Learning Email Filtering Rules with Magi A Mail Agent Interface

Terry Payne

Department of Computing Science
University of Aberdeen
King's College
Aberdeen AB9 2UE

Abstract

As the volume of data on the Internet increases the need for better tools to handle this flood of data is also growing. Interface agents are tools which are designed to aid the user in using various applications. This project describes the development of an agent which employs machine learning techniques to discover rules for filtering email. It explains how the agent observes the user in handling mail and how these observations are used to help automate this task. The agent is then evaluated, through testing, to examine whether such a tool can be useful as a *personal assistant*. A description of existing work is given, along with the design rationale, and a number of future extensions are suggested.

1994

Declaration

I declare that this thesis has been composed by myself and describes my own work. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Terry Payne
5th September 1994
Department of Computing Science
University of Aberdeen
King’s College
Aberdeen

Acknowledgements

I wish to thank my supervisor Dr. Pete Edwards and Winton Davies for their advice and support throughout this project. Many of their ideas and remarks have been invaluable in the development of this project.

I also wish to thank those at the flat; Becky Dodds, Steve (Crash) Ward and Christopher Mullin, not only for enduring my enthusiasm in the testing of this work, but for their support and comments on this dissertation.

Chapter 1

Introduction

This report explains the need for a personalised filtering agent for Email, and describes the process of building such an agent. This introductory chapter outlines the reasons for the work and presents the main objectives of the project.

1.1 The Problem

Over recent years there has been a dramatic increase in the use of networks and networked information retrieval systems. Not only has this happened in the business sector and throughout academia, but due to the increasing number of Internet intermediary services such as Compuserve or Demon Internet Services, an increasing number of hosts have been appearing in the private sector. At this present time, the internet consists of over 31 000 networks, with over two million computers connected to it. Over 20 million people can be reached by electronic mail and have access to the resources on the internet [Leiner 1994].

This increase has led to an explosion of information resources available over these networks. As more information becomes available, searching or retrieving interesting or relevant information is becoming increasingly difficult [Sheth 1994]. Whilst there is a large amount of data available as files on different machines, much of it exists in the form of Email, USENET news, World Wide Web servers and Gopher servers etc, and tools have been developed to access these resources.

With access to these information sources becoming easier, and the number of users becoming greater, the need for easier to use software is becoming critical. However with the increase of easy to use tools for generating and distributing information, the amount of information flowing across the networks is growing [Denning 1982]. Business and research organisations can generate huge amounts of information, such as memoranda, announcements of meetings and conferences etc, yet at any one time this information will be of interest to only a fraction of the recipients [Foltz & Dumais 1992].

Information Filtering is not a new concept. It already exists in the paper world; people buy only certain magazines that contain articles on a particular subject, and then skim

articles to find ones of intertest. With the use of electronic media, some of this filtering can be automated by the retrieval system. Over ten years ago *agents* were seen as anthropomorphic entities which could assist the user by tracking down information it knew its user was interested in [Kay 1984]. This description still holds true and agent technology is growing. Agents are now emerging as *personal assistants* which can assist in automating information filtering as well as information retrieval.

An agent, however, must be able to fulfill certain criteria if it is to perform successfully as an information filter.

- As information requirements vary greatly from user to user, the filtering system should be highly personalised to satisfy the users needs. This *model* it has of the user should be learned from instruction (eg defined rules or a knowledge base) or from example (by observing the user). This model may be used over a long period of time, yet it cannot be assumed that the requirements of the user will remain static, so the agent must be able to notice when these requirements change and revise its model accordingly.
- The agent should attempt to filter out as little information as possible that would validly fit the user model, whilst removing as much information as it can that is not of interest to the user. If the agent fails this criteria more than 50%, then its performance is no better than random guessing and becomes a hindrance rather than an help to the user. The main aim of the agent is to *assist* the user in filtering out unwanted information, not to attempt to classify all information as relevant or irrelevant.
- The user should have some means of accepting or rejecting the decisions of the agent. The agent is not the user interface! It is an entity which collaborates with the user to aid them, and hence they should be able to choose when to allow the agent to perform a task and when to perform the task themselves.

1.2 Issues the Project Addresses

The work presented within this report makes use of techniques taken from the fields of information filtering and of interface agents. These techniques were used to build an agent which observes and aids a user in classifying mail messages into different folders. The following issues are addressed:

- Making and logging observations of the user in order to build a user model. The learning algorithm CN2 [Clark 1989; Clark & Niblett 1989; Boswell 1990] is used to induce rules based on these observations which can then be used to model the user. A method of applying these observations as training examples is investigated and the problems of selecting features from a mail message is examined.
- The use of an interface agent which attempts to classify incoming mail according to the user model. The agent is unobtrusive in the general use of the mail tool, and using its advice is optional. The use of basic positive feedback to reinforce successful classifications is also addressed. More advanced feedback is also examined from a viewpoint of making the agent easy to co-operate with so that it acts as an assistant rather than an interface to the system.
- The use of a genetic based word filter is investigated as a more advanced method of feature extraction. The use of user feedback based on the success of the agent is used to refine features of a message which contribute towards successful classifications. This can be compared to the more naive use of word frequencies to determine high entropy words.
- The use of separate, communicating units, each with a specialised role within the system. These roles include rule generation, feature extraction, message classification, and user observation. Not only do they aid in the maintenance of the system but also improve usability by performing processor intensive calculations at off-peak times.

Experimental results are used to compare different configurations of the system in order to improve performance. These also give some indication as to the properties of the classifications that the agent can confidently advise.

1.3 Overview of Dissertation

The remaining dissertation is organised as follows:

- *Chapter 2* - This presents the problem more thoroughly and examines how previous solutions have tackled the problem. It looks at the developing work with respect to interface agents and information filtering, and assesses the current research in the use of learning techniques to model the user.

- *Chapter 3* - CN2 is used as the core algorithm for learning rules for mail classification. This chapter describes this algorithm and explains the reasons for its use within this specific application.
- *Chapter 4* - Learning Email Filter Rules. This chapter explains how the different parts of the system are built up and how they work together to act as an agent in observing and modeling the users requirements.
- *Chapter 5* - Implementation. This discusses problems encountered with the implementation of the system and how they were solved.
- *Chapter 6* - Results. Due to the nature of the implementation, there are many parameters which can be varied to improve performance of the system. Much of the testing was concentrated on varying these parameters and observing how they affected the performance of the system.
- *Chapter 7* - Conclusion. This draws together the results and presents concluding remarks.
- *Chapter 8* - Future Work. The work presented within this dissertation throws up many possibilities of future work in the area of interface agents and personalised filters. This chapter examines some of these possible routes.

Chapter 2

Mail Filtering and Interface Agents

The advances in networking over recent years have provided a rich environment for sharing and exchange of information. Research is continually investigating faster means of transferring data across the internet, such as the SuperJanet project in the UK, or better methods of storing data, such as advanced optical storage systems [Bains 1994]. Beacuse of this improving media, the rate and complexity of information traffic is continually growing, along with the need to handle this now overwhelming flood of data.

Visionaries, such as Denning and Kay, foresaw the need for mechanisms to aid the user in filtering out unwanted information delivered to the user [Denning 1982] or in searching out information of interest [Kay 1984]. These two distinct research areas of Information Retrieval/Filtering and Software Agents are now beginning to recombine in the development of interface agents. An interface agent can act as a *personal assistant* which can collaborate with the user to assist in filtering out unwanted information and acquiring new potentially interesting information [Maes 1994].

2.1 The problem of too much information - an introduction

Tools have been emerging to handle this information in different ways. On the Internet, a number of different tools have been developed to access some of this wealth of data scattered across the network. These tools access servers such as the World Wide Web (WWW), Gopher and the Wide Area Information Servers (WAIS) [Sheth 1994], as well as more familiar services such as Usenet NEWS and Email. The World Wide Web [Berners-Lee et al. 1994] provides a platform for building and browsing hypertext documents, with internet links to other documents scattered on the internet. Gopher is used for browsing hierarchically organised documents, but both this and the WWW can contain indices to aid the user in searching for documents. WAIS responds to queries for keyword searches and allows the user to refine searches for network based documents.

These servers rely on the information retrieval paradigm of the user requesting information and the server responding to the query. Another paradigm, information filtering, corresponds more to services such as Email and the News Network. Email is a means of sending information from one user to one or more users in the form of a

message. Usenet NEWS consists of messages organised into different topics, or *newsgroups*, which can be read by any user interested in that topic. Both these services deliver information to the user. This can leave the user swamped with a plethora of messages, many of which the individual is not interested in. Whilst there exist basic mechanisms to receive messages of interest, such as the newsgroup organisation hierarchy and mailing lists, it is still possible to be overwhelmed by the high traffic of messages found on some of the lists or newsgroups. In addition to this, there can exist a high volume of organisational mail distributed to users within business or research organisations, much of which is of no interest to the recipient.

2.2 Information Filtering

Information Filtering vs. Information Retrieval

There are many similarities at the abstract level between Information Filtering and Information Retrieval [Belkin & Croft 1992]. Both are concerned with users only receiving information that they are interested in, and so many issues in information retrieval are also relevant to information filtering. Two issues however are of special interest to the information filtering community; they are the issues of text representation and of refinement:

- Unlike retrieval systems which are mostly designed with structured data in mind, such as employee records, filtering systems normally have to deal with unstructured or semi-structured data. Email messages are a good example of this type of data, in that they have well defined header fields but an unstructured text body.
- Filtering normally involves a stream of incoming data over a long period of time. However, as the requirements of the user may change over time, the *profile* or description of user interests should evolve to reflect these changes.

Issues such as the comparison of profiles with message features, or profile refinement through such mechanisms as *relevance feedback* [Salton & McGill 1983] are still as valid within filtering as in retrieval.

Many of the issues of text representation are also valid. Two possible approaches to representation have been widely explored; *statistical* approaches and *semantic* approaches. Statistical approaches operate on the words themselves, such as keyword or probabilistic representations, or vector space representations [Salton & McGill 1983] and

Latent Semantic Indexing [Foltz & Dumais 1992]. The semantic approaches lie within the use of natural language [Ram 1991; Ram 1992; Riloff & Lehnert 1992]. One Information Retrieval system, *SCISOR* [Jacobs & Rau 1990] makes use of a combination of top-down and bottom-up processing techniques in natural language analysis to process on-line news feeds.

Information Filters

In the preliminary stages of the *Oval* project [Malone et al. 1987], Malone identified three approaches to information filtering by surveying different people on what criteria they use to filter information from various systems;

- *Cognitive Filtering* - this characterises a message by the contents and meaning of the message. Participants in the survey looked for certain keywords or phrases to classify messages.
- *Social Filtering* - this complements the cognitive approach by concentrating on the personal and organisational interrelationships between sender and receiver. For example, more attention may be given to messages from a superior such as a supervisor.
- *Economic Filtering* - this is based on a cost-benefit assessment of a message, such as the length of a message.

It was from these studies that the Information Lens, or Oval System was developed. A rich set of hierarchically organised templates were used to structure mail messages. An example of this organisation could be a *notice* template. This would have certain fixed fields added to the mail message. A specialised form of this template, e.g. for meetings, would not only inherit the structure of the *notice* template but would contain extra structured fields. User defined rules could then be defined to not only match keywords within the text body, but to match template types and make use of the extra structured information held within the template fields.

Another study by Stadnyk & Kass [Stadnyk & Kass 1992] examines the possibility of building a knowledge base of description categories that users employ when deciding whether or not to read a message. They also noticed that the type of rules employed by users could be generated automatically using machine learning algorithms.

Email filters have become popular in recent years and there are many which are now freely available. *Elm* [Weinstein 1992] is one such filter built into a mail tool. Similar to *Oval*, it allows rules to be defined by matching certain keywords within the header field or message body. The rules can then filter out unwanted messages, sort message types into mail boxes, or perform more complex commands.

Ram approaches the subject of filters from an Artificial Intelligence approach by applying natural language understanding mechanisms to filtering. The PIES system (Personal Interest Engine for Stories) [Ram 1991; Ram 1992] makes use of a model representing the interest and relevance of different concepts. This model is used to prune away concepts unlikely to be of interest from the story or message. An alternative approach was used by Riloff and Lehnert with their *Relevancy Signatures Algorithm* in classifying articles about terrorism [Riloff & Lehnert 1992]. Other systems have been written to skim and summarise news articles, such as the *FRUMP* system [DeJong 1982].

2.2.1 Interface Agents

Interface Agents are programs that provide assistance to a user for different tasks. Information filters can be seen as agents as they aid the user in handling large quantities of incoming information. So far the approach to developing personalised information filters relies on having a model of the users interest. One of three approaches could be adopted to generate this model:

- The user may customise their own rules. This approach is used in many filters, such as the *Oval* project. The inherent problems with this is that it requires too much insight by the user not only into their requirements, but into how the filter will perform with regard to these rules. The user also has to be responsible for maintaining the rules over time, as interests often change.

- Knowledge Acquisition techniques could be used [Boose & Gaines 1989]. This is found more in interface agents that aid a user in certain environments. For example, UCEgo [Chin 1991] has a large knowledge base about how to use the Unix Operating System. It uses this knowledge base to help users solve problems when using Unix.

  Making use of knowledge acquisition techniques can help capture regularities in the types of classifications users make [Stadnyk & Kass 1992]. The knowledge base no longer requires the user to program complex rules. However this approach fails to

customise the rules to the users specific requirements.

- Observations of the users actions can generate training examples. When applied to machine learning techniques, rules can be induced. This will result in a personalised rulebase with no additional work from the user.

This final approach is being used in a number of interface agents to help the user in performing organisational tasks, such as calendar management [Dent et al. 1992; Maes & Kozierok 1993], exploring newsgroups for interesting articles [Sheth 1994] and Email filters [Metral 1993].

CAP (Calendar APprentice) is a personal learning apprentice which assists in managing a meeting calendar [Dent et al. 1992]. The calendar manager is used by filling in parameters for a given meeting. This forms training examples which are logged and subsequently used by the learning component. The rules generated are then used to automatically fill in parameters for subsequent meetings. In the early stages of this project, two learning algorithms were compared; *THEO* which is a variant of Quinlan's ID3 algorithm [Quinlan 1986] producing decision trees; and Backpropagation [Rumelhart et al. 1986], an artificial neural network (ANN) algorithm. 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].

An interesting issue that arose from the work on *CAP* was that of coverage. *Coverage* is used to represent the number of classes the ruleset can cover. It was noted that as the coverage of the rules increased, the overall accuracy decreased. This could indicate that by making rules cover more possible classes, they can become over-generalised and hence less accurate.

The work at MIT by Kozierok [Kozierok & Maes 1993; Maes & Kozierok 1993] used the machine learning method *Memory-Based Reasoning* [Stanfill & Waltz 1986]. An unusual aspect of this work was the use of caricatures to provide feedback to the user of the current state of the agent. They addressed the problem of trusting the agent by generating a confidence value for each agent prediction. Two thresholds were used; the lower being a "tell-me" threshold, where the agent needed user confirmation before performing the action. The higher was a "do-it" threshold. Predictions with a confidence rating higher than this could be performed automatically.

This work was the basis for a mail filter. Metral [Metral 1993] examined the design of a generic learning interface agent. The agent used Memory-Based Reasoning to store and compare past situations to new ones, and made use of graphical caricatures to represent the current state of the agent. An off the shelf mail application, *Eudora* was modified to interact with the agent.

An alternative approach to learning within agents at MIT was explored by Sheth [Sheth 1994]. A news reader was developed, NEWT (News Taylor) which made use of user feedback to identify articles of interest. The system maintained a set of agents, each responsible for exploring a different interest. New agents can be created by the user, and given a profile of the users interests. The representation used for profiles and incoming articles is based on the vector-space representation [Salton & McGill 1983]. A *genetic algorithm* approach is used to explore different profiles. *Crossover* is used to combine parts of two profiles to create a new one, whereas *mutation* is used to modify a user profile. A fitness score is calculated for each profile and is modified according to a users response to articles found by the agent.

The ideas of Genetic Algorithms were also adopted by Baclace in his Personal Information Intake Filter [Baclace 1991; Baclace 1992]. This filter is also based on ideas from *Agoric Systems* [Miller & Drexler 1988], whereby agents compete via market trading. Each agent is sensitive to features in an article, and possesses a fitness value or "store of money". When these features are found within a new article, the agents sensitive to the features rate the article. Agents rating the article get charged for the privilege of contributing towards the rating. Once the user has read the article, feedback is returned as to how accurate the rating is; accurate agents are rewarded, whereas incorrect agents are penalised. Crossover is used to create new agents by creating a conjunction of two existing features.

2.3 The Proposed Approach

The Mail AGent Interface (*MAGI*) detailed in this dissertation draws inspiration from both the above mentioned fields of research and utilises them in a mail agent. It makes use of a machine learning approach to build up a user model or profile of the users interests. User actions are observed for use in creating rules, and these rules are used to filter incoming mail messages. The CN2 induction algorithm (described in the next chapter) is used to induce rules based on these observations.

The issue of confidence is addressed by use of multiple rules for any action. A single message can fire more than one rule. For each action or classification, there will be many rules created. A confidence threshold is set so that a minimum number of rules must fire with the same action or classification for the action to be accepted. Once an action is accepted by the agent, it will then be proposed to the user, who can accept or reject it. Basic feedback is provided in the form of new training examples if the action proposed by the agent is then executed.

The work will initially concentrate on extracting features from the message body, as well as later using the semi-structured mail header fields. A more advanced approach is proposed to feature extraction based on the work of Miller & Drexler, and of Baclace. This involves a community of agents which extract features of interest from the message body. Agents are then rewarded or penalised depending on how well they describe the message. User feedback is used to determine the quality of the rules and the features extracted.

