Learning Mechanisms for Information Filtering Agents

Terry R. Payne & Peter Edwards
Department of Computing Science
King's College
University of Aberdeen
Aberdeen, Scotland, AB9 2UE
{terry, pedwards}@csd.abdn.ac.uk

Abstract

In recent years, software agents have been developed which assist users with tasks such as information filtering or information retrieval. Such systems have evolved from simple agents that refer to a user-defined script to filter incoming mail, to complex Web agents that not only learn their user's preferences but actively seek out Web pages that could be of interest. To provide personal assistance, an agent needs information about the user's interests and needs. This paper reviews how different mechanisms have been used to define a *user profile*, from simple rules to complex machine learning algorithms. Problems with user-defined scripts are discussed, as are the issues involved with integrating learning mechanisms into agents. One approach currently being developed to learn within an agent environment is then described.

1 Introduction

A software agent[1] can be characterised as a system which aids and assists a user with some common task. It may employ some degree of learning or adaptation to improve the quality of its assistance over time; be autonomous, e.g. the agent can learn without needing to be explicitly trained; offer assistance only when requested, or act independently of the user; be mobile, travelling in search of useful information or to fulfill tasks for the user; interoperate, i.e. exchange information and services with other agents, to provide greater assistance to the user than if the agent worked in isolation [14]. Described as:

> "...computer programs which employ Artificial Intelligence techniques to provide assistance to a user dealing with a particular computer application..." *Pattie Maes* [21],

many systems have been developed to deal with filtering information, such as electronic mail or USENET news articles [32, 27]. Other systems have been developed which actively seek out information [34, 4], and the number of such systems being developed is increasing. With more information becoming available on

[1] For a survey on agent theories and architectures, see Wooldridge & Jennings [37].

the Internet daily, searching for interesting or relevant information is becoming increasingly difficult [32]. Hence, there is a definite need for agents which can assist users with such tasks.

If an agent is to be of assistance, it requires knowledge about the domain application and/or the user. Two approaches have traditionally been used to provide an agent with knowledge about its task domain. The first and most common method is for users to provide their own rules, e.g. using a scripting language. Some systems provide an environment to allow users to simulate the behaviour of rules and to test them with hypothetical messages [8]. The second method makes use of traditional knowledge engineering techniques to identify background knowledge about the application and the user. This technique has been applied to advisory agents, such as UCEgo [6], which provides advice on using the UNIX operating system. 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, approaches such as this cannot be used for personalised tasks, such as information filtering.

An alternative is to acquire knowledge about the user through learning. Many agents employ machine learning techniques to induce a *user profile*. This not only eliminates the need for users to define their own scripts, but allows the agent to adapt to changes. To date, many different learning mechanisms have been employed within agents. Genetic algorithm approaches [13] have been used for news filtering agents [32]; decision tree algorithms such as C4.5 [29] and CN2 [7], and instance-based approaches [33] have been used with mail and USENET news filtering tasks [23, 26, 27] and automated Web browsing [4, 17]. Minimum Description Length techniques have been explored in USENET news filtering [19], and relational learning algorithms such as FOIL [28] have been applied to text categorisation [9]. However, it is difficult to evaluate the relative performance of the techniques employed by learning agents, even within the same domain (such as news filtering). Evaluations performed on various agent systems to date have relied on individually constructed datasets. No standard datasets yet exist within the UCI Machine Learning Data Repository [24] to evaluate such systems[2]. However, some studies [26, 19] have made comparisons of learning techniques across data gathered from existing agent-based systems.

2 Interface Agents

The evolution of knowledge acquisition techniques for interface agents can be illustrated by examining the issues raised by various information filtering/retrieval systems.

2.1 Early Interface Agents

Early interface agents relied on users writing scripts to describe their interests or preferences. Users of systems such as the Information Lens [22] had to define a set of rules which were used to filter and sort incoming mail messages. The Tapestry System [15] is an example of a collaborative mail filtering agent which provides a database query language, so that users can retrieve popular or interesting mail articles. Other systems rely on user-defined scripts which contain short programs, such as those employed by the Information Retrieval Agent (*IRA*) [34]. The use of scripts highlights a number of important issues [9]. Learning and utilising a scripting language may discourage non-technical users from using the system. As

[2]USENET news data gathered for a recent study [16] is currently being prepared for submission to the UCI repository.

well as understanding exactly how they require the system to behave, a user must appreciate how the agent will perform with the script. For example, the behaviour of individual rules may differ when used in isolation and when used with other rules. The user also has to be responsible for maintaining the scripts over time as their interests change.

