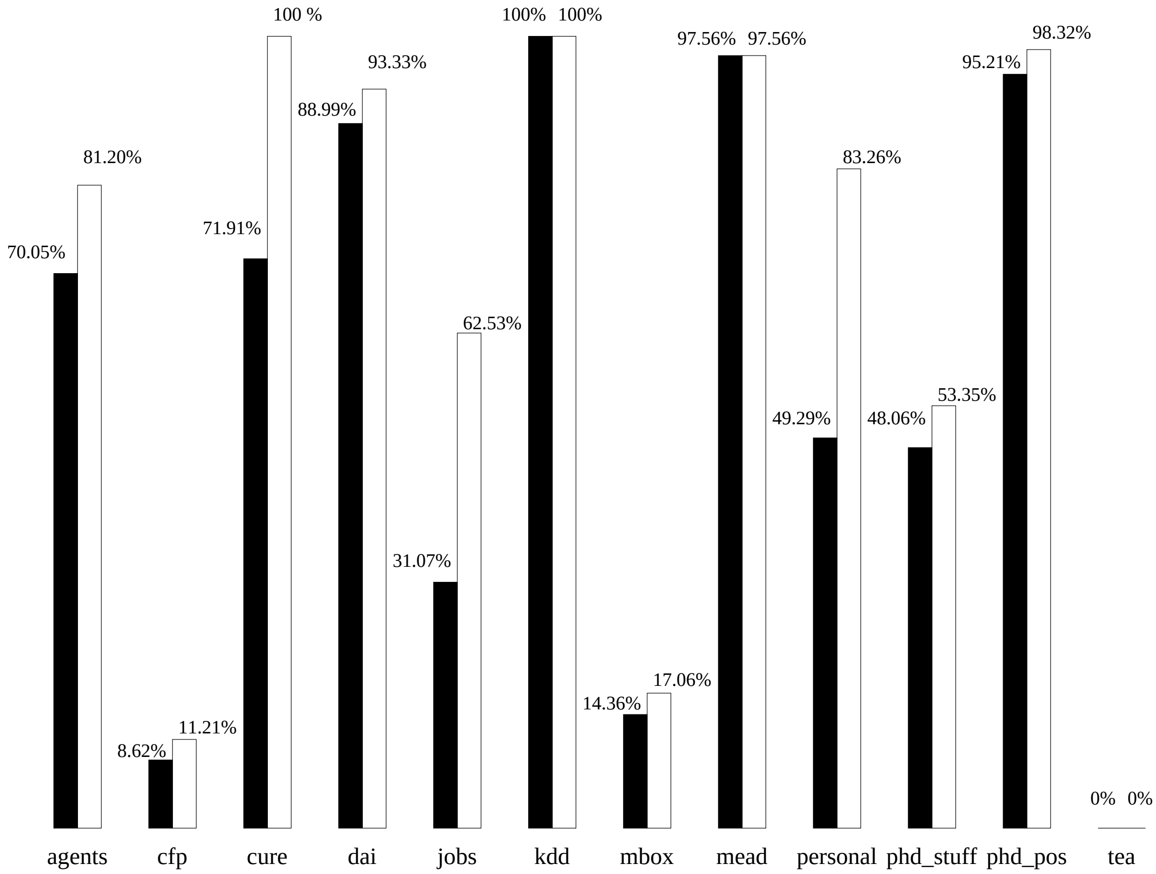



Figure 18: The effect of incorporating confidence thresholds on the accuracy of predictions. The measurements in black indicate the accuracy of predictions made by the prediction module. The white measurements indicate the improved accuracy due to confidence thresholds being used.

Both algorithms suffered from the nature of the data which contains large numbers of infrequently occurring features. The feature extraction approach considered each mail message in isolation, and extracted the most frequently occurring features from that message. This was sometimes at the expense of features occurring less frequently within individual messages, but which occur in many messages. Such features appear to be needed in order for messages to be recognised.

The results do show the importance of generating some level of confidence for each prediction. The agent may be poor at recognising certain message types yet good at others. The user needs to be reassured that few or no incorrect predictions will be made on messages which result in critical actions, such as deletion, etc. Hence, there is a need for individual thresholds for different message types and action classes. Whilst this may result in a decrease in the number of predictions made for certain message types, it attempts to reduce the number of incorrect predictions made.