Chapter 3

Inductive Learning Algorithms

This chapter describes the CN2 algorithm used in Magi, and discusses the problems encountered in applying it in the agent.

3.1 CN2

The use of learning algorithms for inducing concept descriptions from examples has eased the bottleneck of knowledge acquisition. Algorithms such as ID3 [Quinlan 1986] or those from the AQ family have been especially successful. However, both families are susceptible to domains with noisy data. Some members of the AQ family have been developed to preprocess noisy data (eg AQ15 [Michalski et al. 1986]), but these leave the AQ algorithm intact.

CN2 [Clark & Niblett 1989; Clark 1989] was designed to modify the AQ algorithm to solve these problems. The algorithm works in an iterative fashion, with each iteration searching for a complex which covers many examples of class *C* and few of any other class. A *complex* is a conjunction of attribute tests. This complex forms the conditional part of a production rule, where the class *C* is the result of the production rule. Once a complex is found, the examples it covers are removed from the training set, and the rule

if *complex* then *Class C*

is added to the end of the decision list. The algorithm repeats itself until there are no more examples in the training set.

A complex is found by using a beam search to specialise rules. A complex is specialised by either adding a new conjunctive term, or removing a disjunctive element in one of its selectors. A size limited set, called the *star* stores all the complexes that are being considered. These are the 'best complexes found so far'. Initially an empty complex which covers all the training examples is used. Specialisation has been implemented by repeatedly *intersecting* the set of all possible selectors with the current complex. All unchanged elements or null complexes are removed (a null complex is one that contains a pair of incompatible selectors, for example $big{=}y \wedge big{=}n$).

The *star* is then trimmed. This is done by evaluating each of the new complexes, and discarding its lowest ranking elements. This is done by using two evaluation functions. The first evaluation uses the information-theoretic entropy measure (1.1)

$$Entropy = -\sum_i p_i \log_2(p_i) \tag{1.1}$$

where the lower the entropy the better the complex. First the set E' of examples is found which a complex covers. Then the probability distribution $P = (p_1, \ldots, p_n)$ of examples in E' among n classes is also found. The above measure is then applied to the complex.

An alternative search heuristic can be used to trim the star. The *Laplacian* error estimate is shown below (1.2):

$$Accuracy(n, n_c, k) = \frac{(n - n_c + k - 1)}{(n + k)} \tag{1.2}$$

where:

n = total number of examples covered by the rule
n_c = number of positive examples covered by the rule
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. If the difference is greater than that which can be accounted for though pure chance, the complex is considered to be significant.

The significance is calculated by using the likelihood ratio statistic [Clark & Niblett 1989], given below (1.3):

$$2 \sum_{i=1}^{n} f_i \log\left(\frac{f_i}{e_i}\right) \tag{1.3}$$

where the distribution $F = (f_1, \ldots, f_n)$ is the observed frequency distribution, and

$E = (e_1, \ldots, e_n)$ is the expected frequency distribution. This statistic provides a theoretic measure of distance between the two distributions. Under suitable assumptions, this statistic is distributed approximately as χ^2 with $n - 1$ degrees of freedom.

CN2 provides an ordered list of production (or if-then) rules. Ordered rules have the disadvantage in that they sacrifice comprehensibility. This is due to any single rule being dependent on its preceding rules. However, their advantage is that, unlike unordered rules, there is no need to provide an additional mechanism to resolve rule conflicts due to two or more rules firing. Unordered rules can be produced by CN2, by changing the evaluation function to the AQR function used by the AQ family.

3.2 Issues regarding the Application of CN2 to Mail Filtering

The approach of using CN2 to induce rules, as opposed to other approaches such as using Memory Based Reasoning [Metral 1993] or a genetic approach [Sheth 1994] was to generate a set of rules which could be read and modified by the user. Approaches such as those mentioned above maintain a state of learning which is inaccessible to the user, except through the application or other related tools. The rules generated by Magi are in the form of production rules in a standard text file.

In order to induce rules from mail messages, one needs to select features from the message. A message can be split into attributes, one for each fields and one for the message body. However the problem of multiple attribute values soon becomes apparent. A filtering heuristic can be used to find features in the message body which characterise the message (a heuristic based on frequency is described in this dissertation). A message with similar characteristic may not generate exactly the same features, although there may be some overlap.

This raises the issue of whether a single training example for a mail message is sufficient, or whether a disjunct of simpler training examples should be used. Magi makes use of multiple training examples to overcome this problem. A disjunct of examples is created comprising of combinations of the different attribute values generated from a single mail message.

This raises the second issue of multiple rules. If more than one example is generated and then applied to the rules, each example may fire a rule. This introduces a need for some form of conflict resolution; although at most one rule will fire with any given example if

an ordered ruleset is used, many examples may fire many rules. Chapter 4 describes a solution to this problem.

Chapter 4

MAGI - Mail AGent Interface

This chapter describes the mail agent *Magi* and details each of the components used in building the system. Each component is described in turn; why it is needed, its constraints, and its function within the system.

4.1 System Requirements - An Introduction

The requirement is to develop a mail agent which aids its user in handling mail. The agent sits above the mail tool observing the users interactions with the tool. It can interact with the mail tool in order to perform actions automatically for the user.

The following is a list of the types of actions the agent will attempt to learn from observing the user:

- Messages which are filed away in different mailboxes for later browsing.
- Junk mail which the user is never interested in reading.
- Messages which are forwarded to other users.

These actions all fall under Malones criteria of Cognitive filtering [Malone et al. 1987], whereby filtering is based on the contents of the mail message. Other behaviour could be observed, such as the order in which mail is read and if this varies on the author or recipients (Social Filtering), or the time spent reading mail or the size of the mail messages themselves (Economic Filtering).

The agent comprises of a number of modules, each of which are responsible for certain tasks. The main reason for this is one of performance. Each time the rulebase is generated, the existing rules are discarded and new ones are created. This process can be processor intensive, so it occurs as a regular batch process. Classification of new mail messages is performed each time a new message is delivered; again to reduce processing time when the mail tool is invoked.

An aim of the project is to make the agent as transparent as possible. Mail agents such as that which communicates with the mail tool Eudora [Metral 1993] rely on the user interacting with the agent in addition to the mail tool, with caricatures used to reflect the

agents current state. With Magi, the agent is invisible to the user and does not effect the users handling of mail. The only time the user is aware of the agent is when the user requests help. Because of this the agent has to interact with an existing mail tool. The standard Berkeley *Mail* tool is used.

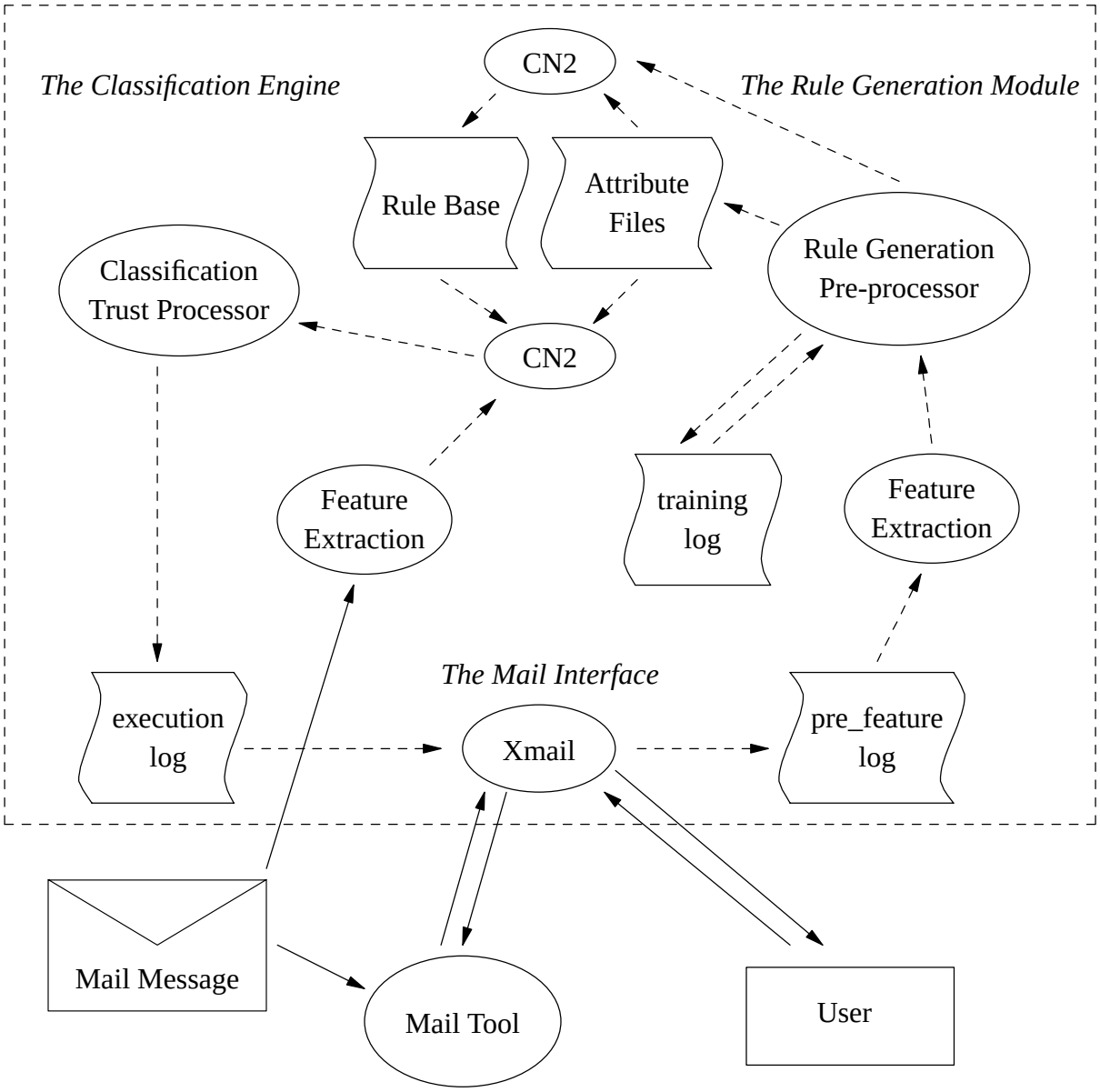


Figure 1 - The Agent Overview

An overview of the agent can be seen in figure 1. This overview highlights the modularity of the agent. The modules communicate by means of shared files. The three modules can be summarised as follows:

- *The Mail Interface* - This module is responsible for observing and negotiating with the user. User actions on mail messages are recorded by the mail interface for later rule generation. The interface also negotiates with the user which automated actions to perform. These automated actions are generated by the classification engine and are actions the agent is confident in performing.
- *The Rule Generation Module* - A record of user actions on mail messages is kept by the mail interface. This record is used to generate training examples which are used to induce rules. Each action has a given *life-time*, during which it contributes towards rule generation. This is so that the user profile can reflect the user over a given time. The rule generation module is responsible for handling this, and for pruning the training set as training examples become old.
- *The Classification Engine* - This module tests each new mail message against the rule base, and analyses the results. As many rules may fire with a given message, this module is responsible for assessing the confidence in the results of the rules. The term *classification* here is used to represent any mail action the agent knows about and can perform on behalf of the user.

An important component common to both the rule generation module and the classification engine is that of feature extraction. This is responsible for breaking up mail messages into features which are matched against the rule base. Similarly, messages need to be broken up into features in the same way to generate a new rule base. This component is described below.

4.2 The Mail Interface

This module sits transparently between the user and the mail tool. With conventional mail tools, the user issues commands directly to the tool, by typing commands or using a graphical user interface (GUI). It then responds to these commands by displaying mail messages or managing the mailbox, depending on the command (Fig 2).

The mail interface sits between the user and the mail tool (Fig 3). Commands are intercepted by the mail agent before being passed to the mail tool. This allows the agent to observe the user handling mail. Likewise, any response from the mail tool can be intercepted by the mail agent. This allows the mail agent to observe the result of user commands.

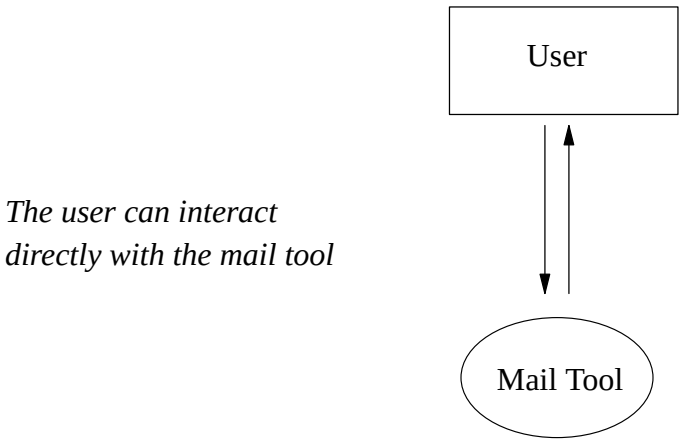


Figure 2 - Using a mail tool

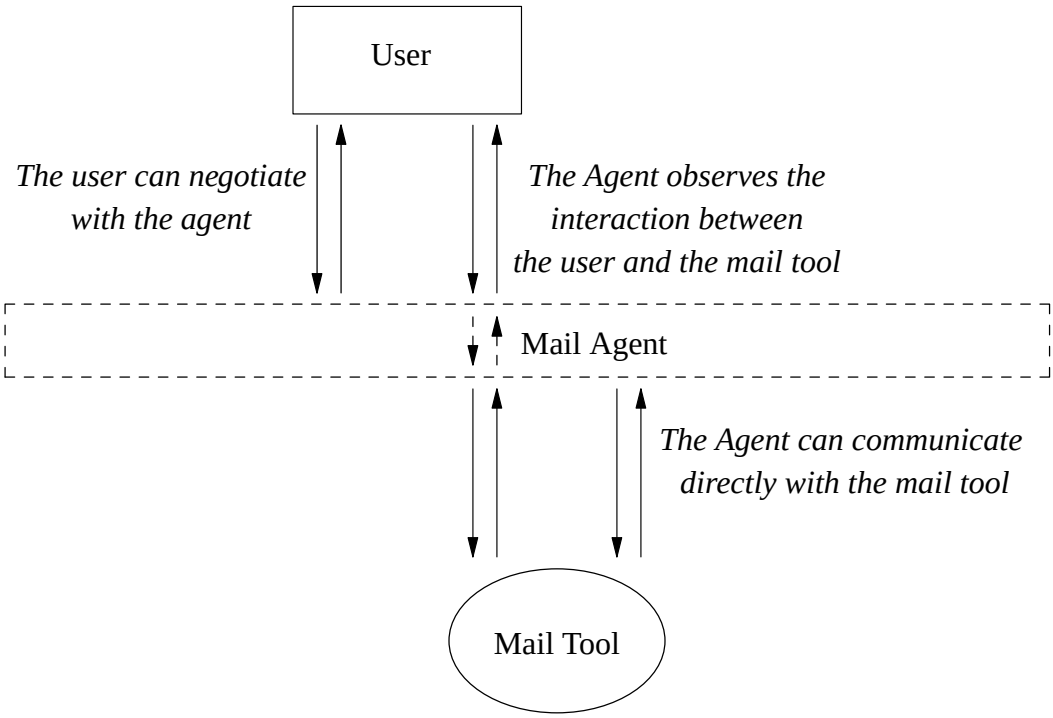


Figure 3 - Inserting the mail interface above the mail tool

As the agent lies between the user and the mail tool, the agent is able to negotiate directly with either the user or the mail tool. In this way, the user is able to examine and select actions proposed by the agent whilst using the mail tool. Once the user is happy with the proposed actions, the agent is then able to interact with the mail tool directly without affecting the user.

Whilst the agent is to be transparent to the user, the user may wish to negotiate with the agent directly. Because of this and other considerations (see Chapter 5 - *Implementation Details*) a graphical user interface was modified. This interface, *Xmail,* sits above Berkeley *Mail* providing a graphical interface but not acting as a mail tool itself. Extra functionality has been added allowing the user to negotiate directly with the agent. This is discussed below in *User Feedback*.

4.2.1 Capturing user actions

The agent is only interested in a subset of mail commands. These commands are mainly responsible for filing away mail in different mailboxes, or the deletion of unread mail.

The commands are intercepted by the agent before passing to the mail tool. At this point all that is known is that there is a message that needs to be filed away or deleted. In filing mail messages, the agent requests the mail message from the mail tool in order to extract features from the message at a later date. This message and the user command are stored in a *pre_feature* log file. Once this *observation* has been made, the command has to be executed. This is done by the agent issuing the command directly to the mail tool, and passing the results back to the user (see fig 4).

Deletion is a different matter. Many mail tools offer the ability to undelete mail messages during the mail reading session (Berkeley *Mail* is one such tool that allows this). If a log is made of a message being deleted, this needs to be compensated for if the message is then undeleted. In this case, the agent keeps a note of all messages deleted without logging this fact. If the message is undeleted, this is also registered by the agent. When the mail tool session ends, the agent then logs all the deleted messages at that point.

There may be times when administering mail that commands need to be issued without being logged by the agent. This may be because of an atypical command or commands which might affect the rules generated. A mechanism is provided so the user can inform the agent to ignore the user until told otherwise.