An alternative to using a user-defined script, is to build *user profile* of the user's preferences and requirements, by either querying the user directly or by passive observation of the user. Letizia [20] is one system which observes the user as he/she browses the Web. By applying a set of heuristics to these observations, the browser can recommend links that it believes the user will find interesting. Defining such heuristics requires an understanding of how users interact with the agent. For example, Letizia assumes that the user frequently visits pages that are of interest. Whilst for some pages this may be the case, others may simply be a starting point for exploration, such as those that contain search forms. Hence, the agent only performs well when the user behaves as anticipated, and cannot react if this behaviour changes.

2.2 Interface Agents & Machine Learning

Machine learning techniques have been investigated as a means of creating *user profiles*, and as reported above, many different learning algorithms have been applied to this problem. The user requires no prior knowledge about scripting languages, as the *user profile* is induced from their actions. The profile can also evolve as the user's behaviour changes. In order to induce this profile, mechanisms are needed to extract information for learning. Machine Learning algorithms represent their knowledge prior to learning as a set of *instances*. An instance contains a fixed number of *attributes*, each of which contains a single value. A feature extraction mechanism must be used to identify terms, such as keywords in news articles or mail messages, and map these to attributes for learning. Different techniques for identifying terms have been used, such as information theoretic approaches [31], or user directed approaches [18].

The USENET news reader, NewT (News Tailor) [32] is one system which uses genetic algorithms [13] to manage a population of *profiles* which reflect the users interest in different topics. Each profile contains a set of tuples which comprise of grouped keywords and an associated weighting value. Only the highest ranking, or *fittest* profiles survive; to control the size of the population, those profiles with a low fitness are eliminated. In addition, two genetic operators, *Crossover* and *Mutation* can be used to search for better solutions. NewT uses *crossover* to create new profiles by exchanging the keywords of two existing profiles. *Mutation* is used to explore new newsgroups by applying the keywords in an existing profile to a different newsgroup.

The *Memory-Based Reasoning* (MBR) algorithm [33] has been employed within the MAXIMS mail filtering agent [23]. MBR is a member of the Instance-Based Learning family of algorithms. This approach stores sets of *exemplars* or instances and then compares these to new unclassified situations. Maxims stores instances which describe a situation (such as receiving a mail message) and the user's response to it (for example, deleting the message). A modified version of the MBR algorithm was used which incorporated *priority weightings* [18]. This enabled the agent to rate elements (such as different keywords) within exemplars by importance. When the user acts upon a message, such as deleting or reading it, keywords from the message and the user's action are stored as an exemplar. The agent then attempts to predict the user's likely response to the arrival of a new message, by comparing it with the exemplars. If a similar match is found, a confidence rating is generated with the prediction, which determines how the agent will react. If 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 message less important than I thought?...", and then adjusts the priority weights for the keywords found in the message accordingly.

Both NewT and MAXIMS addressed how a particular machine learning technique could be applied to information filtering and retrieval, and what additional support was needed from the user for the agent to succeed in its task. Work by Lang [19] on the news-filtering system, NewsWeeder, has studied how well machine learning techniques perform compared to traditional information filtering techniques. The performance of the *Minimum Description Length* principal (MDL), described in [19], was compared to *term-frequency/inverse-document frequency* weighting (*tfidf*) [31]. Results showed that machine learning techniques could outperform information filtering techniques for identifying news articles of interest.

2.3 Experiences with Learning Agents

To explore the issues involved in applying machine learning to information filtering agents, we have designed an agent architecture [26] and have used this to implement Magi [25], UNA [16] and LAW [4]. The instantiation of this architecture for the Magi system is shown in Figure 1.

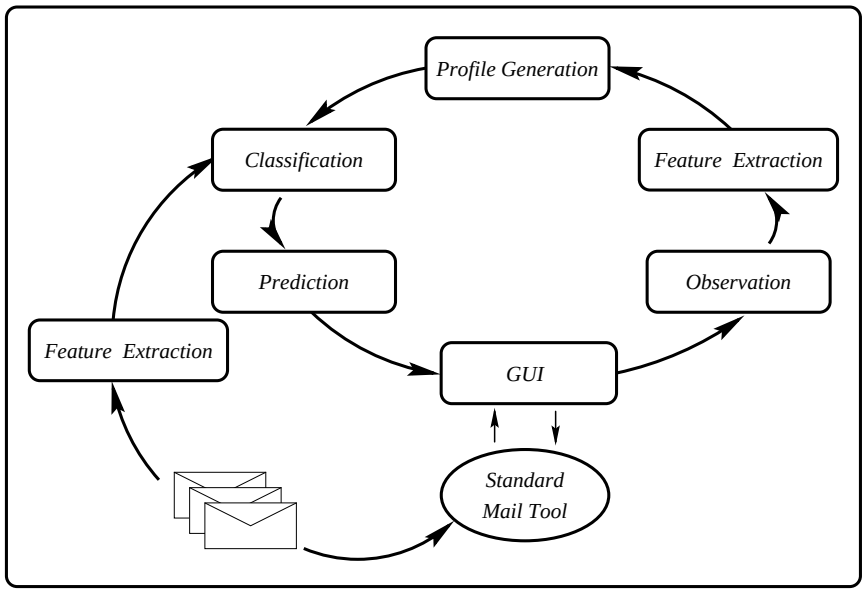

Figure 1: The Magi Architecture.