## 6 Related Work

Existing interface agents can be compared across several different dimensions:

**Autonomy** How transparent are the activities of the agent to the user? Are they *proactive* learners in that they ask questions whilst learning, or do they silently observe and learn from the user?

**User Control** How intrusive is the assistance provided by the agent? Can it be ignored or overruled?

**Feature Identification** What features are used by the agent in determining how to assist the user?

**Learning** What learning methods are used? Does learning occur in real time or periodically?

**Adaptability** Can the agent adjust to changes in user requirements over time?

Our mail agent interface (Magi) is autonomous and remains completely transparent to the user. A *user profile* is learned by silently observing the user categorising and filing mail messages. Features are extracted from the mail messages by applying a standard word frequency algorithm to the message body and using the *From* and *Subject* fields in the mail header. The two learning algorithms compared in our study differ in how they maintain and update their user profiles; CN2 induces new production rules periodically which are then used in making predictions for incoming messages; IBPL1 generates weights and distance metrics from the training data each time new messages are to be classified. Both methods prune the training set regularly by removing old examples. This way, the agent learns to adapt to the user's changing classification criteria. The feature extraction mechanism is also used to identify features from incoming messages. These features are then tested against the production rules or compared with past instances (for CN2 and IBPL1 respectively). Two types of predictions are made: those which need to be confirmed by the user before being performed and those which will be performed without confirmation. However, none of the actions will be invoked by the agent on the new messages until the user has instructed the agent to do so. Thus the user can choose to ignore the agent's predictions or can accept them, or browse them.

NewT (News Tailor) is a USENET news reader (Sheth 1994). In adopting a genetic algorithm based approach (De Jong 1988), this agent can adapt to a user's preferences in reading news articles. Keywords within certain fields (e.g. the subject field) are used to identify articles of interest in a news group.

The system maintains a set of agents, each responsible for exploring a different area of interest, such as sport or business. An agent consists of a number of *profiles*, a profile being a set of tuples containing a field and a weighting value. A profile has a rating, or *fitness* value, which measures how effective the profile is in identifying articles of interest. These profiles are used to identify articles within newsgroups; the names of the newsgroups explored are also maintained by the agent. New agents can be created by the user, and new profiles generated to represent the user's interests. When the user reads a new article, he/she can indicate if it was of interest. This feedback is used to adjust the fitness of the profiles; positive feedback increases this value, whilst negative feedback decreases it. The user can also provide extra feedback by indicating words in the body of the article which are believed to be significant.

By using two genetic processes, *Crossover* and *Mutation*, the system can explore the search space for better solutions (Sheth and Maes 1993). *Crossover* is a mechanism whereby new agents are created by exchanging parts of the profiles of two existing agents. This way, new keywords are used to detect articles of interest within existing newsgroups. To explore new news groups, *Mutation* is used. This creates a new agent from an existing agent by randomly selecting a news group similar to an existing one. If these new agents succeed in finding articles of interest, the fitness of their profiles will increase, otherwise they will decrease. Profiles with a low fitness are eliminated, so controlling the population.

To determine what articles are of interest, NewT periodically filters new articles. This is done by converting articles into their vector space representations (Salton and McGill 1983). Each article is tested against the population of profiles for each agent, and ranked according to the closeness of the match. The highest ranking articles are then presented to the user. The user provides positive or negative feedback as the articles are browsed, which is then reflected by the change in the fitness of a profile. The user can also browse through all the unfiltered newsgroups.

A different approach to learning within an interface was address by Kozierok in her development of a Meeting Scheduling Agent (Maes and Kozierok 1993). This agent uses a variety of techniques for learning: learning by observation, learning from user feedback, and learning from being trained (Kozierok and Maes 1993). The learning mechanism, *Memory-Based Reasoning* (Stanfill and Waltz 1986) is used in conjunction with *priority weightings*. These priority weightings are determined by a reinforcement learning process which modifies them according to user feedback. When the agent makes an incorrect prediction, the user is given an explanation as to why the agent behaved as it did and is given an opportunity to indicate if this is because of incorrectly set weightings (such as too little importance given to a certain field). For example, the agent may ask: "...was the meeting topic more important than I thought?...", and then adjust the priority weights for the keywords found in the meeting description accordingly.