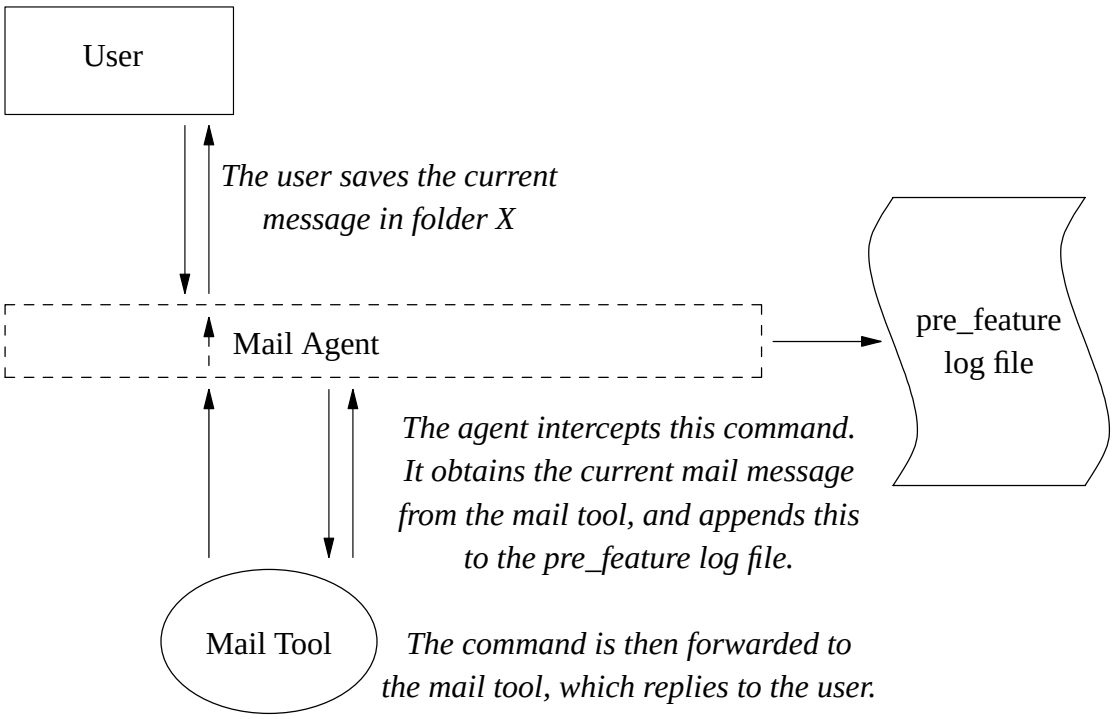


Figure 4 - Making observations on message filing

4.2.2 Automating User Actions

The classification engine examines incoming mail and determines if actions should be proposed by the agent. These proposed actions are stored in a log file called the *execution log*. Each entry comprises of the action or command the agent proposes to issue, and some means of identifying the mail message on which the action is to be executed.

When the user has negotiated with the agent to decide which actions are to be performed, the agent searches for each message in the mail box and performs the actions. The agent makes a record of the actions performed which is displayed to the user.

An important consideration here is that there has to be some way of overriding the agents proposed actions. It is possible that the agent may mis-classify a message, due to poor training examples or as part of the user changing their requirements. Because of this, the facility for the user to negotiate with the agent regarding proposed actions is included. In the majority of cases it would be expected that the user would trust the agent to perform

its proposed actions.

Confidence in the agents proposed actions is represented by the number of rules firing to yield the same action (see below in *Classifying Messages*). A *trust threshold* is used to determine whether or not the agent has high confidence in a proposed action. This threshold is consistent across all actions and has been determined empirically.

Actions proposed by and performed by the agent are also considered for future training examples. In this way messages, for which the agent correctly predicts the action, will contribute towards the correct action when rules are later generated. This provides positive feedback for the rule generation module.

Initiating agent proposed actions and negotiating with the agent as to which message-action pairs are to be performed are two examples where the user communicates with the agent as opposed to the mail tool.

4.2.3 User Feedback

An important function in negotiating with the agent is to be able to select which message-action pairs proposed by the agent are to be performed. For this, the action browser was designed. This lists all the message-action pairs, *marking* each one to indicate that it will be executed. As the messages are referenced by the message-action pairs using a unique identification value (the *Message ID*), this list can mean very little to the user. The user can examine any message from this list. On doing so the message is displayed in the window below. Any actions not to be performed can then be unselected. Once the user is happy with the agent predictions, they can be performed.

Because this mechanism identifies correct and incorrect agent predictions it can be used to provide additional feedback, both positive and negative, to the agent. This is of use to the advanced filtering system described below (see *Advanced Filtering*). Here, feedback is used to reward or penalise features that are used in rule generation.

This list of message-action pairs could be modified so that actions not normally proposed can be selected. Currently the *trust threshold* is set within the classification engine. For each message, a *confidence* value is generated from the number of rules that fire for a given action. The agent will propose a message-action pair if the confidence value is greater than the trust value, ie. the agent has a high confidence in this action. This is explained in greater detail below (see *Classifying Messages*).

An alternative approach would be to add the confidence count to each message-action pair forming a message-action triplet. The user could also define their own *trust thresholds* for every possible action. For each entry in the list, the confidence count could then be compared to these user defined *trust thresholds* within the mail interface. If a confidence count is greater than the *trust threshold* for that action, then that entry could be marked for execution.

The main advantage of this approach would be to improve user feedback. The current action browser displays entries which the agent has high confidence in. The user can then reject any entries not to be performed. This new approach would also list entries for which the agent has low confidence, ie. the confidence value is less than or equal to the *trust threshold* for the specific action. The user can then select entries which the agent has low confidence in to be performed, or to reject entries which the agent has high confidence in.

4.3 Feature Extraction

The rule generation module utilises message features to induce rules for the rulebase. Similarly the classification module compares message features to the rulebase to detect messages the agent may want to propose to the user. Both modules require these features to be extracted from the message.

The feature extraction mechanism was designed so that it could be used from both the classification module and the rule generation module. This ensures that the message will generate the same features for both modules.

Each message is divided into two parts. The first, the message header, contains structure information. The second part is the message body. Two approaches to extracting features from the message body are discussed in the sections *Basic Filtering* and *Advanced Filtering* below.

The message header contains information about the routeing used by the message to reach its deadline. It also contains information about who the sender was, the time and date sent, recipients, status information etc. Magi is currently only interested in two of these fields. These are the *From* and *Subject* fields, which contain the sender and a brief synopsis of the message respectively.

The *From* field is preserved as a single string. The subject field is parsed into words. Words found in the common stoplist are removed, and the remaining words used. See *Basic Filtering* for more details on this parsing.

In generating rules, attribute files are generated, containing attributes which may be found in the rule-base. These files are used for additional filtering when classifying rules (see *Classifying Messages*).

4.3.1 Basic Filtering

This is the main filtering adopted by Magi. The message body is parsed into words. Words are defined as sequences of characters delimited by whitespace, ie. space, tab, or newline characters. These words are then sorted into descending order by frequency. The top *N* words are then used to describe the message, where the number *N* is defined by the agent. Punctuation and numerics are removed to reduce spurious words. See the section *Implementation* for details on how words are parsed out of the mail message. An example of this is given in Chapter 5.

A high proportion of the most frequent words found will be common words used as part of every day language, for example *and*, *is*, *the* etc. A file containing these words, or *stoplist*, is used to filter them out. As words are parsed out of the message body, the stoplist is checked. If the word exists in the stoplist, it is immediately discarded. This is used to improve the *entropy*, or information content of the high occurrence words in the message body.

All the tests described in the *Results* chapter use this method of filtering.

There are many other possible approaches to this problem. The body of the message contains a wide range of words, where verbs can exist in different tenses (eg is, was, will be), nouns in different cases (eg tree, trees), and synonyms (eg freedom, liberty, independence), not to mention the issues of multiple spellings (eg color, colour). Approaches can be taken to take these considerations into account. Thesauri and word hierarchies could also be used [Miller et al. 1990] to reduce the number of synonyms found in messages.

4.3.2 Advanced Filtering

One problem with the basic filtering method described above is that there is limited

feedback of the success of the rules generated. Successful agent proposals will generate further training examples. However, some features may contribute to more than one action, and so may cause mis-classifications. This has been partially investigated by varying the *trust* threshold.

An alternative solution, is based on the work by Baclace [Baclace 1991; Baclace 1992]. The features themselves can be monitored rewarding features which contribute towards correct classifications and penalising features which cause mis-classifications. This is achieved by using an additional filter to the existing basic filter. It builds up over time a knowledge base containing a number of different features. Each feature contains a *fitness* value which can vary, depending on whether the feature contributes toward correct classifications or mis-classifications.

When a new feature is encountered, ie. it does not already exist in the knowledge base, a new entry is made. A standard starting value for its *fitness* is assigned. Once a feature exists in the knowledge base, its *fitness* value determines whether the feature is passed through the filter. Features with a low *fitness* are considered as unhelpful and are filtered out, whereas those with a high *fitness* are allowed through the filter.

The *fitness* of a feature has to reflect how good the feature is in determining classifications. This is achieved by rewarding the feature by increasing its *fitness* if the feature contributes towards a correct classification, and penalising the feature by reducing its *fitness* if the feature contributes an incorrect classification. The amount by which the *fitness* is altered is determined by the frequency of the feature in the classified message. Thus if the feature had a high frequency then the *fitness* is adjusted more than a feature with a lower frequency.

There is the danger that once a feature's *fitness* falls below the filtering threshold (ie. is now filtered out), the feature can never be reused. The feature may at the time yield mis-classification, but may be very useful to the agent at a later date, due to the user profile changing. Once a feature is filtered out, it can never contribute towards a correct classification and hence be rewarded.

This problem is overcome by only keeping these features in the knowledge base for a limited period. Once a feature is considered unhelpful, ie its *fitness* falls below a threshold, its *fitness* is then decremented during each classification. Once its *fitness* falls below zero it is then removed from the knowledge base.

This is in the prototype stage. Due to a lack of time at the end of the project this filter was not complete. Hence no test results exist to show if this filtering improves performance. An example of this mechanism is shown in Appendix A.

4.4 Learning Rules

The machine learning algorithm CN2 is used to induce rules based on a series of training examples. The training examples are generated from two sources. The first set is generated from observations made of the user by the agent since rules were last generated. The observations are stored in the *pre_feature* log file. Each observation consists of the mail message and the action performed on it. It is at this point that features are extracted from the message using the feature extraction component.

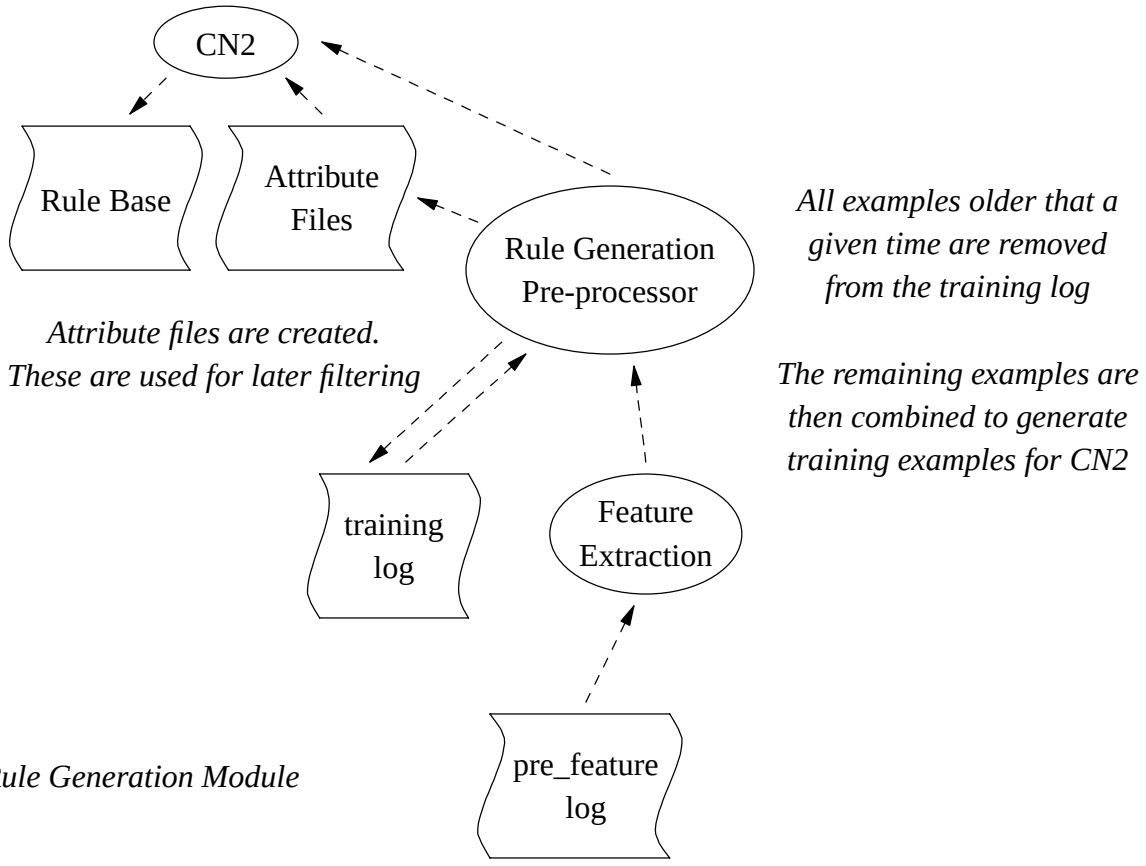


Figure 5 - The rule generation module

The second source of training examples comes from training examples used in past rule generations. Once an example is used in the induction of rules, it is given a *shelf-life*.

This shelf-life can be determined empirically. When rules are generated, the old ruleset is discarded. Many of the old rules may still be valid, but may not be generated by the new observations. This is overcome by preserving the most recent training examples over a given time. Once a training example has been preserved for longer than this time it will be discarded.

The advantage of this approach is that user behaviour is modelled over a recent given time (for example, the last month). This allows for infrequent but predictable actions to be learned, but also enables the agent to adapt to new interests. User behaviour will change, so there is a need to *forget* training examples which are no longer valid. This is accomplished by discarding old examples once they have exceeded this shelf-life (see figure 5).

A single mail message will generate multiple training examples. Up to *N* features are extracted from the message body, which can be used to characterise the message. The learning algorithm accepts a single value for each attribute. The attribute can hold a conjunction of all *N* features, but the features themselves and any ordering of the features will be meaningless. An alternative approach (used by Magi) is to generate multiple examples, each with one of the features contained by the message body attribute.

The number of training examples increases with the use of other attributes which may contain more than one feature. The number of examples generated is given in 1.4:

$$Num\ of\ \ Examples = (a_1 + a_2 + , \ldots , + a_m) \tag{1.4}$$

where

a = number of possible features for a given attribute
m = number of attributes

An example of this is given in Chapter 5 (see *Feature Extraction*).

The rule generation module also produces attribute files. In order to generate rules, the learning algorithm needs lists of valid attributes and actions. These are generated at this stage before learning the rules. The attribute files are also used for filtering in classifying messages.

4.5 Classifying Messages

Mail on a UNIX operating system is automatically delivered to a users mailbox. It is also possible to send a copy to a process. This mechanism is used by the *classification engine*. When mail arrives for the user, a copy is sent to the classification engine, which then attempts to classify the message and propose an action in the execution log.

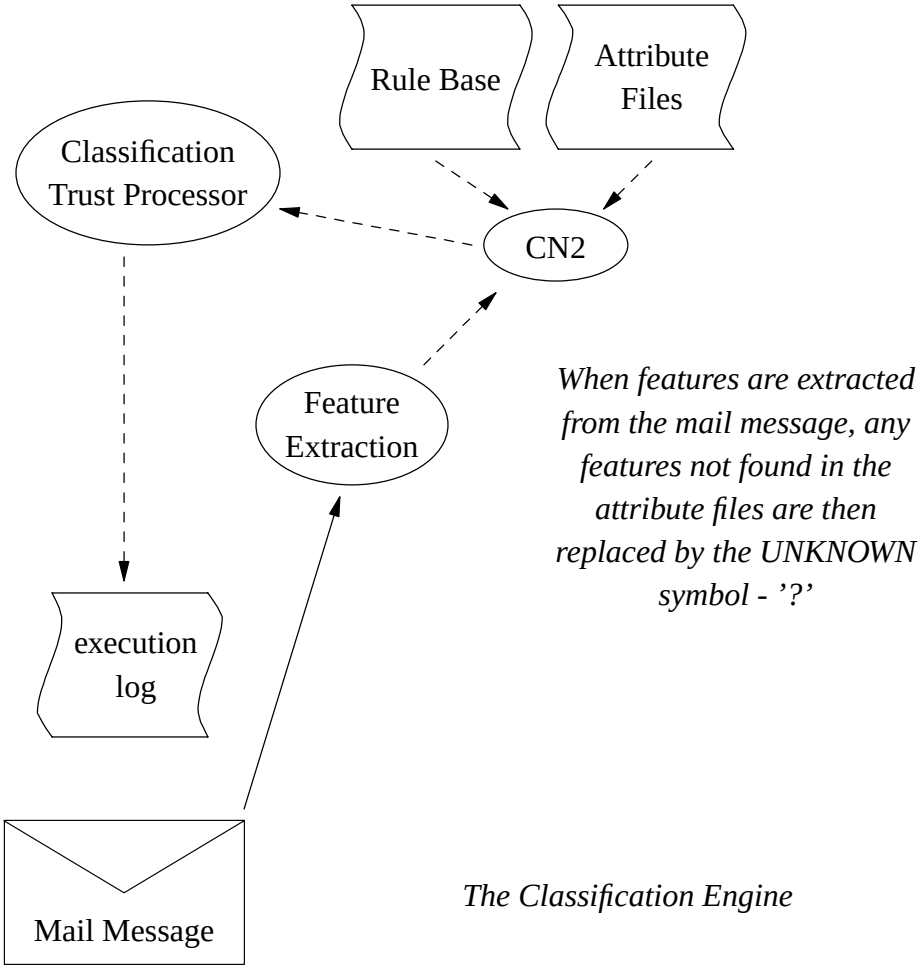


Figure 6 - The classification engine

In order to classify an arriving mail message, it needs to be broken down into features which are then applied to the rule-base. The features generated from the message undergo a further filter stage than when generating the rules. Each of the top occurring set of words is then checked to see if it occurs in the correct attribute file. If it cannot be found it is replaced with *UNKNOWN* (see figure 6). The reason for this is that the

classification engine recognises this symbol and no lookup is performed. If a feature cannot be found in the attribute files, then it will not exist in the rule-base.