The work on Magi (Mail Agent Interface) explored the use of this architecture in developing an agent which aids a user in sorting incoming electronic mail [25]. A version of *Xmail*, a graphical user interface for mail, was modified to record *observations* of the user, i.e. to record what actions the user performed on his/her mail messages. Keywords, or *features*, were extracted from these observations, and used to generate a *user-profile*. A symbolic rule induction algorithm, CN2 [7], was used to induce the profile. Features were then extracted from incoming mail messages and tested by the classification engine. A classification was generated, which determined in which mailbox the message should be placed. The performance of this agent was compared to

that achieved when the instance based technique, IBPL1 [26] was used instead of CN2. Coverage (i.e. how many new messages could be classified) and accuracy (i.e. whether the classifications made were correct) were recorded for the system. A total of 408 mail messages were used in the evaluation, sorted into 12 mail boxes. The results are described in [26].

UNA (USENET News Agent) applied the agent architecture to the task of identifying interesting USENET news articles. A version of the *xrn* news browser was modified so that users could indicate their level of interest (on a scale of 1 - 6) in the articles read. A similar feature extraction mechanism to that developed for Magi was used, and the performance of *user-profiles* generated by both CN2 and IBPL1 were compared. This work is described in [17].

LAW (Learning Agent for the Web) extended this work by addressing some of the issues involved in feature extraction. Like its predecessors, LAW observed the user as he/she browsed the World-Wide Web, using a modified version of the web browser, *Chimera*. These observations were then used to induce the *user-profile*, using IBPL1 or C4.5 [29]. The profile was used to identify interesting Web pages. This was done in two ways: by highlighting links believed to point to pages of interest, and by using the profile as input to an autonomous Web robot. In addition, four information retrieval techniques (*term frequency*, *tfidf*, *signal to noise*, and *term relevance*) were compared to investigate which method extracted the best features for inducing the *user-profile*. See [4] for a full description of the results obtained, and [17] for a discussion of the practicality of this agent.

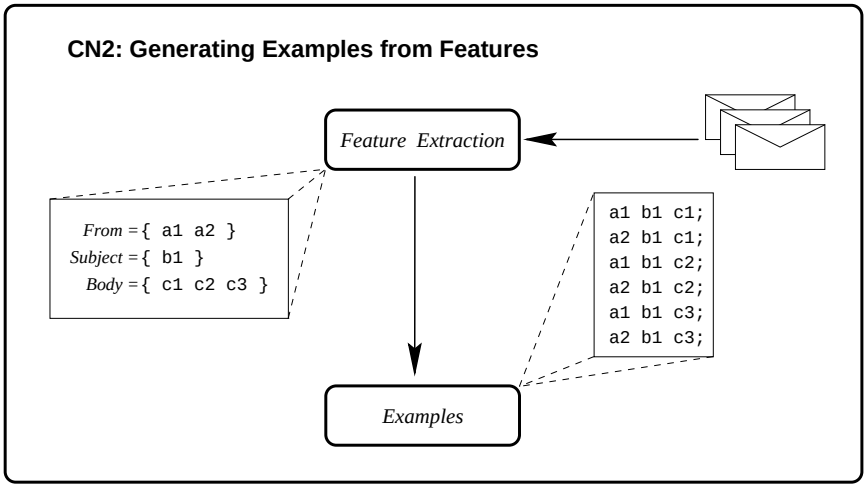

Figure 2: Generating Multiple Examples from Feature Sets for Magi.

From our work on these systems, it is clear that there are a number of important issues relating to agent-based information filtering which need to be explored. Our work on Magi [26] highlighted the problems associated with extraction of multiple features from each mail message. The rule induction algorithm used in that study, CN2 [7], expected single values for each attribute. This required all possible permutations of values to be created from the original feature sets, which resulted in large numbers of instances being generated for each training example. For example in Figure 2, six instances are generated from features extracted from a single mail message. Not only did the algorithm take longer to induce the rule set from these examples, but it biased the rule set towards values in feature sets with fewer values, such as the *Subject*

set in Figure 2.