As the agent learns or is trained, the priority weightings are modified and the memory of examples is updated. However, Memory-Based Reasoning utilises weights and distance metrics to determine the nearest neighbours of a test example. As these values are only re-calculated periodically (e.g. each evening), the new examples will not affect the performance of the learning algorithm until then.

New meetings are arranged by the user completing a form. Email messages are then sent to the participants of the meeting. Each participant can enter into negotiation regarding the meeting via their calendar managers. Each agent attempts to predict their user's likely response to the meeting request and so automate the negotiation process. This is done by comparing the current negotiation episode with previous episodes. If a similar match is found, a confidence rating is generated with the prediction, which determines how the agent will react. Two thresholds are used to determine this: a *tell-me* threshold and a *do-it* threshold. In addition to this, a caricature is displayed to indicate when a negotiation episode is taking place and the confidence the agent has in its prediction. The caricature is in the form of a face, with different expressions to indicate the agent's confidence in negotiating on behalf of the user. If the rating falls below the *tell-me* threshold, the caricature will appear "unsure" and present the user with the details of the meeting request. If the rating falls between the two thresholds, the prediction will be displayed with the meeting details, and the caricature will suggest to the user a reply to the meeting request. Depending on whether the user confirms or rejects the prediction, the agent will question the user to determine what factors were important in this response. This feedback is used in adjusting the priority weights. If the *confidence rating* is greater than the *do-it* threshold, the prediction will be automated, and a report of the agent's actions will be provided.

Metral (Metral 1993) adapted Kozierok's agent architecture, which had been applied to filing mail messages in the ChainMail (Drake 1993) application, and developed a generic mechanism whereby the agent could be attached to any application. In doing this, he utilised the AppleEvents protocol to provide a means of inter-process communication between the agent and an electronic mail application Eudora (Dorner 1988) running on an Apple Macintosh™. The ChainMail application made use of mail message header information, namely the `from`, `cc` and `subject` fields.

CAP (Calendar APprentice) is another personal learning apprentice which assists in managing a meeting calendar (Dent et al. 1992). Unlike the work by Kozierok, this calendar manager assists the user in completing meeting forms[3] by predicting values for certain fields. Built within the Emacs environment, the predicted values are suggested as default values. These defaults can be accepted by the user, or overwritten with preferred values.

CAP learns rules to propose values for the *duration, location, day-of-week* and *time* of meetings. CAP communicates with THEO (Mitchell et al. 1991) which provides the representation, inference and learning mechanisms upon which CAP was built. The agent was tested with two inductive learning methods provided by the THEO system, a variant of Quinlan's ID3 algorithm (Quinlan 1986) producing decision trees, and a connectionist algorithm using back-propagation (Rumelhart et al. 1986). Whilst empirical studies showed that both learning methods produced comparable results when trained with the same data, subsequent work concentrated on decision trees (Mitchell et al. 1994) for two reasons. Firstly, decision trees can generate production rules which are intelligible to human users, whereas the weights generated by back-propagation are difficult to interpret. Secondly, it was easier to monitor performance of individual rules over time; this would prove difficult with the connectionist approach.

Periodically, the learning process generates rules from a set of training events. This set consists of approximately 180 examples covering a period of approximately 2 months. This number was found to produce satisfactory results, allowing rules to reflect the current meeting preferences, which can change during the year. New rules are generated periodically, and between 10% and 50% of these were found to be new and interesting rules. This new rule set is then merged into the existing rule base. Duplicate rules are removed and the rules sorted, according to their performance. This ordering is important as advice is generated by the agent according to the first rule in the list which fires.

The system is autonomous, as the user is observed, and these observations used to generate new rules. Suggestions are made as the meeting forms are completed, and if the suggestions are ignored, this is used as feedback to reorder the rule-base. The resulting meeting is then used as a new training example for the system.