Multiple testing examples are generated in a similar fashion to those generated for learning (see *Learning Rules* above). As in the training examples, every combination of attributes are generated for testing. Any attributes not found in the appropriate attribute file are replaced by *UNKNOWN*. Each combination is tested against the rulebase. All rules that fire are then processed to count the number of rules firing for each different action. For each action, if the number of firing rules is greater than the *trust* threshold, then the agent proposes the action for the appropriate message by adding it to the end of the *execution log*.

Because the number of features generated by the subject line can vary, the number of firing rules needs to be normalised. This is achieved by treating the *trust* threshold as a percentage of test examples firing. For example, the *trust* threshold may be set to 30%. For the agent to have confidence in an action for a given message, more than 30% of the test examples for that message must fire rules which give that action.

Chapter 5

Implementation Details

This chapter discusses considerations taken into account when implementing the mail agent Magi. The agent was developed to run on a UNIX platform, and is implemented using Bourne shell scripts and the language 'C'.

5.1 Changes to Xmail

The main requirement in writing the agent was to allow it to observe the user whilst allowing the user to work with the mail tool, and to allow it to interact directly with either the mail tool or the user. This poses a problem as asynchronous communication has to take place between the agent and both mail tool and user.

This problem has been nicely solved by *xmail*, written by M.Wagnitz [Wagnitz 1992]. Xmail is an X11 window based visual interface to the standard Berkeley *Mail* program (see figure 7). It is written in 'C' and uses the X toolkit with the Athena Widget Set. Not only does this set up and maintain a communication channel to Berkeley *Mail*, but it provides a modifiable graphical user interface. A single button and dropdown menu was added to the main interface to provide the user access to the agent.

Two main modifications were made to *xmail*. They were to observe user actions and to handle agent actions. Due to the design of the interface, different buttons were used for different actions. Each button causes a callback function to be executed. Calls were added to the save buttons to call the routine *ActionLog (actionlog.c)*. The design of this routine is very simple:

i. The *pre_feature* log file is opened for appending.

ii. The action is written to the log file

iii. *Mail* is queried for the appropriate mail message, which is then written to the file.

The code attached to the command button could then be executed.

The Deletion and Undeletion button created more of a problem. No log updates were performed until the mailbox was to be updated. All deletions by the user were also

performed by *Mail*. Berkeley *Mail* allows deleted messages to be undeleted, provided the mailbox has not been updated. Deletions were recorded internally in a buffer. This way undeletions would then cancel the corresponding deletion. When the mailbox was to be updated, the buffer was then traversed, and the remaining deleted messages were then added to the log.

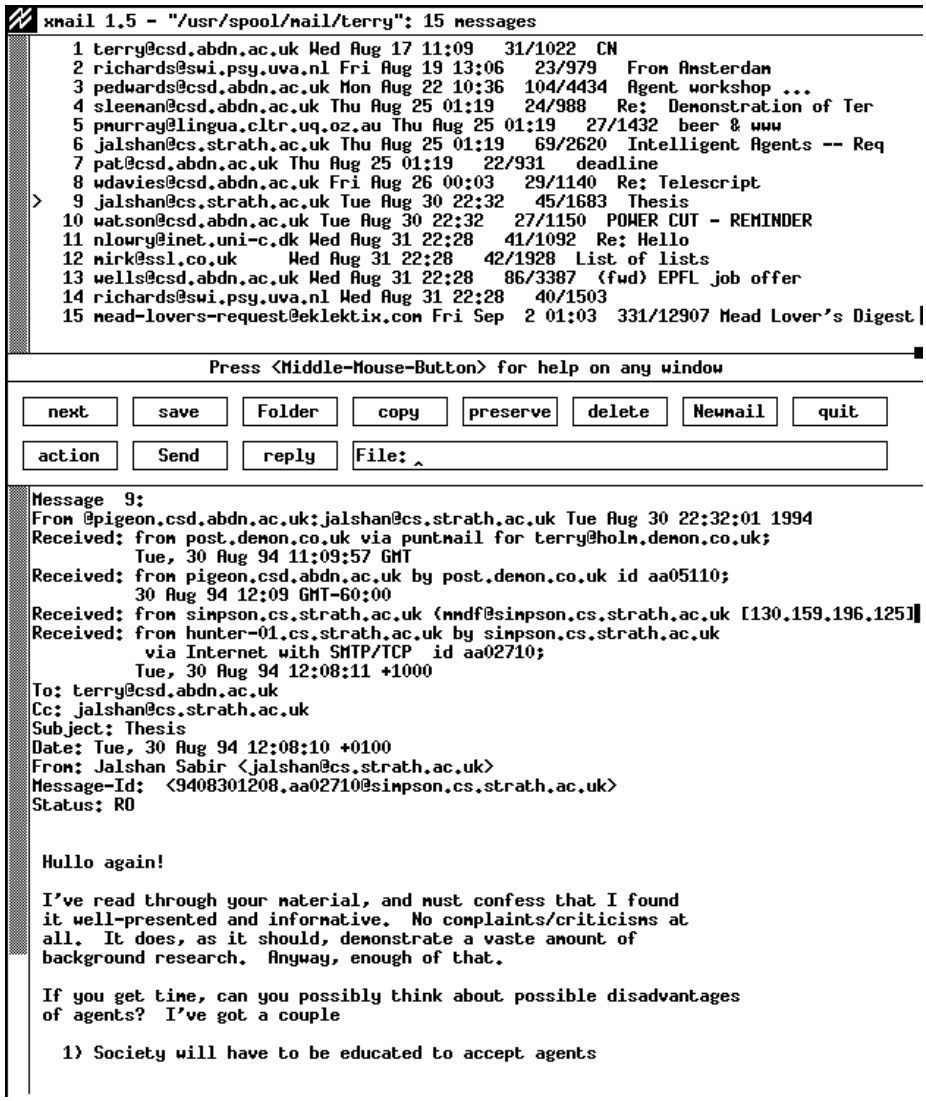


Figure 7 - The user interface

Handling agent actions involved more work. The classification engine appends proposed actions to the *execution log* which is then read by the action. Each entry comprises of the Message-Id and an action. An example of an entry can be seen in Figure 8. The Message-Id is used to identify the message within the mailbox, and is always sought from

the most recent message to the oldest.

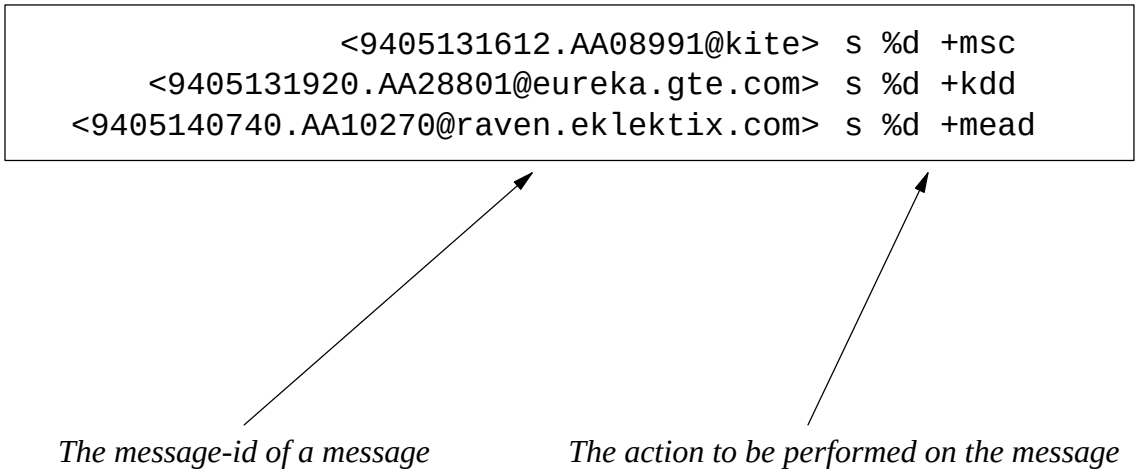


Figure 8 - An Execution Log Entry

This *execution log* is only loaded into the interface when the user selects the *action* button. The interface is designed so that the buttons on the main window can also act as dropdown menus. If the action button is clicked with the left mouse button, the *execution log* is loaded and executed. By clicking the button with the right mouse button a menu appears with four options:

- *action* - this performs the same action as clicking with the left mouse button. The reason for this is consistency; all menus have the same action for their top entry as the button itself.
- *Select Actions* - This invokes the action browser.
- *Log Actions* - This turns on agent observations and logging.
- *NoLog Actions* - This turns off agent observations and logging.

The last two menu entries allow the user to determine when the agent observes actions. This mechanism can be used to prevent atypical actions from being used as erroneous training examples. By default the agent will observe and log user actions.

When agent actions are performed, the agent looks up the Message-Id of a message-action pair in the current mailbox by requesting all the messages from *Mail* in turn, from the most recent to the oldest. The agent is actually seeking the message number of the

message with the matching Message-Id. Once this is found, it is used to complete the command (or action) which is then sent to *Mail*. The a summary of messages and actions performed on them is then presented to the user (see Figure 9).

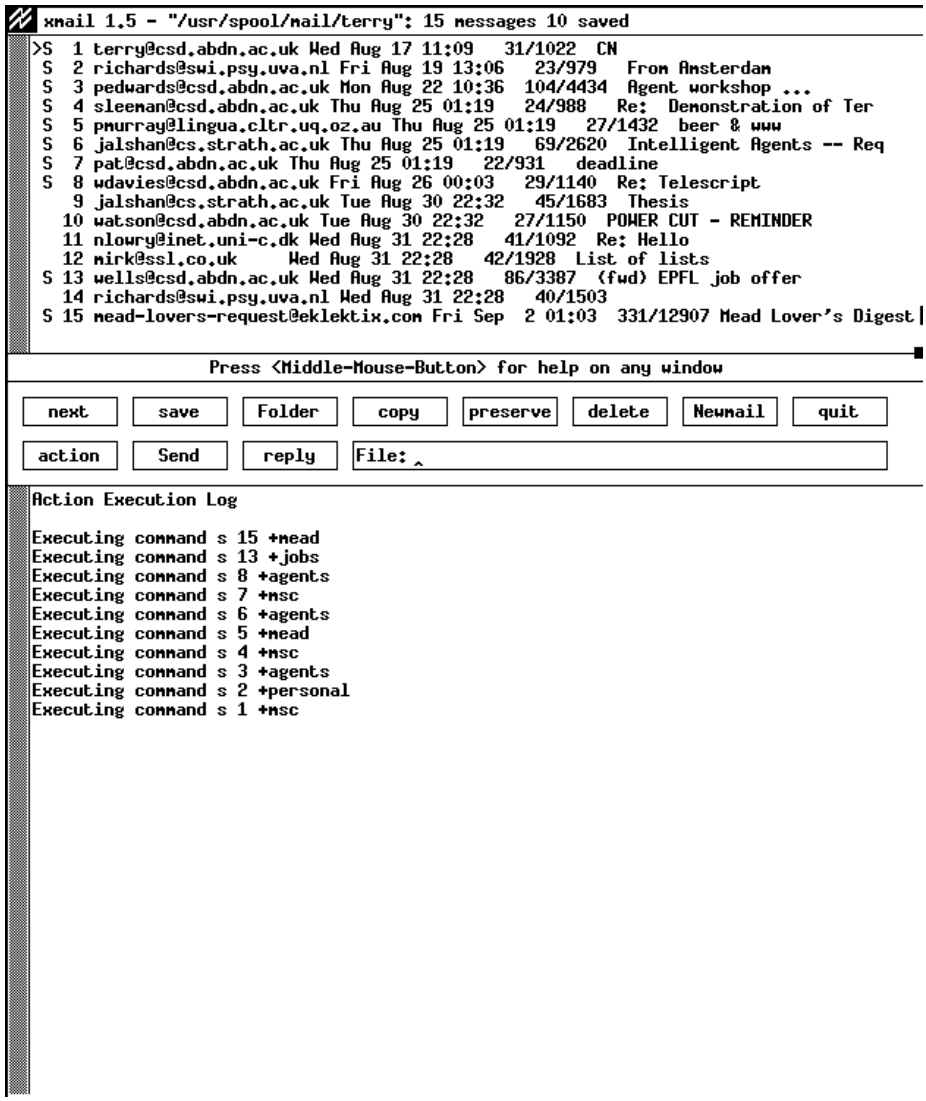


Figure 9 - Summary of agent actions

The action browser is a mechanism whereby the user can select message-action pairs and instruct the agent to display the message. The user can also select or de-select messages to be executed. This enables the user to provide both positive and negative feedback to the agent about its predictions.

Due to the lack of time at the end of the project, this browser was not completed.

5.2 Feature Extraction

This component lies at the heart of both the rule generation module and the classification engine. Written in 'C', it is responsible for parsing mail messages and extracting features from the message body. Both modules require similar behaviour, but subtle differences exist in use:

- *Rule Generation* - messages held in the *pre_feature* log are parsed. These also include the actions that were performed on them. All features extracted are used, coupled with the action.
- *Action Classification* - newly arrived messages are parsed so features can be applied to the rule-base. No action is provided, and hence none appears with the features. Attribute files are used to determine which features may appear in the rule-base; features not found in these files are replaced with the symbol *UNKNOWN*.

The message is parsed a line at a time. The "*From*", "*Subject*" and "*Message-ID*" fields are extracted from the header, along with action information if it exists. The subject line is also broken up into features similar to the message body. The algorithm used in parsing the subject line and message body is as follows:

i. *Periods '.' are converted to spaces.*
ii. *All uppercase characters are converted to lower case.*
iii. *Whitespace characters are recognised to delimit words. Whitespace is defined as a tab character, a space or a newline character.*
iv. *All non alphabetics are stripped from words.*
v. *Words are then looked up in the stoplist. If found they are ignored.*
vi. *The words are then stored in a tree data-structure. A frequency count for each word is maintained.*

Step (iv) was chosen for simplicity. It filters out punctuation. However the decision to strip numerics may cause a loss in potential features (such as the word "CN2").

The stoplist used was the one used in generating permuted indexes - */usr/lib/eign* which can be found on Sun UNIX platforms. This removes commonly occurring words such as "*the*" or "*and*". Appendix C contains the list of words found in this stoplist.

If features are being generated for the classification engine then an extra stage in the filtering takes place. Features not found in the attribute files generated by the rule generator are replaced by the symbol *UNKNOWN*. Test examples have two constraints imposed by the classification engine:

- a test example must contain a value for each attribute.
- only attribute values in the attribute files can be used.

The *UNKNOWN* symbol does not cause any rules to fire, but is special to the classification engine and so can be used to complete the test examples.

There may be times when identical examples will be generated, because of more than one *UNKNOWN* feature. These are preserved, as the number of times the rules fire is significant in determining action classifications.

The following is an example of a message and the features generated.

```
From: terry
Subject: The Tao of Pooh

For a good read, this book
about Pooh is a very good book.

    Read it.
```

The features generated are as follows:

From	terry
Subject	tao pooh
Body	read book pooh

All different combinations of the attributes will be generated as training or test examples for the rule generator or classification engine respectively. The examples shown below are for the classification engine. All the features, except *tao* can be found in the respective attribute files for each attribute. Hence, tao is replaced with the *UNKNOWN* symbol "?".

	Attributes	
From	Subject	Body
terry	?	read
terry	?	book
terry	?	pooh
terry	pooh	read
terry	pooh	book
terry	pooh	pooh

5.3 Rule Generation

Two sources of training examples are used to generate rules; the first contains dated examples used to generate previous rules, the second contains examples from observations made since the last rules were induced. Old training examples are kept for a limited time, and this set of examples is pruned to remove the oldest examples before new rules are generated.

The training examples are then parsed to build the attribute files. An attribute file contains all the values for a given attribute found in the training examples. These are required not only by the classification engine, but also by CN2, in order to induce rules.

The algorithm used is shown below:

i. Features are extracted from recent agent observations held in the *pre_feature* log. These features are used to generate training examples.
ii. Old training examples are pruned, so that examples used to generate the new rules reflect usage over a given time period.
iii. The training examples are parsed to generate the attribute files.
iv. The new training examples are date-stamped and prepended to the old training examples and saved for later use.
v. New rules are then induced from the training examples by CN2, which then writes these rules in a rule-base for later classification.

5.4 Message Classification

As new messages arrive, a copy is passed to the classification engine. It is then used to

attempt to predict how the user fill handle the message, such as deleting it or saving it. This is performed by converting the message into a set of test examples. Features are extracted from the message, and features not found in the attribute files are replaced with *UNKNOWN* symbols. A copy of the Message-Id of the message is retained whilst parsing the message, so the predicted action can be applied to the message by the user interface.

The CN2 algorithm was modified to perform classifications, by reading in a rule-base and then applying test examples to these rules. If a rule fires, then the test example with attributes is logged, together with the action resulting from the rule. This log is then processed to see how many and what actions fired. These actions are counted and compared to the *trust* threshold. Actions which rules fire more times than the value of this threshold are then stored in the *execution log*, along with the Message-Id.

Although CN2 generates ordered rules with a default action, this default not taken into account when classifying rules. This partially addresses the problem of over-generalisation, where the default attempts to catch any case not covered by the rules.