The work on UNA highlighted the need for learning algorithms that learn graded concepts. For example, UNA represents the graded concepts *dull* and *interesting* by rating the article on a scale of 1-6. A rating of 1 indicated that the user found the article extremely *dull* or uninteresting, whilst a rating of 6 indicated that the user found it highly *interesting*. If two articles are rated 5 and 6 respectively, they may share many features which indicate that the article is of interest. However, nothing can be learned from features shared by widely differing articles (i.e. rated 1 and 6 respectively), as such features characterise both *dull* and *interesting* articles.

UNA also highlighted the need for alternative feature extraction techniques. The technique used by Magi identified features that characterised the mail message. However, this approach was unsuited to learning interest within a newsgroup. For example, in a sporting newsgroup (such as *rec.sport.orienteering*), the predominant features extracted from all the articles will be sporting terms, irrespective of whether the user found the article interesting or not. This leaves few features in the training data from which the concepts of *dull* and *interesting* can be learned. The work on feature extraction methods in LAW explored different ways of improving the feature extraction stage. This stage is crucial in providing quality training instances from which concepts can be learned.

3 Developing an Instance-Based Learning Algorithm for Information Filtering

The UNA and LAW systems [17] emphasised the need for good feature extraction techniques which generate quality training instances. Both systems consider feature extraction and learning as separate stages in the generation of the *user-profile*. Many agents (such as NewsWeeder [19] and Magi [25]) filter out the majority of features based on frequency or relevance measures, and retain only the most frequent or most relevant features. Other agents (e.g. NewT [32]) rely on the user to specify keywords from the text. The learning process then identifies which of these features best represent a concept or class. The problem with such approaches is that many of the features which are removed by the feature extraction stage may improve the concepts learned. By combining these two stages into one, features could be identified by the learning algorithm which act as good classifiers, and these used for learning.

One solution to this problem would be to create training instances that contain all the features. For example, for each mail message, Magi generates a vector of all the words and their frequency counts within the message. This vector can be expanded to contain every word encountered, and each word can be mapped to a single attribute in the training instance. This would result in instances whose size is of the order of 20 000 - 100 000 attributes [19]. The number of instances required to learn a concept increases as the number of attributes in each instance increases, and therefore many more instances would be needed to determine which attributes characterise the class. In addition, the time required to induce a *user-profile* from such instances could be so great that it would make this approach infeasible.

An alternative solution would be to utilise Prototypical Learning [5]. This type of learning has been explored by the Instance-Based Learning community as a means of reducing the amount of space required to store training instances. By identifying the most useful or significant instances stored, those that contribute little to future classifications can be removed. This approach could also be applied to sets of features. If the most

useful or significant features are identified within each set, those features that contribute little to future classifications can be removed.

3.1 Prototypical Learning and Instance-Based Learning

As mentioned earlier, Instance-Based Learning algorithms use specific instances rather than a pre-compiled abstraction when classifying new instances. Also known as *exemplar-based* [10] and *nearest-neighbour* methods [12], these techniques learn by storing instances as points in a feature space, and then calculating distances between a new instance and the existing exemplars. Distances can be calculated simply by using normalised Euclidean values if the instances contain numeric data. However, for symbolic data, distances may be calculated by considering the vector of overlapping features or using more complex statistical metrics. They have also been described as *Lazy Learning* algorithms because they store instances and postpone any form of generalisation until classification time. They can learn graded concepts, as shown by the Bloom algorithm [1], and provide a basis for exploring prototypical learning [5]. The use of multiple nearest neighbours, known as k-nearest neighbours (k-NN), has been shown in some cases to improve classification over methods that use the single nearest neighbour [11]. There is also work on the use of weighting methods [35, 36] to improve classification accuracy.

As instance-based algorithms store all the information held within each instance, noisy, redundant or irrelevant attributes are also preserved. Such algorithms therefore require large amounts of storage. Because of these factors, there has been a growing interest in prototypical learning. Much of the recent work in prototypical learning has emerged from psychological studies of humans. Rosch & Mervis [30] argue that natural concepts are represented by an *image* of the *prototype* of the concept. A prototype is an object that belongs to the concept and represents it best. Prototypicality can also be viewed as a graded property. In other words, as members of a concept get further away from the prototypical member, the less representative they are of the concept. Barsalou [3] argues that concepts in real world applications contain a number of graded instances. For example, when asked, American college students agreed that a robin was a typical bird, a pigeon was a moderately typical bird, and an ostrich was atypical. Zhang [38] developed a measure of *typicality* of an instance, i.e. how likely an example is to appear within one class, as opposed to other classes. This value was calculated in order to learn graded concept structures. The TIBL (Typical Instance-Based Learning) system was developed by applying this measurement to a standard instance-based learning algorithm, IB1 [2] to reduce the number of instances stored. When compared to other instance-based learning techniques, the results showed that with datasets which contained graded concepts, TIBL recorded lower storage requirements and higher classification accuracies than these other methods. Similar work has recently been carried out by Biberman [5].