A related apprentice system addresses the task of assisting the user with repetitious form filling (Hermens and Schlimmer 1993). This system applies learning to a graphical form, such as a Personnel Leave Report Form. Each form is designed to be completed in a particular order, left to right and top to bottom. Each time the user selects the *Next* control button, the agent makes use of values from previous forms and values entered so far to predict values for the new form. These predicted values are entered into slots on the form that have yet to be filled by the user. Completed slots are not overwritten and if no prediction can be made for a slot, it will be left blank. As each new slot is completed, the agent revises its predictions for the remaining fields. This use of values from previous forms was found to increase performance and to be effective in learning sequences when the user completed repetitious, or sequential forms. Completed forms are

[3] A system is currently being developed which can make use of this learned knowledge to negotiate semi-autonomously via email on behalf of the user (Mitchell et al. 1994)

then used to provide new training examples.

Two learning algorithms were used in testing the system, an incremental variant of Quinlan's ID3 algorithm (Quinlan 1986) called ID4 (Schlimmer and Fisher 1986), and COBWEB (Fisher 1987), a conceptual clustering algorithm. Tests showed that ID4 dramatically reduced the number of keystrokes required to complete a form, by 87% on 269 forms processed. COBWEB did not perform as well, but still reduced keystrokes by 64%. However, if the forms were completed in a random order, simple methods (using the *most commonly used* value or *most recently used* values) performed better than the learning methods.

The agent could be described as an autonomous learner, as no specific training or explanations are required to use the system. Predictions are automatically inserted into the form, but these can easily be accepted or rejected by the user. All training examples are used to train the agent[4], however it is not clear whether the agent is trained immediately or periodically.

Other related systems which apply learning mechanisms to assist users include an interactive note-taking system (Schlimmer and Hermens 1993) for pen-based computers which attempts to predict user's input, and creates a custom user interface on request; learning apprentice systems such as LEAP (Mitchell et al. 1990) which observes and learns from users in their use of a VLSI design system; and DISCIPLE (Tecuci and Kodratoff 1990), a multistrategy, integrated learning apprentice which has been used in building a variety of practical expert systems.

## 7 Conclusion

A number of researchers have investigated the application of learning techniques in the context of interface agents. Many store training examples collected from the user over a given time, and these are used to induce new rules or weights periodically. Others identify key features and determine over time the quality of these features (as in NewT (Sheth 1994)) thus reducing the storage overhead. Techniques are emerging to identify good features within training data for learning (Aha et al. 1991), or to share information between agents (Lashkari et al. 1994).

The manner in which the learning component deals with features is also important. As shown in this study, the inclusion of certain information may result in an overall increase in the number of predictions or suggestions, with no change in accuracy. Alternatively, it may result in an overall increase in accuracy, by reducing the number of incorrect predictions.

Each of the agents discussed above addresses the issue of autonomy in a different way. Those that automate actions, such as filtering or organising meetings (Metral 1993; Maes and Kozierok 1993; Payne 1994) develop a trust relationship by informing the user about the actions which will be performed. Feedback is used to build up this trust, until a point is reached where the user is confident enough to trust the agent's predictions. For systems such as these, accuracy is very important. Unless there is trust in its predictions, the agent is unable to be of assistance, as the user will have to confirm each prediction before the agent can perform it. Other agents, such as those concerned with form filling (Dent et al. 1992; Hermens and Schlimmer 1993) make unintrusive suggestions to the user. The accuracy of these predictions is less important; if the suggestion is incorrect then the user simply ignores it and enters the correct value.

There is a growing need to identify the impact of interface agents that learn on day to day activities.

[4]The authors reported that at the time of writing (Hermens and Schlimmer 1993) the system had not been active for a long enough period to observe the effects of old and no longer relevant data.

Not only is it important to consider how well they perform within a working environment, but also to consider if they really do provide assistance without becoming a nuisance. Issues such as speed and size are also significant. Agent technology is now being applied to everyday tools, such as calendar managers, news readers and email filters. These issues have to be considered when designing new agent architectures or applications.

## 8 Acknowledgements

T.R. Payne acknowledges financial support provided by the UK Engineering & Physical Sciences Research Council (EPSRC). We gratefully acknowledge the authors and maintainers of Xmail, Michael C. Wagnitz & Jeff Markham, for allowing us to use their software. Xmail is copyright 1990,1991,1992 by National Semiconductor Corporation. We acknowledge Winton Davies for his original ideas regarding induction and mail messages. We also thank Ciara Byrne, Fraser & Zak Mitchell for their helpful comments on an earlier draft of this paper.