The algorithm used is shown below:

i. Features are extracted from the message. Those features which are not found in the attribute files replaced by "?".
ii. The features are used to generate a set of test examples. The Message-Id of the message is identified.
iii. The modified CN2 is then used to apply the test examples to the rule-base. Rules that fire generate the attributes of the examples that fire them and the resulting actions.
iv. The results of CN2 are counted and applied to a *trust* threshold. Those counts which are greater than this threshold are then saved to be proposed to the user.

It is possible for the test example set to vary in size depending on the message. Because of this, the *trust* threshold varies depending on the size of the test set, so that equal proportions of the sets are required to fire rules in order for the agent to propose actions to the user.

Chapter 6

Experimental Results

This chapter describes the testing of the feature extraction, classification and learning engines. It describes and evaluates the dataset used to test these modules. Each of the experiments are described in detail, graphs of the results are presented and these results are analysed. Detailed tables of results can be found in Appendix D.

Unlike many empirical case studies that compare different learning strategies within a specific domain (for example within a calendar management agent [Dent et al.(1992), Mitchell et al.(1994)] and in English Text to Speech Mapping [Diettrich et al.(1990)], this study makes use of a single learning mechanism within its domain. The aim is to see if the agent can correctly classify *some* mail messages accurately more than 50% of the time. Given this the following tests vary different operational parameters to explore the system and to determine if it does provide a viable solution.

6.1 The Dataset

In order to test out the agent's ability to learn from and classify mail actions, a large number of mail messages was required. Six mail categories were identified and messages were gathered over a four month period during the project (May 1994 - August 1994). They were sorted into six files, one for each category. Two further mail files taken from the authors archives were added to the dataset, providing eight different categories into which mail message could be sorted.

The number of messages and size of each category file varied. Each message was categorised purely on its contents. Factors regarding social or economic filtering [Malone et al. 1987], such as the size of the message, its author, the recipients etc were not taken into account. Because of this, the number of messages in each category varies.

Different types of mail message exist, such as special interest messages, organisational mail etc. Table 6.1 identifies some of these message types, and illustrate which of the categories in the dataset share these properties. The properties are defined as follows:

Category	digest	mailing list	organisational	specific interest	general interest
agents				□	
cure		□		□	
jobs			□	□	
kdd	□	□		□	
mead	□	□		□	
msc			□		□
personal					□
phd				□	

Table 6.1 Properties shared by the message categories in the dataset.

- *digest* - This is a mailing list where messages are sent to an individual who is responsible for collating and sometimes moderating messages about the given subject area. Each digest message will contain one or more contributing messages. The *From* field is consistent across each digest, and often there is little variation in the *Subject* field.
- *mailing list* - Normally automated, this is a server which forwards messages sent to it to recipients listed within a list. The *From* and *Subject* fields are those of the original sender and so will vary with each message.
- *organisational* - these are messages grouped because of some organisational structure, such as messages sent to a mail alias.
- *specific interest* - messages related to a specific theme, such as a music mailing list or special interest list.
- *general interest* - messages covering a broader range of interests.

The following is a description of each category:

- *agents* - this category contains various articles, and announcements about a specific topic; that of ‘agents’. This dataset is the smallest, having been collected for a short period, and contains messages mostly forwarded from a single source. Hence this

category is not expected to yield good results. *12 messages*

- *cure* - this category contains selected messages collected over a four year period. All the messages pertain to a music mailing list. *51 messages*
- *jobs* - this category contains various messages regarding posts available in academia. *24 messages*
- *kdd* - the messages in this category are in the form of digests from a mailing list. As only a single digest was received every month, a number of archived digests were added to the dataset. *15 messages*
- *mead* - again messages forming a digest from a mailing list. Messages were received at a rate of two to three a week. *29 messages*
- *msc* - these messages were sent to a departmental mailing list regarding various organisational matters. *26 messages*
- *personal* - messages in this category comprise of personal communications and cover a wide range of topics. *43 messages*
- *phd* - these messages, regarding a specific topic, originated from a handful or senders over the period of a year. This category also contains the largest number of mail messages. *64 messages*

6.2 Classifying different datasets

In order to evaluate the classification engine and the rule generation module, a test suite was developed. The algorithm is listed below:

i. For each message category file a percentage of messages, selected at random, are extracted for training the agent. These are processed to emulate the agent observing the user's actions, and are stored in the *pre_feature* log. The action is that of saving the message in a mail folder with the same name as the category file.

ii. Once messages from all the category files have been pre-processed and added to the *pre_feature* log, rules are then induced. Entries in the *training* log are removed, so that only the *pre_feature* training examples are used.

iii. For each message category file:

   i. The remaining messages not used in training the agent are then passed to the classification engine as incoming messages.

   ii. The messages are then classified. The *execution* log is processed, to investigate how many messages were classified correctly, and how many were mis-classified.

   iii. These results are appended to a test results file.

This test was repeated for training percentages from 20% to 80% in 10% steps. A lower percentage would provide insufficient training examples to train the agent, and a higher percentage would leave to few test examples to adequately test the rules.

Care should be taken when examining these limits for categories with low message counts such as *agents* and *kdd*; as little as a single message from these categories may have been used to train or test the agent at these boundaries.

A sample set of results is shown below. Coverage tables similar to those found in the rules are used to identify the categories of mis-classified messages. This information could be used in future work to discover categories which have a more than random overlap in features. For now the results were analysed to calculate the proportion of messages that are correctly classified by the agent, as opposed to the number of mis-classifications made.

```
Training with 50%
Thu Sep  1 12:49:44 BST 1994
./testdir/kl_agents 4 agents 4 [ 4 0 0 0 0 0 0 0 ]
./testdir/kl_jobs 10 jobs 2 [ 1 2 0 1 0 0 0 0 ]
./testdir/kl_kdd 6 kdd 6 [ 0 0 6 0 0 0 0 0 ]
./testdir/kl_msc 9 msc 1 [ 4 0 0 1 0 0 0 0 ]
./testdir/kl_cure 28 cure 12 [ 0 0 0 0 12 0 2 0 ]
./testdir/kl_mead 18 mead 18 [ 0 0 0 0 0 18 0 0 ]
./testdir/kl_personal 31 personal 3 [ 1 0 0 1 1 0 3 0 ]
./testdir/kl_phd 30 phd 21 [ 0 0 0 0 0 0 0 21 ]
```

Each entry contains the number of messages tested of that category, the category name, the number of correct classifications made and a table listing the total number of classifications made for that category file.

The CN algorithm was used with the evaluation algorithm set to *laplacian*, with a significance threshold of 0.0 and a star size of 5. These parameters have remained throughout all the testing described in this dissertation.

The original tests concentrated on features generated from the message body. No attributes were provided by the subject or from fields. The number of features $N = 10$ was arbitrarily chosen. The *trust* threshold, T was set to 2, so that more than two rules have to fire with the same action before the agent has confidence in the result.

The test was repeated 30 times, and from theses results averages were calculated. Two results, *proposed actions* and *effective actions*, are generated for each category. The *proposed actions*, or PA (shown as a solid line) represents the percentage of successful proposals made by the agent for a given category. The *effective actions* take into account mis-classifications made whilst classifying a given category.

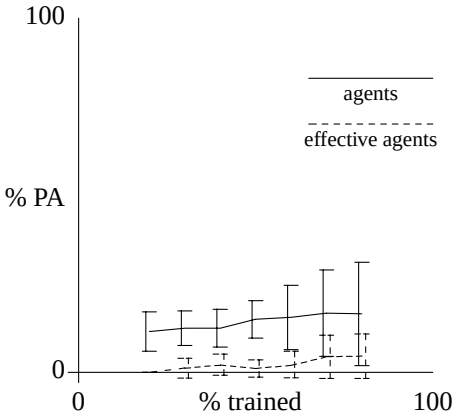
EA is calculated by the equation shown below (1.5)

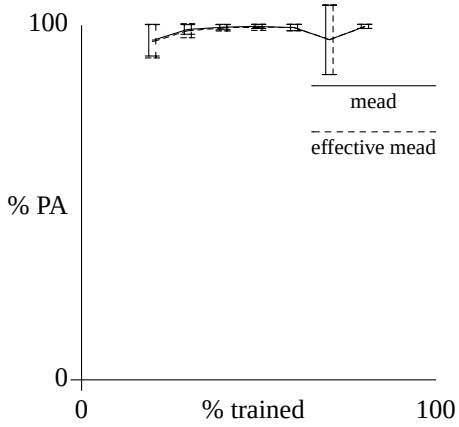
$$EA_c = 2PA_c - \sum_{i=1}^{m} PA_i \tag{1.5}$$

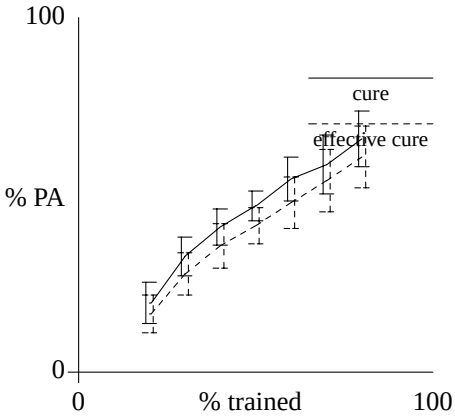
where EA_c = the *effective actions* value of category c, PA_c = *proposed actions* of category c, and the accuracy distribution $PA = (PA_1, \ldots, PA_m)$ is the *proposed action* value PA for each category. Negative *effective actions* are replaced with a value of zero. This measurement (shown as a dashed line) provides a better indication of the performance of the agent.

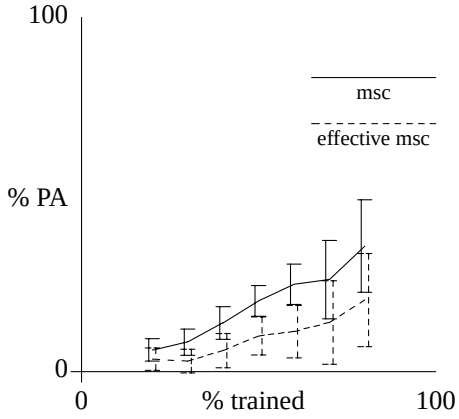
6.2.1 Results

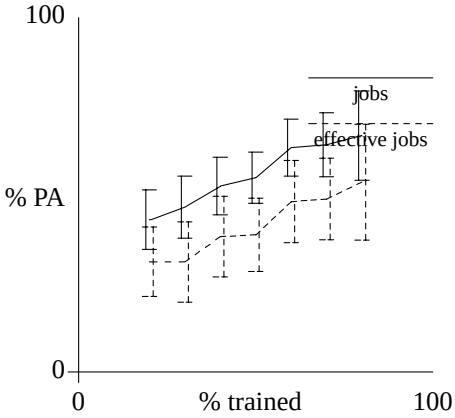
The following eight graphs show the percentage of correct classifications made by the agent in determining each category. Each graph corresponds to each of the categories tested. For each point the standard deviation has been calculated and is shown. Tables of results are then given on the following page and discussed.

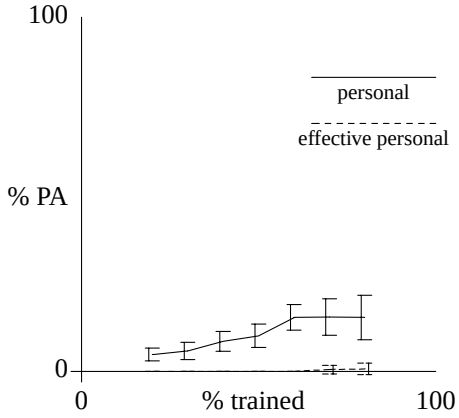

100
agents
effective agents
% PA
0
0
% trained
100

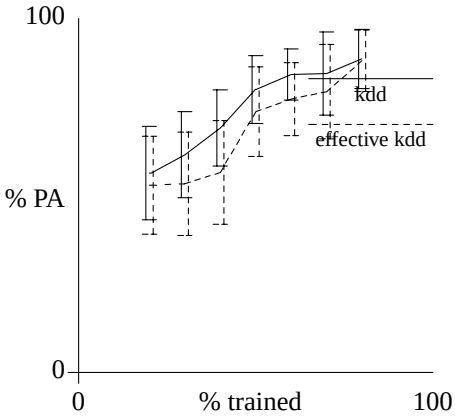

100
mead
effective mead
% PA
0
0
% trained
100

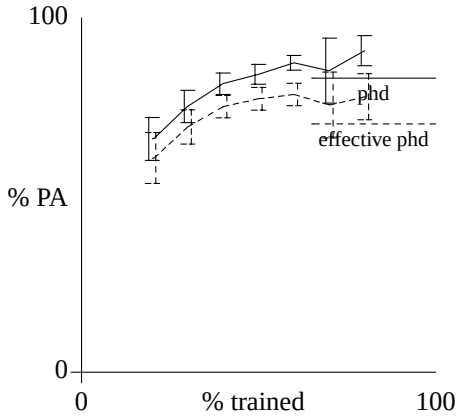

100
cure
effective cure
% PA
0
0
% trained
100


100
msc
effective msc
% PA
0
0
% trained
100


100
jobs
effective jobs
% PA
0
0
% trained
100


100
personal
effective personal
% PA
0
0
% trained
100


100
kdd
effective kdd
% PA
0
0
% trained
100


100
phd
effective phd
% PA
0
0
% trained
100

%	agents	cure	jobs	kdd	mead	msc	personal	phd
20	11.48(11.18)	19.51(11.73)	42.94(16.77)	56.29(26.34)	95.88 (9.02)	6.06 (6.33)	4.76 (3.58)	65.90(12.29)
30	12.41 (9.74)	32.60(10.95)	46.46(17.52)	61.51(24.25)	98.88 (2.74)	8.31 (7.45)	5.73 (4.92)	75.01 (9.07)
40	12.42(10.72)	40.94(10.28)	52.39(16.25)	69.04(21.51)	99.56 (1.48)	13.71 (9.22)	8.44 (5.52)	81.39 (6.08)
50	14.92(10.53)	46.84 (8.38)	54.79(14.57)	79.91(19.14)	99.75 (1.22)	19.94 (8.69)	10.03 (6.57)	84.06 (5.67)
60	15.48(18.17)	54.47(12.32)	63.27(16.12)	84.15(14.44)	99.45 (1.87)	24.63(11.44)	15.20 (7.19)	87.33 (4.17)
70	16.67(24.38)	58.55(16.65)	64.07(18.03)	84.42(23.52)	96.00(19.60)	25.95(22.07)	15.33(10.32)	85.13(18.36)
80	16.46(29.12)	65.82(15.76)	66.70(25.23)	88.53(16.70)	99.75 (1.22)	35.40(26.12)	15.20(12.60)	90.77 (8.50)

Percentage of Proposed or Predictive Actions, PA

%	agents	cure	jobs	kdd	mead	msc	personal	phd
20	0.00 (0.00)	16.39(10.58)	31.07(19.57)	52.86(27.59)	95.56 (9.42)	3.48 (6.33)	0.00 (0.00)	60.43(14.36)
30	1.14 (5.60)	27.64(11.86)	31.00(22.68)	53.31(29.24)	98.53 (3.96)	2.95 (6.74)	0.00 (0.00)	69.25 (9.68)
40	2.17 (5.95)	35.61(12.55)	38.13(22.75)	56.47(29.29)	99.34 (1.79)	5.91 (9.65)	0.00 (0.00)	74.96 (6.30)
50	1.00 (4.90)	41.30(10.21)	38.64(20.72)	73.63(25.28)	99.50 (1.70)	10.03(10.79)	0.00 (0.00)	77.19 (6.39)
60	2.13 (7.48)	47.77(14.44)	48.03(23.14)	77.24(20.59)	99.45 (1.87)	11.28(14.83)	0.00 (0.00)	78.37 (6.31)
70	4.33(12.27)	53.98(17.62)	48.74(23.06)	79.29(26.64)	96.00(19.60)	13.88(23.38)	0.50 (2.45)	75.40(18.58)
80	4.51(12.50)	60.70(17.46)	53.52(32.67)	88.03(17.28)	99.75 (1.22)	20.13(26.26)	0.67 (3.27)	77.73(12.96)

Percentage of Effective Actions, EA

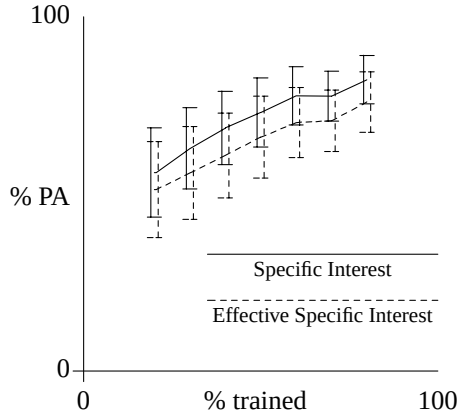
6.2.2 Discussion

The results indicate that regularities are being discovered within the training examples. For each of the categories, there is an increase in PA as the number of training examples increases. There is some increase in standard deviation towards the higher testing percentages; the increase becomes noticeable above 60%. This is probably explained by the smaller number of examples being tested per category, hence variation in results can increase. Both *agents*, *kdd* and to some extent *jobs* also had higher standard deviations overall than the other categories, though this can be explained by these three categories having the smallest number of examples for training and testing.

The results for the two general interest groups, *msc* and *personal* are poor. This could be an early indication that the agent may have difficulty in inducing rules for these categories. Of the remaining special interest categories, all but *agents* show reasonable performance. The category *mead* performed really well, despite the anomalous result at 70% which was probably due to to a rogue result.

The EA results show a similar picture, except that these predictions were lower. Standard deviations are also slightly higher overall in these results. This implies a number of misclassifications are being made, and that the *trust* threshold is set too low. The results from *agents* and *personal* indicate that classifications for these categories are mostly random, as there are approximately equal numbers of misclassifications as correct classifications.