3.2 The IBPL1 Algorithm

To explore the use of prototypical learning for information filtering agents, an instance-based learning algorithm was required. The Memory-Based Reasoning algorithm [33] was studied, as it had formed the basis for the learning component in a previous mail filtering system, MAXIMS [23]. We have developed a new instance-based algorithm, IBPL1, which is an extended version of MBR.

As described earlier, a large number of instances containing single values for each attribute were generated

when the CN2 algorithm was used within Magi (see Figure 2). 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 inter-instance distances within two respective feature sets (Figure 3) when calculating the distance between two instances. These inter-instance distances are then averaged to find an overall distance value.

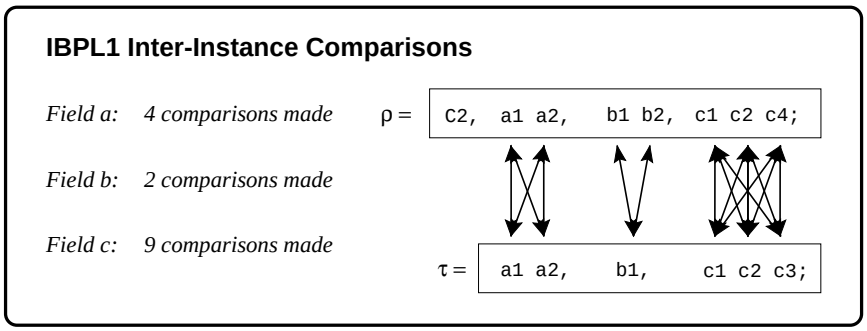

Figure 3: Calculating the Inter-Instance Distance.

Before the IBPL1 algorithm can be discussed in detail, some terms must be defined. Instances used in training, known as episodes or *exemplars*, and instances which are classified, known as *examples*, are denoted by the Greek letters ρ and τ respectively. These instances contain *fields*, such as *From* and *Subject* used in our mail filtering study. Each field contains a set of *features*, as shown in Figure 3. A field f in an instance ρ 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 ρ or example τ. 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 τ and a memorised exemplar ρ 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 ρ (Eq. 1).

$$\Delta_g(\mathcal{D}, \tau, \rho) = \sum_{f \in P_\rho} \delta_f^g(\mathcal{D}, \tau.f, \rho.f) \tag{1}$$

The partial distance ($\delta_f^g(\mathcal{D}, \tau.f, \rho.f)$) (Eq. 2) 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. $w_i^g(\mathcal{D}, \tau.f.i)$ defined in Equation 3 and $d_{ij}^g(\mathcal{D}, \tau.f.i, \rho.f.j)$ defined in Equation 4. 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 metric is used to determine the quality of a feature as a good classifier.

$$\delta_f^g(\mathcal{D}, \tau.f, \rho.f) = \frac{\sum_{i \in I_{\tau.f}} \sum_{j \in J_{\rho.f}} d_{ij}^g(\mathcal{D}, \tau.f.i, \rho.f.j) w_i^g(\mathcal{D}, \tau.f.i)}{|I_{\tau.f}| \times |J_{\rho.f}|} \tag{2}$$

$$w_i^g(\mathcal{D}, \tau.f.i) = \sqrt{\sum_{v \in \mathcal{V}_g} \left(\frac{|\mathcal{D}[i = \tau.f.i][g = v]|}{|\mathcal{D}[i = \tau.f.i]|} \right)^2} \tag{3}$$

$$d_{ij}^g(\mathcal{D}, \tau.f.i, \rho.f.j) = \sum_{v \in \mathcal{V}_g} \left(\frac{|\mathcal{D}[i = \tau.f.i][g = v]|}{|\mathcal{D}[i = \tau.f.i]|} - \frac{|\mathcal{D}[j = \rho.f.j][g = v]|}{|\mathcal{D}[j = \rho.f.j]|} \right)^2 \tag{4}$$

Once the distances Δ^g (Eq. 1) 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[3] of the distance Δ^g (Eq. 5). The top k exemplars with a classification κ are then summed for each κ. (Eq. 6). The classification κ 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 (κ), and the confidence in other classifications ($\neg\kappa$), as shown in (Eq. 7).

$$Score^g(\mathcal{D}, \tau, \rho) = \frac{1}{\Delta^g(\mathcal{D}, \tau, \rho) + 0.01} \tag{5}$$

$$Confidence^{v \in \mathcal{V}_g}(\tau) = \sum Score^v(\mathcal{D}, \tau, \rho) \tag{6}$$

$$Confidence_\kappa^g(\tau) = Confidence^{[g=\kappa]}(\mathcal{D}, \tau, \rho) - \sum_{v \in \neg\kappa} Confidence^{[g=v]}(\mathcal{D}, \tau, \rho) \tag{7}$$