## References

Aha, D. (1991). Incremental Constructive Induction: An Instance-Based Approach. In *Proceedings of the 8th International Machine Learning Workshop*, pp. 117–121.

Aha, D., D. Kibler, and M. Albert (1991). Instance-Based Learning Algorithms. *Machine Learning 6*, 37–66.

Baclace, P. (1992). Competitive Agents for Information Filtering. *Communications of the ACM 35*(12), 50–51.

Chin, D. N. (1991). Intelligent Interfaces As Agents. In J. W. Sullivan and S. W. Tyler (Eds.), *Intelligent User Interfaces*, pp. 177–206. New York, New York:ACM Press.

Clark, P. and T. Niblett (1989). The CN2 Induction Algorithm. *Machine Learning 3*, 261–283.

Cohen, P., A. Cheyer, M. Wang, and S. Baeg (1994). An Open Agent Architecture. In *Software Agents: Papers from the 1994 Spring Symposium*, pp. 1–8. Menlo Park, CA:AAAI Press.

Cover, T. (1968). Estimation by the Nearest Neighbor Rule. *IEEE Transactions on Information Theory 14*(1), 50–55.

Cover, T. and P. Hart (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory 13*, 21–27.

Davies, W. and P. Edwards (1994). Agent-K: An Integration of AOP and KQML. In Y. Labrou and T. Finin (Eds.), *CIKM'94 Intelligent Agents Workshop.*

De Jong, K. (1988). Learning with Genetic Algorithms: An Overview. *Machine Learning 2*, 121–138.

Dent, L., J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski (1992). A Personal Learning Apprentice. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pp. 96–103.

Dorner (1988). Eudora: Bringing the P.O. Where You Live. Qualcomm Inc.

Drake (1993). ChainMail. BSc Thesis, Department of Electrical Engineering and Computer Science, MIT.

Etzioni, O., N. Lesh, and R. Segal (1994). Building Softbots for UNIX (Preliminary Report). In *Software Agents: Papers from the 1994 Spring Symposium*, pp. 9–16. Menlo Park, CA:AAAI Press.

Fisher, D. (1987). Knowledge Acquisition Via Incremental Concept Clustering. *Machine Learning 2*, 139–172.

Foltz, P. and S. Dumais (1992). Personalized Information Delivery: An Analysis of Information Filtering Methods. *Communications of the ACM 35*(12), 51–60.

Gil, Y. (1994). Trainable Software Agents. In *Software Agents: Papers from the 1994 Spring Symposium*, pp. 99–102. Menlo Park, CA:AAAI Press.

Hermens, L. and J. Schlimmer (1993). A Machine-Learning Apprentice for the Completion of Repetitive Forms. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, pp. 164–170. Orlando, Florida: IEEE Computer Society Press.

Jacobs, P. and L. Rau (1990). SCISOR: Extracting Information from On-Line News. *Communications of the ACM 33*(11), 88–97.

Jourdan, J., L. Dent, J. McDermott, T. Mitchell, and D. Zabowski (1993). Interfaces That Learn: A Learning Apprentice for Calendar Management. In S. Minton (Ed.), *Machine Learning Methods for Planning*, pp. 31–67. San Mateo, CA: Morgan Kaufmann.

Kay, A. (1984). Computer Software. *Scientific American 251*, 41–47.

Kozierok, R. and P. Maes (1993). A Learning Interface Agent for Scheduling Meetings. In *Proceedings of the ACM-SIGCHI International Workshop on Intelligent User Interfaces*, pp. 81–88. New York, New York:ACM Press.

Langley, P. and W. Iba (1993). Average Case Analysis of a Nearest Neighbor Algorithm. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, IJCAI-93*, pp. 889–894.

Lashkari, Y., M. Metral, and P. Maes (1994). Collaborative Interface Agents. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 444–449.

Lewis, D. (1991). Learning in Intelligent Information Retrieval. In *Proceedings of the 8th International Machine Learning Workshop*, pp. 235–239.

Maes, P. (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM 37*(7), 30–40.