The following table and graph show an overall percentage of correct classifications for all the specific interest categories with the exception of *agents*. Whilst this does not show the overall accuracy of the system, it does indicate that, for some categories the agent can begin to be a help to the user.



%	PA	EA
20	56.10 (25.26)	51.26 (27.10)
30	62.89 (22.94)	55.95 (26.16)
40	68.66 (20.74)	60.90 (23.89)
50	73.07 (19.50)	66.05 (23.08)
60	77.73 (16.46)	70.17 (19.83)
70	77.63 (14.06)	70.68 (17.31)
80	82.31 (13.64)	75.95 (17.04)

Overall Percentage of predicting special interest messages

6.3 Varying the *trust* threshold - *T*

The results so far have indicated that whilst many correct classifications are made by the classification engine, a number of bad classifications, or mis-classifications also occur. This is suggested by the difference between the EA and the PA values. One way of reducing mis-classifications is to increase the *trust* threshold. This would result in requiring more rules to fire for the correct classification, and less incorrect rules firing.

Three sets of experiments were performed. Each was identical to the test outlined in the above section, *Classifying different datasets*, except that the *trust* threshold varied. The values $T = 3$, 4 and 5 were tested, and compared to the above tests where $T = 2$.

Again, the evaluation algorithm set to *laplacian*, with a significance threshold of 0.0 and a star size of 5. Only features from the message body were used, and the number of features, N was left at 10.

6.3.1 Results

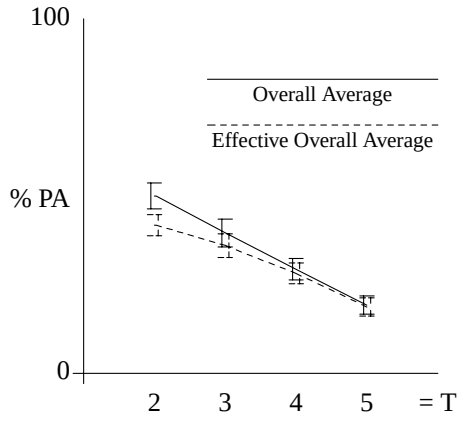
The results of these experiments, complete with graphs can be found in Appendix D.

6.3.2 Discussion

The results clearly indicate that by increasing *T*, the following two effects occur:

- the number of inaccuracies or mis-classifications fall, and
- the number of correct classifications drops.

These two phenomena are illustrated in the graph below. An overall average (for PA and EA) is calculated for each value of *T*. For each training percentage, an overall percentage of correct predictions is calculated across all categories (see results in Appendix D). These overall percentages are averaged out to give an average PA and average EA for each value of *T*. These values are plotted below. The *Proposal Accuracy* is calculated from these two values, and is an approximation of the percentage of correct classifications.



T	Overall PA	Overall EA	Proposal Accuracy (%)
2	50.01 (7.29)	41.77 (5.97)	83.52
3	39.56 (7.85)	36.01 (6.71)	91.03
4	29.33 (6.04)	28.14 (5.93)	95.94
5	19.22 (5.20)	18.72 (5.17)	97.40

Average PA and Average EA with respect to T

Results indicate that there is an initial decrease in mis-classifications, followed by smaller decreases as *T* increases, but that the decrease in correct classifications falls consistently by about 10% per increase in *T*. This indicates that $T = 3$ may be a good value for calculating agent confidence.

The EA value does not increase for *personal*, but a drop in percentage of proposed actions occurs indicating that predictions are unlikely to be made for this category.

6.4 Varying the number of message body features - *N*

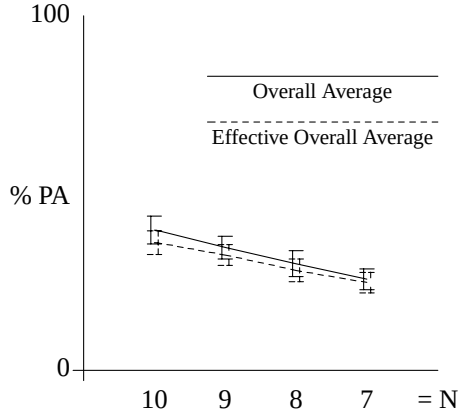
So far, the number of features extracted from the message body *N* has been set to 10. Tests on the decay of frequency in extracted features (see Appendix B) which were based on $N = 10$ indicated that there was little difference in frequency between the lowest frequency words considered. Because of this, the next three tests were concerned with varying this number of features. The test was performed with the values of *N* set to 7, 8, then 9.

6.4.1 Results

The results of these experiments, complete with graphs can be found in Appendix D.

6.4.2 Discussion

A similar trend occurs in reducing *N* as occurred in increasing *T* in the previous tests. As *N* is reduced, the percentages of correct classifications and mis-classifications fall. This is illustrated in the graph below. See *Varying the trust threshold - T* for details on calculating these values.



N	Overall PA	Overall EA	Proposal Accuracy (%)
10	39.56 (7.85)	36.01 (6.71)	91.03
9	34.60 (6.41)	32.50 (5.86)	93.94
8	30.06 (7.26)	28.16 (6.49)	93.67
7	25.65 (5.94)	24.74 (5.82)	96.45

Average PA and Average EA with respect to N

The above trend is more apparent when observing separate categories with higher proposal percentages (eg *phd* or *kdd*). The trend is less obvious in the above graph, due to the averaging effects of using the very low proposal percentages, which are relatively uneffected by varying *N*.

The trend is not as marked as that shown by varying *T*. Both trends indicate that there is some correlation between these two factors, and more work is needed here to determine ideal values for them.

6.5 Using the rest of the mail message

Once aspects of the message features had been explored, the use of header fields was investigated. The two fields of interest were the *From* and *Subject* fields. Three further tests were performed:

i. Incorporate the *From* field with message features

ii. Incorporate the *Subject* field with message features

iii. Incorporate both the *From* and *Subject* fields with message features.

Testing was performed with $N = 10$ and $T = 3$. The number of training examples varied with tests involving the *Subject* field. No investigation was made to see if this would effect the accuracy of classifications, however the classification engine takes into account the number of test examples when calculating confidence in a prediction from the *trust* threshold T (see *Chapter 5 - Message Classification*).

Tests involving just the *Subject* or *From* field were not performed, due to the small number of test examples that would be generated.

6.5.1 Results

The results of these experiments, complete with graphs can be found in Appendix D. Due to time, only 16 iterations of test *ii*, incorporating the *Subject* field were carried out.

6.5.2 Discussion

The results of these experiments were compared to those generated from using the message body only.

Using the *From* field

Almost all the categories benefited from the *From* field. Though timings and rule size are not discussed in this dissertation, these tests generated significantly smaller rule sets, and consequently both rule generation and classification appeared much faster than with any

other tests.

The only category adversely effected was the *jobs* category. An analysis of this dataset showed few regularities in the *From* fields, but this was also the case for categories *msc* and *personal*, both of which showed some improvement.

The performance of the two mailing list digests, *kdd* and *mead* improved radically when the *From* field is used. This was not surprising as each message from the mailing list came from the same source. Hence every training example will contain this same value for the *From* attribute, and provided that a rule exists which tests for this attribute value, rules will fire for each testing example.

The *agents* category also did surprisingly well. On analysing this small dataset it can be seen that nearly all the messages were forward from the same source.

The handling of the *From* field was very naive; the whole field was used with little parsing. More information could be extracted from this field if more advanced parsing were used, for example to recognise internet addresses in different forms (for example *<everson@COM.BBN>* and *<everson@bbn.com>*), extracting the senders full name, or the name of any forwarding agent (eg *Russel Winder <R.Winder@cs.ucl.ac.uk> (by way of jhunter (Jim Hunter))*).

Using the *Subject* field

Although there was an overall increase in actions proposed by the classification engine, the accuracy of these predictions fell dramatically when using the *Subject* field. Though this is shown in the accuracy graph below, it is very apparent from the results of the individual graphs for each category.

Again the digest categories, *kdd* and *mead* performed very well, due to having very similar features in the subject line for each message of the digest.

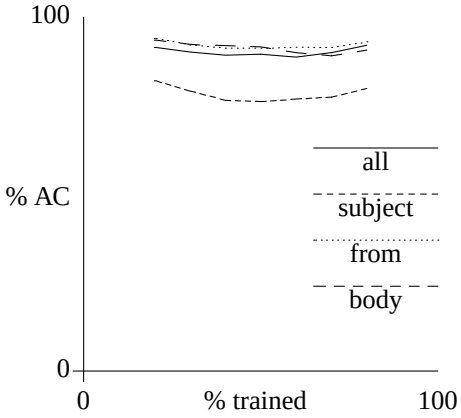
The small category *agent* performed better with the use of *Subject* features.

The confidence value calculated from the *trust* threshold varies linearly with the number of features tested. Due to time, no tests were performed to investigate the effects of increasing T or changing this confidence calculation with varying numbers of test examples. This work, had it been completed, may have explained whether this factor is

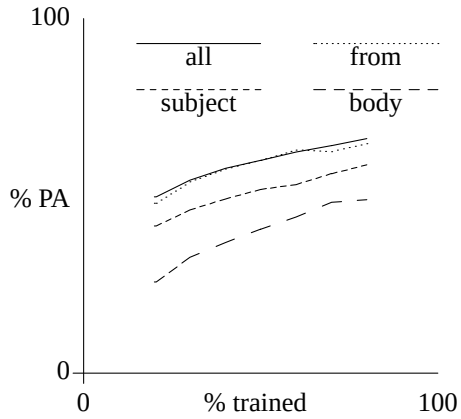
responsible for the drop in accuracy, or whether it is due to the *Subject* field itself.

Using both *From* and *Subject* fields

The graphs below show the percentage accuracies overall and percentage classifications overall for the agent when using different attributes from the mail message.



%	approximate % accuracy (AC)			
	all	subject	from	body
20	91	81	93	93
30	90	79	92	92
40	89	76	91	91
50	89	76	91	91
60	88	76	91	89
70	89	77	91	89
80	92	79	92	90



%	% of predictive actions (PA)			
	all	subject	from	body
20	49.71	41.49	47.93	25.69
30	54.41	46.00	53.92	32.63
40	57.79	49.08	57.60	36.84
50	60.06	51.82	60.04	40.61
60	62.36	53.17	62.93	44.03
70	64.17	56.27	62.50	48.18
80	66.19	58.78	64.78	48.95

The percentage accuracy is calculated from dividing the effective actions by the predictive actions. As the predictive action value is the number of times a prediction is made, and the effective action value takes into account mis-classifications, these values can be used to approximate accuracy values. Note that these values are an indication only and do not represent the actual accuracies.

Adding the *From* field or using both header fields appears to make little difference overall to the accuracy of predictions. However, adding *Subject* field attributes only to the message body attributes causes the accuracy to drop to values as low as 34% (calculated average accuracy from *cure* category).

Likewise, examining the percentage predictions made by the classification engine show that by applying the *Subject* field to message body features causes a reduction in the number of classifications made can be seen, whereas making use of the *From* field causes an overall increase in proposed actions.

6.6 Concluding remarks

The results show that a reasonable percentage of incoming mail messages can be classified with a high degree of accuracy. Making use of the *From* field improves both the number of predictions and the accuracy of predictions, whereas including features from the *Subject* field has no beneficial effect.

These results are promising as an agent need not classify every incoming message. However, any classifications made must be accurate. Issues such as coverage of rules [Mitchell et al. 1994], have not yet been investigated, but are strongly relevant to this aim.

Chapter 7

Conclusion

The work in developing Magi has demonstrated that it is possible to build an agent which is capable of learning email filtering rules from observations. The results show that features from the body of an email message can be utilised as well as more traditional features such as the subject or sender fields. The use of a trust threshold provides a high accuracy of classifications which can be made on some messages, thus providing the service of a *personalised assistant* to the user.

This work examines some of the considerations in designing and developing an interface agent. An agent which allows a user to access their mail through a widely used mail-tool was developed, and some of these considerations implemented.

The results suggest that applying information found in the *Subject* field does not necessarily improve performance as would be expected, whereas utilising the *From* field can improve both accuracy and number of predictions for the majority of mail categories.

Chapter 8

Future Work

The results of testing this agent raise many questions. Much is still unknown about the behaviour of this agent, such as the effects of varying the number of features *N* and the *trust* threshold when using header information. Other header information, such as the recipients or the date sent could also be used.

Whilst the experimental results demonstrate that the agent is capable of learning from a user and assisting the user in managing mail, little testing was performed on actual mail users. Also, the tests reflect the applicability of the agent to a single user (the author) who's classifications were used. As criteria for sorting mail may vary with each individual, more testing is required before any further claims can be made about the agent.

An important issue that was not investigated within this work was the effects of modifying the learning algorithm's parameters. The evaluation functions used the *Laplacian* error heuristic and the default significance threshold. Other heuristics and significance thresholds could have been explored. This may provide a means of reducing coverage. Coverage represents the proportion of training examples that can be *covered* by the rules. Tests on varying coverage [Mitchell et al. 1994] show that accuracy can increase as coverage is reduced.

The work on Magi proposed, but did not complete, a more advanced form of filtering which itself can adapt to the users needs through user feedback (by using the action browser). Other forms of filtering could also be explored, such as implementing thesauri or better means of correlating similar words due to case, tense etc (such as car & cars or catch & catching).

Organisational information could be used. This could improve performance on categories such as *msc* used in this report, where all the senders are members of an organisation. A variant on this has been explored by Maes [Maes 1994] whereby agents can communicate with each other to judge their model of the users. Mitchell [Mitchell et al. 1994] proposes the idea of co-operative learning, whereby rules are learned by pooling training examples from users. This could aid learning rules about organisational matters.

The learning and classification engines could be applied to other domains, such as USENET news readers or calendar managers (both have been explored to some degree, see Chapter 2). Agents could be made to communicate with each other across domains, thus gaining insights into the users requirements.

Chapter 9

References

[Baclace 1991] P.E.Baclace; Personal Information Intake Filtering. *Bellcore Workshop on High Performance Information Filtering*, Morristown, NJ, Nov 1991

[Baclace 1992] P.E.Baclace; Competitive Agents for Information Filtering. In *Communications of the ACM*, Dec 1992, Vol 35, 50-51

[Bains 1994] S.Bains; The Trillion-Bit Cube. In *New Scientist*, Aug 1994, Vol 143, 22-23

[Belkin & Croft 1992] N.J.Belkin & W.B.Croft; Information Filtering and Information Retrieval: Two Sides of the Same Coin? In *Communications of the ACM*, Dec 1992, Vol 35, 29-38

[Berners-Lee et al.] T.Berners-Lee, R.Cailliau, Ari Luotonen, Henrik Frystyk & A.Secret The World-Wide-Web. In *Communications of the ACM*, Aug 1994, Vol 37, 76-82

[Boose & Gaines 1989] Knowledge Acquisition for Knowledge-Based Systems: Notes on the State-of-the-Art. In *Machine Learning 4*, Kluwer Academic Publishers, 1989, 377-394

[Boswell 1990] R.Boswell; Manual for CN2 version 4.1. The Turing Institute, 1990

[Buchanan & Mitchell 1978] B.G.Buchanan & T.M.Mitchell; Model-Directed Learning of Production Rules. In *Pattern Directed Inference Systems*, Waterman & Hayes-Roth (Eds), 1978, 297-312

[Chin 1991] D.N.Chin; Intellegent Interfaces as Agents. In *Intelligent User Interfaces*, J.W.Sullivan & S.W.Tyler (Eds), ACM Press, 1991, 177-206

[Clark 1989] P.Clark; Functional Specification of CN and AQ. The Turing Institute. 1989

[Clark & Niblett 1989] P.Clark & T.Niblett; The CN2 Induction Algorithm. In *Machine Learning 3*, Kluwer Academic Publishers, 1989, 261-283

[DeJong 1982] G.DeJong; An Overview of the FRUMP System. In *Strategies for Natural Language Processing*, W.G.Lehnert & M.H.Ringle (Eds), Lawrence Erlbaum Associates, 1982, 149-176

[Denning 1982] P.J.Denning; Electronic Junk. In *Communications of the ACM*, Mar 1982, Vol 25, 163-165

[Dent et al. 1992] L.Dent, J.Boticario, J.McDermott, T.Mitchell & D.Zabowski; A Personal Learning Apprentice. In *AAAI-92 Proceedings, Tenth National Conference on Artificial Intellegence*, AAAI Press, 1992, 96-103

[Dietterich et al. 1990] T.G.Dietterich, H.Hild, G.Bakiri; A Comparitive Study of ID3 and Backpropagation for English Text-to-Speech Mapping. In *ML-90, Proceedings of the Seventh International Conference on Machine Learning*, B.W.Porter & R.J.Mooney (Eds), 1990, 24-31

[Foltz & Dumais 1992] P.W.Foltz & S.T.Dumais; Personalized Information Delivery: An Analysis of Information Filtering Methods. In *Communications of the ACM*, Dec 1992, Vol 35, 51-60

[Goldberg et al. 1992] D.Goldberg, D.Nichols, B.M.Oki & D.Terry; Using Collaborative Filtering to Weave an Information Tapestry. In *Communications of the ACM*, Dec 1992, Vol 35, 61-70

[Jacobs & Rau 1990] P.S.Jacobs & L.F.Rau; SCISOR: Extracting Information from On-Line News. In *Communications of the ACM*, Nov 1990, Vol 33, 88-97

[Kay 1984] A.Kay; Computer Software. In *Scientific American*, Sep 1984, Vol 251, 41-47

[Kozierok & Maes 1993] R.Kozierok & P.Maes; A Learning Interface Agent for Scheduling Meetings. In *Proceedings of the 1993 International Workshop on Intellegent User Interfaces*, ACM Press, 1993, 81-88