3.3 The IBPL2 Algorithm

As IBPL1 was developed, many issues were raised, such as how unknown values should be handled, and how the algorithm could overcome noise. However, the concept of learning with sets was new, and it introduced additional problems, such as how comparisons should be made between a set of values and either its superset or subset (e.g. comparing the sets $\{a, b, c, d\}$ and $\{b, c\}$). Alternative methods for calculating distances are currently being considered. In IBPL1, the distances between inter-instance attribute values are calculated and averaged. Because of this, two identical instances will not be found to be identical (i.e. will not have a distance of zero) due to the effects of averaging all the distances. This can be overcome by only considering the *closest value* distance. The inter-instance calculation shown in Equation 2 has been modified so that only the smallest (i.e. the closest) distance between the value $\tau.f.i$ in the example and each of the values in the feature set $\rho.f$ in the exemplar are considered when finding the average distance between instances (Eq. 8). This new calculation forms the basis for a new version of the algorithm, IBPL2.

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

$$\delta_f^g(\mathcal{D}, \tau.f, \rho.f) = \frac{\sum_{i \in I_{\tau.f}} Min_{j \in J_{\rho.f}} \left(d_{ij}^g(\mathcal{D}, \tau.f.i, \rho.f.j) w_i^g(\mathcal{D}, \tau.f.i) \right)}{|I_{\tau.f}|} \quad (8)$$

Our work so far on IBPL1 and IBPL2 has concentrated on learning concepts from sets of values. Work will begin shortly to explore the use of the weighting metric (Eq. 3) as a means of identifying words (terms) which act as poor classifiers. This value reflects the *typicality* of a term, i.e. how likely the term is to occur in one class but not in others. Terms which appear with equal frequency in all classes have a low weighting value, whereas those that mostly appear in a single class have a higher value. These low weighted terms can be identified and removed from the feature sets in the stored examples prior to classification. This will result in a smaller number of values in the feature sets during classification, and hence fewer distance calculations will be required to identify the nearest neighbour.

3.4 An Evaluation of IBPL1/IBPL2

To date, IBPL1 has been used in three agent-based systems: Magi, UNA and LAW. For each system, comparisons were made between IBPL1 and a rule induction algorithm. The results from Magi indicated that the overall accuracy of predictions made by IBPL1 was slightly lower (57%) than those for CN2 (65%), although the results for individual mail boxes varied. IBPL1 was found to perform badly on very small mail boxes, typically those containing less than 10 messages. However, this can be explained by considering the voting strategy used by IBPL1. The closest k exemplars are considered when determining the classification. In these tests, k was set to 10. Cover [11] demonstrated that a larger value of k results in an improvement in the behaviour of large samples, at the expense of small sample behaviour. IBPL1 was also found to be significantly faster than CN2. This was due to the length of time taken by CN2 to induce rules from the large number of permutations of training instances.

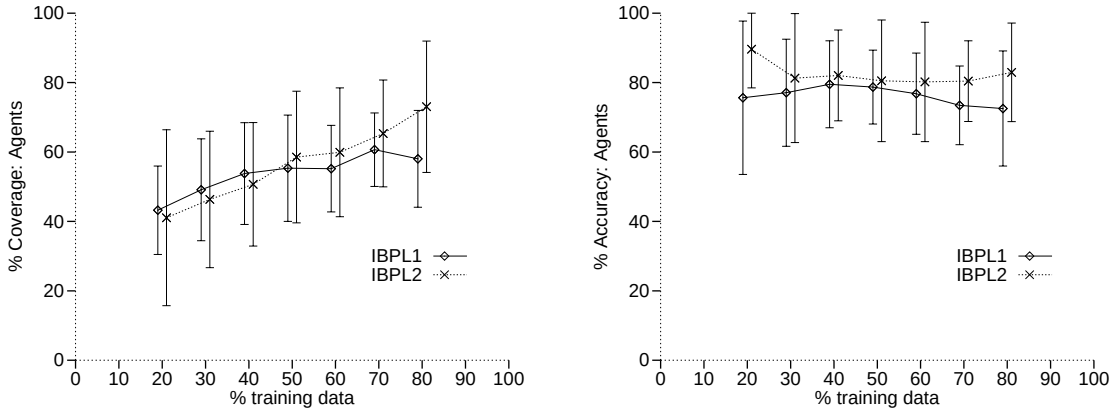

Figure 4: Comparing IBPL1 & IBPL2 as the Learning Component within Magi.