Maes, P. and R. Kozierok (1993). Learning Interface Agents. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pp. 459–465.

Malone, T., K. Grant, F. Turbak, S. Brobst, and M. Cohen (1987). Intelligent Information-Sharing Systems. *Communications of the ACM 30*(5), 390–402.

Metral, M. (1993). Design of a Generic Learning Interface Agent. BSc Thesis, Department of Electrical Engineering and Computer Science, MIT.

Miller, M. and K. Drexler (1988). Markets and Computation: Agoric Open Systems. In B. Huberman (Ed.), *The Ecology of Computation*, pp. 133–176. North Holland: Elsevier Science Publishers.

Mitchell, T., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, R. Ringuette, and J. Schlimmer (1991). Theo: A Framework for Self-Improving Systems. In *Architectures for Intelligence, The 22nd Carnegie Mellon Symposium on Cognition*, pp. 323–357. Lawrence Erlbaum Associates.

Mitchell, T., R. Caruana, D. Freitag, J. McDermott, and D. Zabowski (1994). Experience with a Learning Personal Assistant. *Communications of the ACM 37*(7), 81–91.

Mitchell, T., S. Mabadevan, and L. Steinberg (1990). LEAP: A Learning Apprentice for VLSI Design. In Y. Kodratoff and R. Michalski (Eds.), *Machine Learning, An AI Approach Volume III*, pp. 290–301. San Mateo, CA: Morgan Kaufmann.

Nessen, E. (1989). SC-UM: User Modeling in the SINIX Consultant. *Applied Artificial Intelligence 3*, 33–44.

Payne, T. (1994). Learning Email Filtering Rules with Magi, A Mail Agent Interface. MSc Thesis, Department of Computing Science, University of Aberdeen, Scotland.

Payne, T., P. Edwards, and C. Green (1995). Experience with Rule Induction and *k*-Nearest Neighbour Methods for Interface Agents that Learn. In *ML95 Workshop on Agents that Learn from Other Agents.*

Quinlan, J. (1986). Induction of Decision Trees. *Machine Learning 1*, 81–106.

Ram, A. (1992). Natural Language Understanding for Information-Filtering Systems. *Communications of the ACM 35*(12), 80–81.

Rumelhart, D., G. Hinton, and R. Williams (1986). Learning Internal Representations by Error Propagation. In D. Rumelhart and J. McClelland (Eds.), *Parallel Distributed Processing*, Volume 1, pp. 318–362. Cambridge, MA:MIT Press.

Salton, G. and M. McGill (1983). *Introduction to Modern Information Retrieval.* New York: McGraw-Hill.

Schlimmer, J. and D. Fisher (1986). A Case Study of Incremental Concept Induction. In *Proceedings of the 5th National Conference of Artificial Intelligence*, pp. 496–501.

Schlimmer, J. and L. Hermens (1993). Software Agents: Completing Patterns and Constructing User Interfaces. *Journal of Artificial Intelligence Research 1*, 61–89.

Sheth, B. (1994). A Learning Approach to Personalized Information Filtering. Master's Thesis, Department of Electrical Engineering and Computer Science, MIT.

Sheth, B. and P. Maes (1993). Evolving Agents For Personalized Information Filtering. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, pp. 345–351. Orlando, Florida: IEEE Computer Society Press.

Shoham, Y. (1993). Agent-Oriented Programming. *Artificial Intelligence Journal 60*, 51–92.

Stanfill, C. and D. Waltz (1986). Toward Memory-Based Reasoning. *Communications of the ACM 29*(12), 1213–1228.

Tan, M. and J. Schlimmer (1990). Two Case Studies in Cost-Sensitive Concept Acquisition. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pp. 854–860.

Tecuci, G. and Y. Kodratoff (1990). Apprenticeship Learning in Imperfect Domain Theories. In Y. Kodratoff and R. Michalski (Eds.), *Machine Learning, An AI Approach Volume III*, pp. 514–551. San Mateo, CA: Morgan Kaufmann.

Thomas, S. (1993). *PLACA, an Agent Oriented Programming Language.* Ph. D. thesis, Computer Science Department, Stanford University.

Wayner, P. (1994, May). Agents Away. *Byte*, 105–114.