[Leiner 1994] B.M.Leiner; Internet Technology. In *Communications of the ACM*, Aug 1994, Vol 37, 32-33

[Maes 1994] P.Maes; Agents that Reduce Work and Information Overload. In *Communications of the ACM*, Jul 1994, Vol 37, 31-40

[Maes & Kozierok 1993] P.Maes & R.Kozierok; Learning Interface Agents. In *AAAI-93*

Proceedings, Eleventh National Conference on Artificial Intellegence, AAAI Press, 1993, 459-465

[Malone et al. 1987] T.W.Malone, K.R.Grant, F.A.Turbak, S.A.Brobst & M.D.Cohen; Intellegent Information-Sharing Systems. In *Communications of the ACM*, May 1987, Vol 30, 390-402

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

[Michalski et al. 1986] R.S.Michalski, I.Mozetic, J.Hong, N.Lavrac; The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In *AAAI-86 Proceedings, Fifth National Conference on Artificial Intellegence*, AAAI Press, 1986, 1041-1045

[Miller et al. 1990] G.A.Miller, R.Beckwith, C.Fellbaum, D.Gross & K.Miller; Introduction to Wordnet: An On-Line Lexical Database. Set of five papers supplied with *WordNet* distribution. 1990

[Miller & Drexler 1988] M.S.Miller & K.E.Drexler; Markets and Computation: Agoric Open Systems. In *The Ecology of Computation*, B.A.Huberman (Ed), North-Holland, 1988, 133-176

[Mitchell et al. 1994]; T.Mitchell, R.Caruana, D.Freitag, J.McDermott & D.Zabowski; Experience with a Learning Personal Assistant. In *Communications of the ACM*, Jul 1994, Vol 37, 81-91

[Quinlan 1986] J.R.Quinlan; Induction of Decision Trees. In *Machine Learning 1*, Kluwer Academic Publishers, 1986, 81-106

[Ram 1991] A.Ram; Interest-based information filtering and extraction in natural language understanding systems. *Bellcore Workshop on High Performance Information Filtering*, Morristown, NJ, Nov 1991

[Ram 1992] A.Ram; Natural Language Understanding for Information-Filtering Systems. In *Communications of the ACM*, Dec 1992, Vol 35, 80-81

[Riloff & Lehnert 1992] E.Riloff & W.Lehnert; Classifying Texts Using Relevancy Signatures. In *AAAI-92 Proceedings, Tenth National Conference on Artificial*

Intellegence, AAAI Press, 1992, 329-334

[Rumelhart et al. 1986] D.E.Rumelhart, G.E.Hinton & R.J.Williams; Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing Vol 1*, D.E.Rumelhart & J.L.McClelland (Eds), MIT Press, 1986

[Salton & McGill 1983] G.Salton & M.J.McGill; *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983

[Sheth 1994] B.D.Sheth; A Learning Approach to Personalized Information Filtering. MSc Thesis, Department of Electrical Engineering and Computer Science, MIT, Jan 1994

[Stadnyk & Kass 1992] I.Stadnyk & R.Kass; Modeling Users' Interests in Information filters. In *Communications of the ACM*, Dec 1992, Vol 35, 49-50

[Stanfill & Waltz 1986] Towards Memory-Based Reasoning. In *Communications of the ACM*, Dec 1986, Vol 29, 1213-1228

[Wagnitz 1992] M.C.Wagnitz Xmail manual, supplied as part of the *xmail* distribution. 1992

[Weinstein 1992] S.Weinstein; The Elm Filter System Guide, Datacomp Systems Inc. Oct 1992

[Weiss & Kapouleas 1989] S.M.Weiss & I.Kapouleas; An Empirical Comparison of Pattern Recognition, Neural Nets and Machine Learning Classification Methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1989, 788-793

Appendix A

Example of the Advanced Filter

This example demonstrates the advanced filter described in Chapter 4. Due to time this was not implemented in the agent, but remains as future work to be applied to the agent. Values given in this example are arbitary and are used to help explain the filter. In the implemented version, they would be determined empirically.

Imagine that the knowledge base contains, among others, the following five entries:

Feature	Fitness
yeast	67
honey	50
above	31
knowledge	45
fire	2

and that the threshold where features become unhelpful was 30. The feature *knowledge* did not exist prior to building the most recent set of rules. Because of this the *fitness* value is set to a standard starting value. In this example, this starting value is 45. The feature *fire* has a *fitness* value less than the helpfull threshold. This feature will filtered out for the classification module and rule generation module.

All four features above the helpfull threshold exist within the rules. The following example shows four rules which illustrate this. See the documentation of CN2 [Boswell 1990] for a description of the rules and *coverage tables* shown below. The rules below have been simplified for brevity.

Each action defined in the attribute files has an index into the coverage table. Each entry indicates how many training examples are covered by that rule. The first entry in the table refers to the action "*s %d +mead*" whereas the second to "*s %d +kdd*".

The feature *above* is less specific than the other three features. Three training examples were responsible for the action in that rule, but the feature was also found in other training examples for different actions. Whilst this may mean that the rule may cause

mis-classifications, it may still fire to generate correct classifications.

```
...
IF    feature = yeast
THEN  action = "s %d +mead"  [5 0 0]
ELSE
...
IF    feature = honey
THEN  action = "s %d +mead"  [3 0 0]
ELSE
...
IF    feature = knowledge
THEN  action = "s %d +kdd"  [0 1 0]
ELSE
...
IF    feature = above
THEN  action = "s %d +kdd"  [1 3 1]
ELSE
...
```

A sample of the rule-base

A message arrives and the action is classified as "*s %d +mead*". Three features found in the message are *yeast*, *honey* and *above*. A score is then calculated based on the frequencies of these features in the message. Preliminary studies on frequency decay indicate that this decay is logarithmic (See appendix B), so a simple equation is used to calculate the score. The frequencies are normalised across all features used in classification and the log of this percentage used. This equation is presented as part of the example, and may not be the equation finally used in the implementation, but illustrates the calculation of the score.

$$Score = (int) \ln \left(\frac{frequency^{i}}{\sum frequency} * 100 \right)$$

The following table illustrates the frequencies for the individual features and their calculated scores. The sum of all frequencies used here is 35.

Feature	Frequency	$\frac{frequency^i}{\sum frequency} * 100$	Score
yeast	12	34	4
honey	20	57	4
above	3	12	2

As this message has been classified correctly, the three features are rewarded. This is done by adding the score for each feature to the feature's *fitness*. The knowledge base now looks like this:

Feature	Fitness
yeast	71
honey	54
above	33
knowledge	45
fire	1

Note that the *fitness* value for the feature *fire* has been decremented. This occurs with every classification. Once the *fitness* falls to zero the feature will be removed from the knowledge base. This mechanism is used to allow features to be reused at a later date. Whilst this feature may lead to incorrect classifications at the present time, this may not be the case in the future.

Another message arrives. This time the action is classified as "*s %d +kdd*". The feature *above* was the only feature responsible for this classifiction. However this is a mis-classification. The table below shows the newly calculated score. The sum of all frequencies was 18.

Feature	Frequency	$\frac{frequency^i}{\sum frequency} * 100$	Score
above	18	100	5

The feature *above* will now be penalised by subtracting this score from its *fitness*. As the knowledge base now shows, this *fitness* value has fallen below the helpfull threshold. This feature will now be filtered out in subsequent filtering. As with the feature *fire*, its *fitness* will be decremented at each classification.

Feature	Fitness
yeast	71
honey	54
above	28
knowledge	45

It is interesting to note that the feature *fire* has now been removed from the rule-base. This is due to its *fitness* value falling to zero at the last classification. This feature will now be included for classification or rule generation if it appears in future messages.

Appendix B

Frequency Decay in Extracted Features

Feature Extraction used within Magi is based on word frequencies within a body of text. The top *N* occuring features are used in classifying actions for incomming messages and generating new rules. Because of this, the rate of frequency decay was investigated.

The results of this test were used as a basis for the calculations used in the advanced filtering. The number of features used, *N*, was set to 10 for this test. Each action classification file (see Chapter 6) was tested.

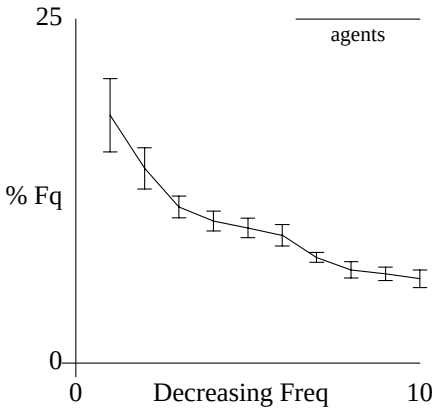
For each test, the top *N* features are extracted complete with frequency counts. These frequency counts are then normalised using equation (1) to give percentages for each feature. These are averaged across all messages to give the graphs shown below.

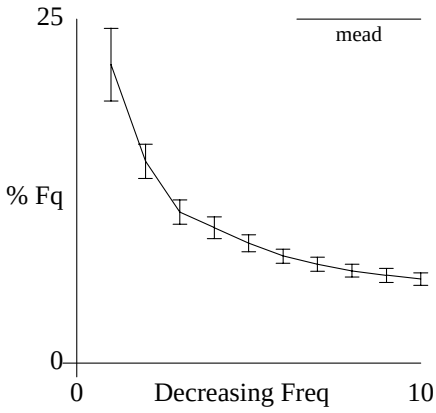
$$\%\ Fq = \frac{frequency^{i}}{\sum frequency} * 100 \qquad (1)$$

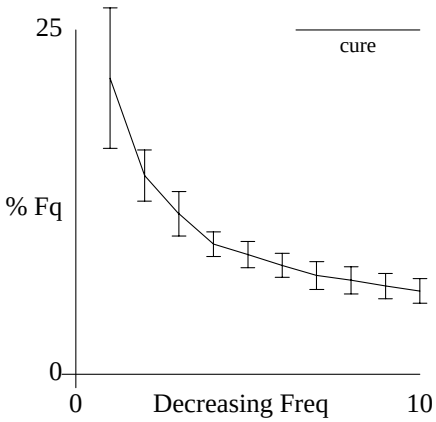
It is expected that increasing *N* will cause the curve to flatten, as extra features will become less frequent. As the frequency falls, there is a greater chance the features will occur in other action classifiaction messages.

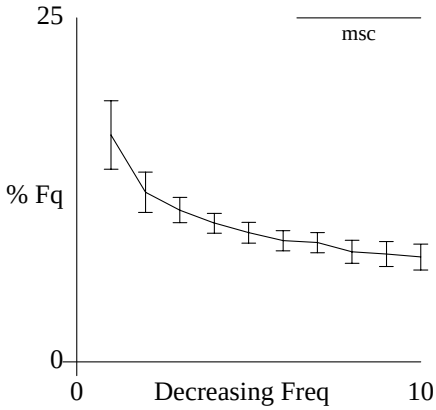
These results indicate that there is little difference in frequency between the lowest frequency words. Due this, tests were performed with the testbed, with the value of *N* varying from 7 to 10. The effects of varying this value is discussed in Chapter 6.

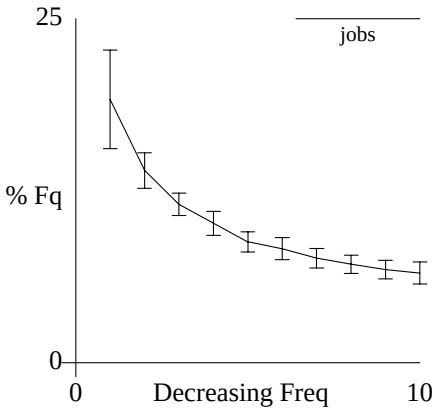
The graphs for each mail category used in the testsuite are shown below.

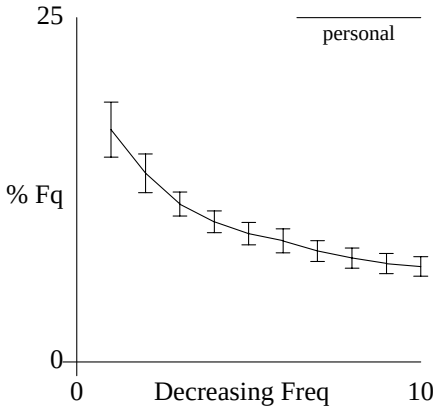

25
agents
% Fq
0
0
Decreasing Freq
10

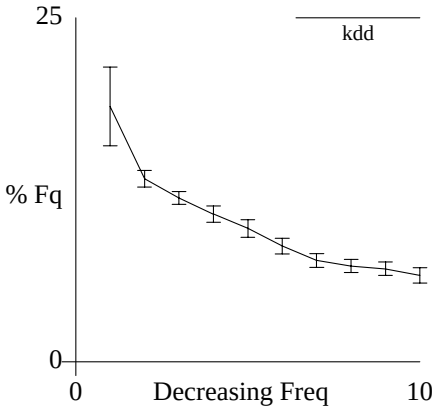

25
mead
% Fq
0
0
Decreasing Freq
10

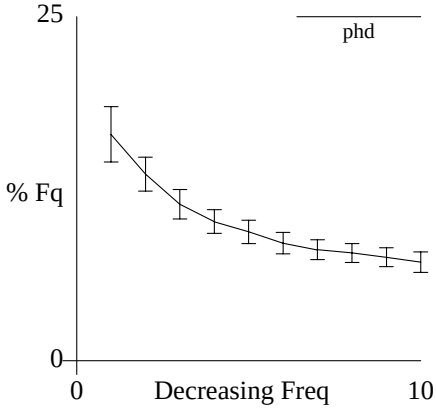

25
cure
% Fq
0
0
Decreasing Freq
10


25
msc
% Fq
0
0
Decreasing Freq
10


25
jobs
% Fq
0
0
Decreasing Freq
10


25
personal
% Fq
0
0
Decreasing Freq
10


25
kdd
% Fq
0
0
Decreasing Freq
10


25
phd
% Fq
0
0
Decreasing Freq
10

Appendix C

/usr/lib/eign

This Appendix contains the list of words in the file /usr/lib/eign which was used as a stoplist to filter out common words from message bodies.

the	of	and	to	a	in
that	is	was	he	for	it
with	as	his	on	be	at
by	i	this	had	not	are
but	from	or	have	an	they
which	one	you	were	her	all
she	there	would	their	we	him
been	has	when	who	will	more
no	if	out	so	said	what
up	its	about	into	than	them
can	only	other	new	some	could
time	these	two	may	then	do
first	any	my	now	such	like
our	over	man	me	even	most
made	after	also	did	many	before
must	through	back	years	where	much
your	way	well	down	should	because
each	just	those	people	mr	how
too	little	state	good	very	make
world	still	own	see	men	work
long	get	here	between	both	life
being	under	never	day	same	another
know	while	last	might	us	great
old	year	off	come	since	against
go	came	right	used	take	three

Appendix D

Complete results

This appendix contains the full set of results generated and used within the report.

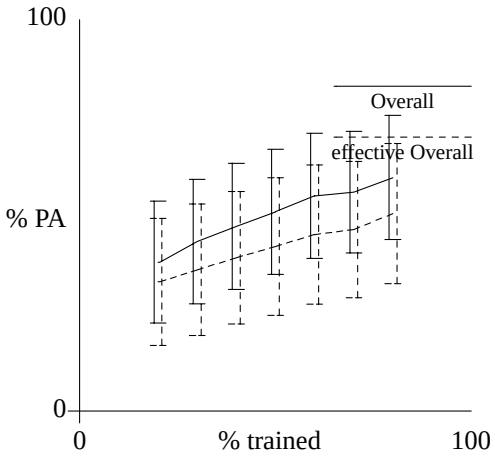
D.1 Results - Classifying different datasets

%	agents	cure	jobs	kdd	mead	msc	personal	phd	Overall
20	11.17(11.30)	20.60(11.37)	41.26(18.90)	60.35(26.14)	96.84 (7.63)	7.05 (7.31)	3.99 (3.34)	63.08(12.15)	38.04(31.10)
30	13.73(11.33)	32.44(12.01)	45.59(18.12)	65.07(24.77)	98.97 (2.51)	10.56 (9.97)	5.28 (4.59)	74.73 (9.49)	43.30(31.77)
40	13.08(12.22)	40.00(11.12)	50.62(17.68)	71.14(19.94)	99.71 (1.23)	14.71(11.85)	7.49 (5.02)	80.79 (7.69)	47.19(32.25)
50	17.02(11.16)	45.01 (9.98)	53.00(16.63)	79.36(17.51)	99.83 (1.01)	20.20(12.16)	9.19 (6.53)	83.46 (6.58)	50.88(31.89)
60	17.21(17.20)	53.20(11.53)	59.35(17.69)	85.59(14.34)	99.63 (1.56)	24.19(12.16)	12.94 (7.45)	87.85 (4.30)	54.99(31.97)
70	18.29(22.80)	57.16(15.92)	61.75(18.50)	86.41(21.46)	97.22(16.43)	24.50(20.70)	15.08(10.42)	86.97(15.78)	55.92(31.04)
80	17.50(28.18)	63.48(14.99)	63.34(26.97)	92.04(14.89)	99.82 (1.04)	34.58(25.75)	15.19(11.98)	91.30 (7.95)	59.66(31.77)