The results obtained for UNA were lower than expected. It is believed this was due to the type of features that were extracted from each news article (see above). Comparison of IBPL1 and C4.5 within LAW showed that the performance of both algorithms was similar. These results demonstrate that IBPL1 can learn *user-profiles* for mail filtering and Web browsing. A full discussion of the results from tests on Magi can be found

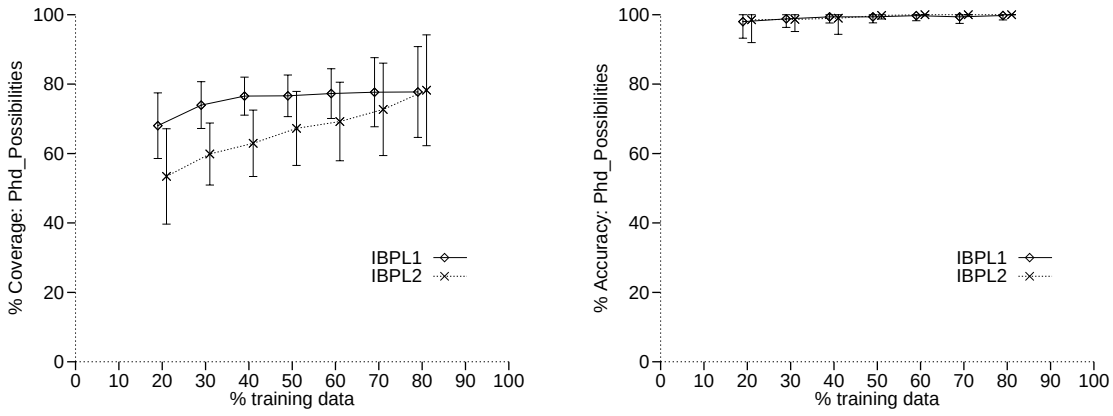

Figure 5: Comparing IBPL1 & IBPL2 as the Learning Component within Magi.

in [26].

Preliminary studies have been carried out with IBPL2, to investigate alternative methods of calculating distances. So far, it has been compared to IBPL1 for classifying mail messages. Both algorithms were tested within the Magi architecture to evaluate their performance. The results to date indicate that both IBPL1 and IBPL2 perform equally well when applied to classifying mail messages. Figures 4 and 5 show the number of messages for which predictions are made (i.e. coverage), and the accuracy of those predictions for both algorithms on the *Agents* mail box (Figure 4) and *Phd_Possibilities* mail box (Figure 5). Further tests are to be carried out with both algorithms on USENET news data and various datasets from the UCI Machine Learning Data Repository [24]. This is to determine if the *closest value* distance algorithm (IBPL2) is superior to the average distance algorithm (IBPL1), and if the quality of features used affects the performance of each algorithm.

4 Conclusions & Future Directions

The studies so far on Magi and UNA [25, 27] have shown that the type of information extracted from mail messages or USENET news articles can affect the concepts that are learned. Recent work on LAW [4] investigated different feature extraction methods but concluded that no single method improved the overall performance of the system. It is important to note that certain feature extraction techniques may be unsuitable for identifying features for some applications. For example, the feature extraction technique used by Magi was unsuitable for rating USENET news articles in UNA (as described above).

The work on IBPL1 described here has explored the use of *feature sets* to learn *user-profiles*. This overcomes the requirement to generate multiple instances, as is the case when existing learning algorithms such as CN2 and C4.5 are used. Different methods of comparing sets of values within exemplars and examples have been investigated, such as the *closest value* distance algorithm (IBPL2).

Algorithms that combine information filtering methods with machine learning methods are being considered. The IBPL1 algorithm is being modified to utilise the information retrieval technique, *tfidf* when considering weights and distance calculations. The use of weights will then be used to identify features that result in good classifiers. By pruning values that act as poor classifiers from feature sets, the number of distance

calculations made can be reduced. The separate feature extraction stage which appears in our architecture can then be eliminated, as the learning stage identifies the most useful features.

As the means of providing knowledge about the user shifts from user-defined scripts to the use of machine learning techniques, the user loses explicit control over defining the agent's behaviour. To avoid unexpected behaviour due to inadequately learned *user-profiles*, predicted actions (such as deleting mail messages or suggesting news articles) are generated with a confidence rating. This rating can be used to determine when the agent should become autonomous and act on behalf of the user without requesting confirmation. Some systems (such as MAXIMS [23]) interact directly with the user, requesting immediate feedback when the agent is unsure of how to act. Others (such as Magi) record predicted actions and allow the user to cancel or confirm actions before they are performed (see [26] for more details).

If machine learning techniques are to be used in practical agent systems, the user will no longer have to learn scripting languages or maintain scripts as requirements change. However, sufficient resources are needed to store observations and training instances from which the *user-profile* is learned. The learning and classification mechanisms have to be fast if such systems are to be of any practical use. The user will also have to develop trust in the agent's assistance for the agent to behave autonomously, and the agent should learn from its mistakes, through user feedback or by monitoring its own performance.