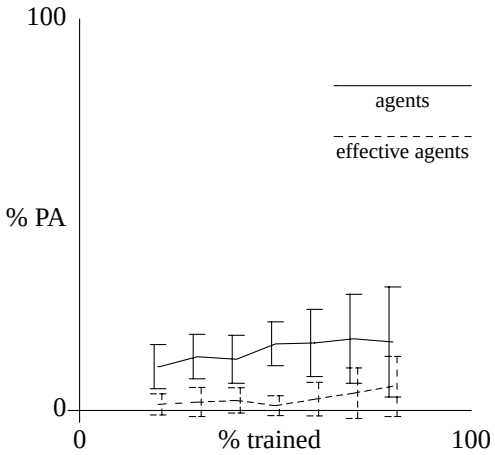
Percentage of Predictive Actions, PA

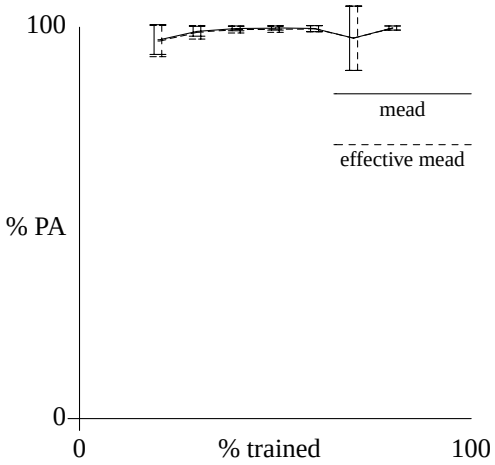
%	agents	cure	jobs	kdd	mead	msc	personal	phd	Overall
20	1.55 (5.43)	17.34(10.98)	30.02(21.80)	56.29(26.52)	96.51 (8.02)	3.95 (7.05)	0.00 (0.00)	58.16(13.53)	32.98(32.38)
30	2.12 (7.42)	27.60(12.80)	30.11(24.20)	56.70(28.99)	98.73 (3.45)	4.30 (8.17)	0.00 (0.00)	69.08(10.46)	36.08(33.56)
40	2.55 (6.50)	34.71(12.96)	36.70(23.94)	59.11(26.90)	99.42 (1.67)	6.88(11.51)	0.00 (0.00)	73.93 (8.80)	39.16(33.88)
50	1.22 (5.12)	39.79(11.29)	36.66(23.27)	72.37(23.44)	99.50 (1.67)	10.53(12.42)	0.00 (0.00)	76.18 (8.58)	42.03(35.09)
60	2.88 (8.57)	46.63(13.92)	43.95(23.21)	77.43(25.02)	99.63 (1.56)	11.35(13.76)	0.00 (0.00)	78.83 (6.55)	45.09(35.56)
70	4.40(12.95)	51.63(18.20)	47.17(23.45)	80.99(27.72)	97.22(16.43)	12.65(20.14)	0.62 (2.59)	76.41(16.45)	46.39(34.83)
80	6.13(15.30)	57.41(17.27)	49.85(33.48)	90.76(15.88)	99.82 (1.04)	20.79(27.25)	0.46 (2.74)	78.18(13.09)	50.43(35.75)

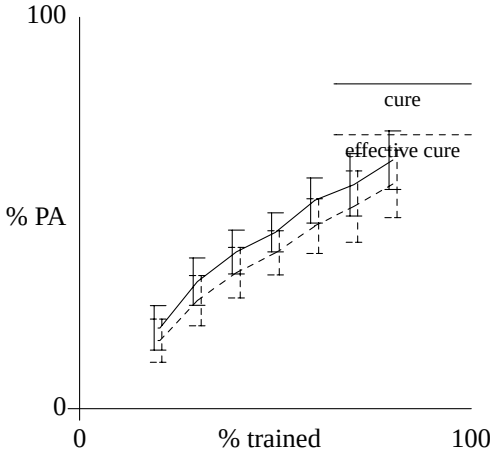
Percentage of Effective Actions, EA

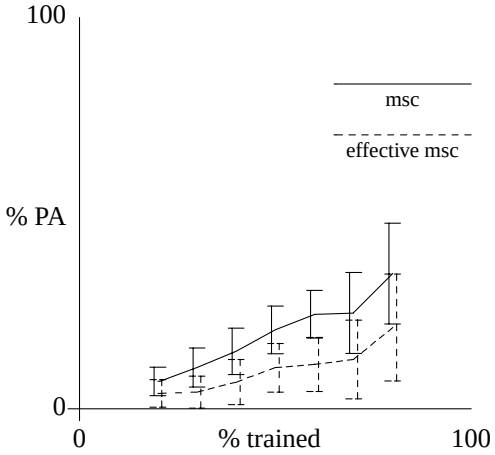


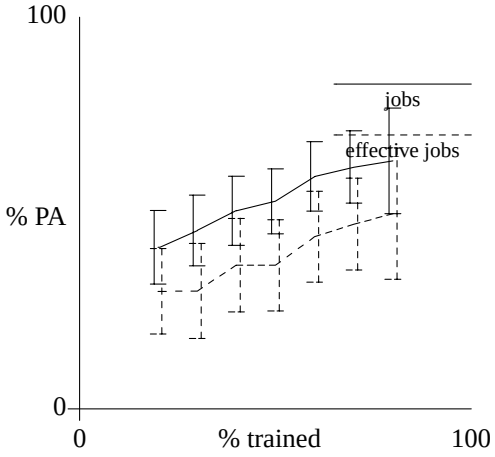
See	*Chapter 6 - Classifying different datasets*
N =	*10*
T =	*2*
Features	*message body*
Subject	*no*
From	*no*
Iterations	*30*

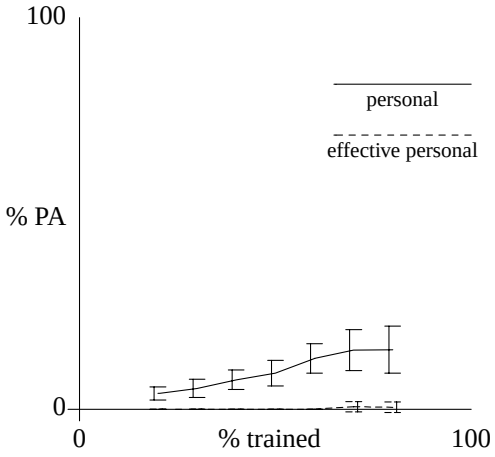

100
agents
effective agents
% PA
0
0
% trained
100

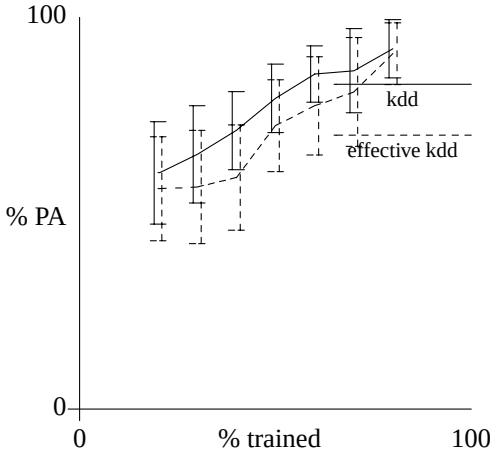

100
mead
effective mead
% PA
0
0
% trained
100

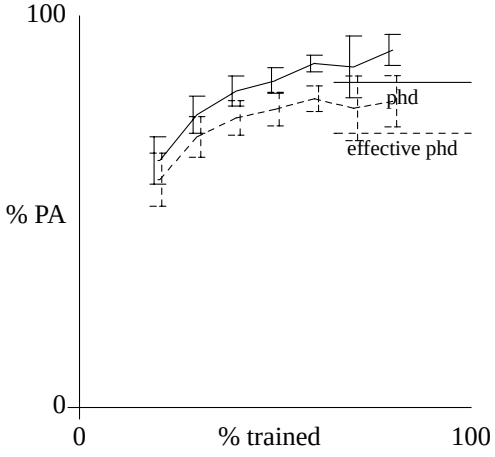

100
cure
effective cure
% PA
0
0
% trained
100


100
msc
effective msc
% PA
0
0
% trained
100


100
jobs
effective jobs
% PA
0
0
% trained
100


100
personal
effective personal
% PA
0
0
% trained
100


100
kdd
effective kdd
% PA
0
0
% trained
100


100
phd
effective phd
% PA
0
0
% trained
100

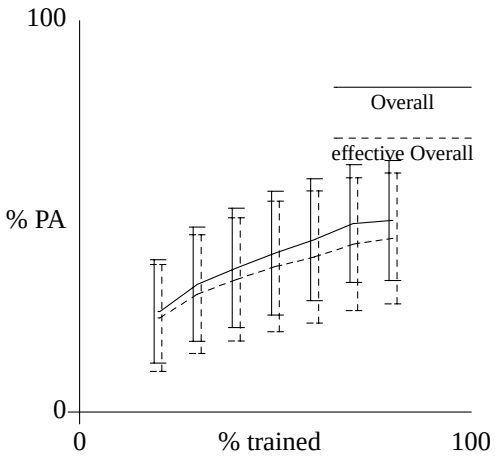
D.2 Results - Varying the trust threshold T = 3

%	agents	cure	jobs	kdd	mead	msc	personal	phd	Overall
20	4.21 (6.73)	9.18 (7.08)	20.93(17.67)	35.15(27.59)	84.69(15.48)	5.43 (5.58)	2.71 (1.83)	43.20(11.94)	25.69(26.39)
30	7.97 (9.87)	15.14 (8.24)	28.02(17.57)	50.19(27.20)	92.49 (6.06)	8.43 (6.92)	3.15 (2.02)	55.67(10.77)	32.63(29.18)
40	6.77 (8.11)	20.37 (8.36)	38.57(14.22)	59.38(24.49)	94.36 (5.62)	10.41 (7.27)	3.17 (1.93)	61.67 (9.65)	36.84(30.50)
50	8.92(11.44)	25.35 (8.10)	41.65(17.24)	69.91(17.49)	96.16 (4.67)	11.96 (9.12)	4.41 (2.90)	66.53 (9.87)	40.61(31.59)
60	13.39(20.16)	30.53(11.29)	47.07(15.44)	74.30(21.13)	97.08 (4.33)	15.34(10.25)	5.73 (3.93)	68.77(12.64)	44.03(31.15)
70	18.83(21.80)	37.20(12.11)	53.00(18.96)	77.67(22.45)	97.03 (5.16)	20.06(14.77)	8.51 (5.83)	73.16(10.07)	48.18(30.10)
80	16.38(27.55)	40.22(13.27)	58.31(26.39)	76.11(24.79)	97.46 (6.72)	14.64(20.77)	12.81(11.39)	75.66(13.09)	48.95(30.64)

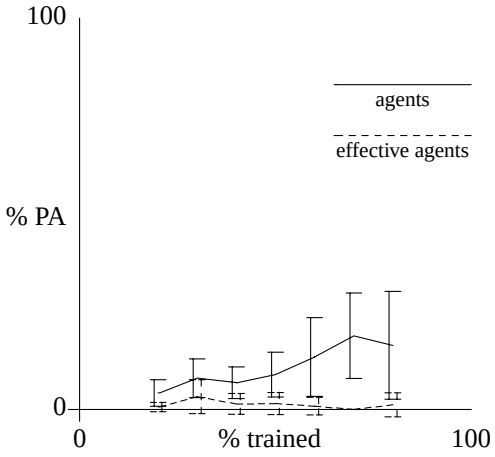
Percentage of Predictive Actions, PA

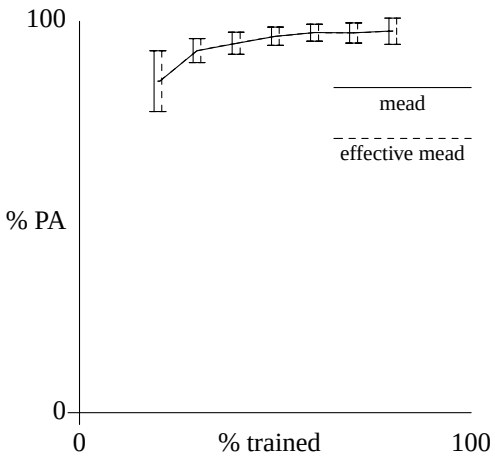
%	agents	cure	jobs	kdd	mead	msc	personal	phd	Overall
20	0.64 (2.38)	8.71 (7.08)	16.66(18.09)	34.31(28.09)	84.69(15.48)	4.30 (5.19)	0.00 (0.00)	42.88(12.29)	24.02(27.26)
30	3.25 (8.66)	14.68 (8.32)	21.55(16.93)	47.70(28.60)	92.49 (6.06)	5.99 (6.98)	0.00 (0.00)	55.22(10.90)	30.11(30.32)
40	1.39 (5.31)	19.74 (8.45)	31.24(14.99)	55.21(29.13)	94.36 (5.62)	8.01 (7.99)	0.00 (0.00)	61.05 (9.81)	33.87(31.49)
50	1.50 (5.65)	25.11 (8.37)	34.13(19.65)	67.44(20.53)	96.16 (4.67)	7.59 (9.04)	0.00 (0.00)	65.55(10.52)	37.18(33.33)
60	0.83 (4.49)	30.05(11.27)	39.02(19.84)	72.63(21.44)	97.08 (4.33)	10.07(10.59)	0.22 (1.20)	66.85(12.98)	39.60(33.83)
70	0.00 (0.00)	36.75(12.24)	46.88(22.77)	75.06(24.38)	97.03 (5.16)	14.86(15.62)	0.94 (2.84)	71.69(11.14)	42.90(33.98)
80	1.15 (6.08)	39.16(13.79)	50.80(31.56)	75.00(25.91)	97.46 (6.72)	13.55(20.81)	5.51(10.25)	72.29(16.59)	44.37(33.42)

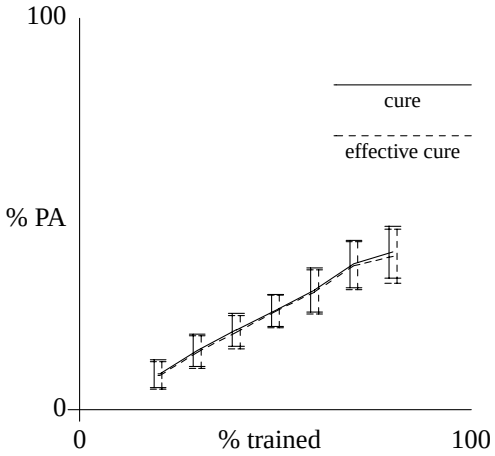
Percentage of Effective Actions, EA

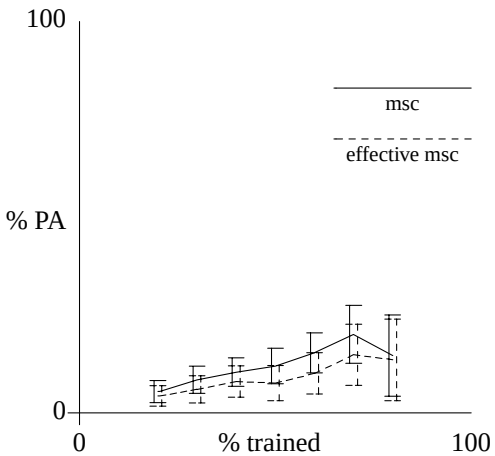


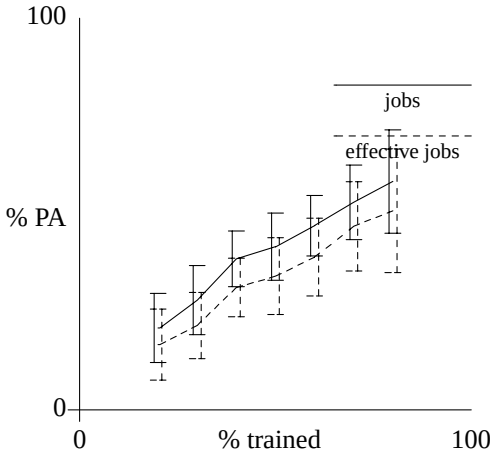
See	*Chapter 6 - Varying the trust threshold T*
N =	*10*
T =	*3*
Features	*message body*
Subject	*no*
From	*no*
Iterations	*30*

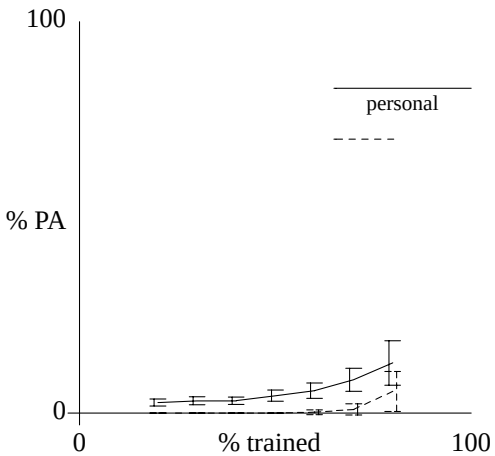

100
agents
effective agents
% PA
0
0
% trained
100

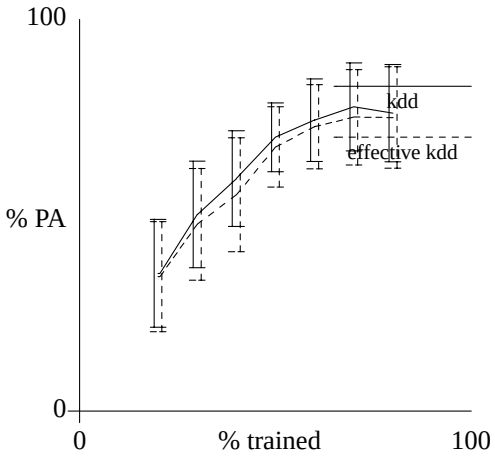

100
mead
effective mead
% PA
0
0
% trained
100


100
cure
effective cure
% PA
0
0
% trained
100


100
msc
effective msc
% PA
0
0
% trained
100


100
jobs
effective jobs
% PA
0
0
% trained
100


100
personal
% PA
0
0
% trained
100


100
kdd
effective kdd
% PA
0
0
% trained
100