5 Acknowledgements

T.R. Payne acknowledges financial support provided by the UK Engineering & Physical Sciences Research Council (EPSRC). Thanks to Claire Green and Karla Fitzhugh for their helpful comments on earlier drafts of this paper.

References

[1] D.W. Aha. Incremental Instance-Based Learning of Independent and Graded Concept Descriptions. In *Proceedings of the 6th International Machine Learning Workshop*, pages 387–392, 1989.

[2] D.W. Aha and D. Kibler. Noise-Tolerant Instance-Based Learning Algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 794–799, 1989.

[3] L.W. Barsalou. Ideas, Central Tendency, and Frequency of Instantiation as Determinants of Graded Structures in Categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 11(4):629–654, 1985.

[4] D. Bayer. A Learning Agent for Resource Discovery on the World Wide Web. MSc Thesis, Department of Computing Science, University of Aberdeen, Scotland, 1995.

[5] Y. Biberman. The Role of Prototypicality in Exemplar-Based Learning. In *The 8th European Conference on Machine Learning*, pages 77–91, 1995.

[6] D. N. Chin. Intelligent Interfaces As Agents. In J. W. Sullivan and S. W. Tyler, editors, *Intelligent User Interfaces*, pages 177–206. New York, New York:ACM Press, 1991.

[7] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3:261–283, 1989.

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

[9] W.W. Cohen. Text Categorization and Relational Learning. In *The 12th International Conference on Machine Learning*, pages 124–132, 1995.

[10] S. Cost and S. Salzberg. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10:57–78, 1993.

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

[12] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[13] K. De Jong. Learning with Genetic Algorithms: An Overview. *Machine Learning*, 2:121–138, 1988.

[14] M.R. Genesereth and S.P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, 1994.

[15] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[16] C.L. Green. USENET News Agent. BSc Final Year Project Report, Department of Computing Science, University of Aberdeen, Scotland, 1995.

[17] C.L. Green, D. Bayer, and P. Edwards. Towards Practical Interface Agents which Manage Internet-Based Information. Submitted to Intelligent Agents Workshop, BCS Specialist Group on Expert Systems and Representation & Reasoning Special Interest Group, 1995.

[18] R. Kozierok. A Learning Approach to Knowledge Acquisition for Intelligent Interface Agents. Master's Thesis, Department of Electrical Engineering and Computer Science, MIT, 1993.

[19] K. Lang. NewsWeeder: Learning to Filter Netnews. In *Proceedings of the 12th International Machine Learning Conference (ML95)*, pages 331–339. San Francisco, CA:Morgan Kaufmann, 1995.

[20] H. Lieberman. Letizia: An Agent That Assists Web Browsing. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 924–929, 1995.

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

[22] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen. Intelligent Information-Sharing Systems. *Communications of the ACM*, 30(5):390–402, 1987.

[23] M.E. Metral. Design of a Generic Learning Interface Agent. BSc Thesis, Department of Electrical Engineering and Computer Science, MIT, 1993.

[24] P.M. Murphy and D.W. Aha. UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California, Irvine, CA. [http://www.ics.uci.edu/~mlearn/MLRepository.html], 1994.

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

[26] T.R. Payne and P. Edwards. Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface. Submitted to Applied Artificial Intelligence, 1995.

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

[28] J.R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5:239–266, 1990.

[29] J.R. Quinlan. *C4.5 Programs for Machine Learning.* San Mateo, CA:Morgan Kaufmann, 1993.

[30] E. Rosch and C.B. Mervis. Family Resemblances: Structures in the Internal Structure of Categories. *Cognitive Psychology*, 7:573–605, 1975.

[31] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.

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

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

[34] E.M. Voorhees. Software Agents for Information Retrieval. In *Software Agents: Papers from the 1994 Spring Symposium*, pages 126–129. Menlo Park, CA:AAAI Press, 1994.

[35] D. Wettschereck and D.W. Aha. Weighting Features. In *Proceedings of the 1st International Conference on Case-Based Reasoning.* Lisbon, Portugal:Springer-Verlag, 1995.

[36] D. Wettschereck, D.W. Aha, and T. Mohri. A Review and Comparative Evaluation of Feature Weighting Methods for Lazy Learning Algorithms. Technical Report AIC-95-012, NRL NCARAI, 1995.

[37] M.J. Wooldridge and N. Jennings. Agent Theories, Architectures and Languages: A Survey. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[38] J. Zhang. Selecting Typical Instances in Instance-Based Learning. In *Proceedings of the 9th International Machine Learning Workshop*, pages 470–479, 1992.