Scrutable Programming By Demonstration for Email Management

Alex C. P. Lai
University of Sydney
Sydney, NSW, Australia
Email: imax_lai@hotmail.com

Abstract

Information overload is an increasing problem. A significant contributor is the large amount of email that people receive. It will be valuable if users can have assistance in managing email. The first stage in such a process is the classification of mail messages, which need to be treated alike into groups.

The IEMS [4] work is based upon machine learning for defining the rules. This project presumes that the user wishes to make use of this approach but then should be easily able to tune the rules. We also assume users want to be able to scrutinize the whole process so that they feel in control of the filtering rules or other mechanisms used to predict the classification of their mail.

1. Introduction

This paper describes the IEMS (Intelligent-Electronic Mail Sorter) [4] project which has the broad goal of improving our understanding of how to build systems which can assist users in managing email. In particular, we discuss work on automated support for classifying messages into appropriate folders. Choice of folder may depend on many factors including aspects such as the sender and nature of the email. For example, email from your supervisor may be filed into your "supervisor" folder.

Users can be assisted in the task of classifying email if they make use of filtering rules available within many widely used mail interfaces such as Netscape Messenger and Microsoft Explorer. These rules can be expressed in terms of strings appearing in different parts of an email message. To handle an email item, the rules are evaluated in order and the first rule that applies to the item triggers the email client to move the message into the associated folder.

The difficulty with rules is that the process of a rule is cognitively demanding and there is a real, potentially unacceptable risk of misfiling mail. Generally, users seem to avoid customizing software [8] [6]. In a recent study of user's management of email, the authors observe "Most of our users (17 interviewees, or 60 percent) say they don't use filters. Several simply haven't figured out how to use them, suggesting that either filters need to be simpler to use or that they are not that useful"

The motivation for rules is based on first, a belief these rules will relatively easy for end users to understand and modify, second, a suspicion that learning methods alone are not an adequate solution for categorization problems of this type. It seems likely that instead some mix of automatically and manually constructed classifiers will be necessary to account for the fact that both the user's interests and the distribution of message change (sometimes quite rapidly) over time. In many cases, the users themselves might have a better idea of the rules that would be appropriate for the applications they would like to build, so it would be useful for them to be able to specify new rules. For instance, at the time of this writing, the rules above may be accurate for messages I have received over the last few months; however at some point it will certainly become appropriate to modify it by replacing "Tutorial No: 05" with "Tutorial No: 06" and "Tutorial No: 5" with "Tutorial No: 6".

Our solution is to build a SPBD as an intelligent rule explorer that reduces the cognitive burden and the time required for easily understanding and customizing rules, to solve email classification into folder automatically in sort time. As we know rules make decisions based on a small number of keywords. Rules do not base classification decisions on word frequency, only on the presence or absence of a word. A problem with PBD has always been how to represent the rules to users, and how the users can come to feel in control the whole processes is becoming our major problem.

In this paper we will introduce our approach to solve the above problem. We use a system where the users can accept the rules given or view a pop up interface which gives them the opportunity to scrutinize the rule, and how it was derived, and then to modify it or to leave it as predicted by our system. It offers ease of use and flexibility to cater to users who wish to modify rules.

2. Previous Works

Most of the current systems have used simple rule-based inferencing for their generalizations. For example, the early Peridot system, developed in 1987 to create widgets by example uses about 50 hand-coded rules to infer the graphical layout of the objects from the example [9]. Each rule has three parts, one for testing, one for feedback, and one for the action. The test part checks the graphical objects to determine whether they match the rule. For example, the test part of a rule that aligns the centers of two rectangles checks whether the centers of the example rectangles are approximately centered. Because these rules allow some sloppiness in the drawing, and because multiple rules might apply, the feedback part of the rule asks the user whether the rule should be applied. For example, Peridot might ask something like: " Do you want the selected rectangle?" If the user answers yes, then the action part of the rule generates the code to maintain the constraint. The subsequent systems have used similar mechanisms, though often without the explicit list of rules we used in Peridot. For example, Tourmaline, which formats documents form example, contains rules that try to determine the role of different parts of a header in a text document, such as section number, title, author, and affiliation, as well as the formatting associated with each part. The results are displayed in a dialogue box for the user to inspect and correct.

Results have shown PBD has very high performance to solve classification problems based on rule-based inferencing, from different users (especially novices) by a series of very simple actions. As we know the purpose of PBD's characteristic is that it is easily understood. This characteristic led us alternative approach to encourage the users to solve email classification problem.

A variety of approaches have been taken to address the problem of automating email classification. Most of these can be split into two groups: filtering junk email; and general classification of email. At first glance, one might presume email classification was simply a special case of text categorization. However, even the seemingly similar work on the Reuters-21578 dataset is quite different from learning how to predict an individual user's classification of their own mail.

In any instance, it is desirable that any learning algorithm should produce useful results quite quickly, with small amounts of data: in our case, the learning is for a single user's filtering preferences and it is desirable that rules for automating this should be learnt from small numbers of example.

Further, the classification task may change with time. Changes in classifications might be due to changes in the user's activity: for example, a user who teaches a course in programming in one semester may not be involved in that type of activity in the next semester. This affects the task of a learner since it needs to recognize such changes. There are also many other changes that affect classifications. For example, if the user's supervisor changes or other personnel at work change their roles, a learner will need to adjust its classifications.

We note two other important aspects of this domain: user differences and differences in the difficulty in learning to predict the categorization for different mail classes. We know that different people use quite different mail management strategies, as noted, for example in [6]. We would expect that it is easier to learn the classifications applied by some users than would be the case for others.

On the matter of the varying difficulty of learning an individual users' different mail classes, Machine Learning and Information Retrieval approaches have demonstrated good performance can be achieved on spam/junk email. For example, SpamCop [10], using a Naïve Bayes approach achieved accuracy of 94%. Sahami *[14]* applied a Bayesian approach and achieved precision of 97.1% on junk and 87.7% on legitimate mail and recall of 94.3% on junk and 93.4% on legitimate mail. Katirai (1999) used a genetic classifier and its best run overall achieved a precision of 95% and a recall of 70%. Androutsopoulos *[1]* compared Naïve Bayes with a keyword approach. The keyword approach uses the keyword patterns in the anti-spam filter of Microsoft Outlook 2000 (which they believe to be hand constructed). They reported the keyword approach achieved precision of 95% and the Naïve Bayes approach 98%. On recall the corresponding performance was 53% and 78%. Androutsopoulos *[1]* also explored a memory based learning approach IiMBL (a simple variation of the K-Nearest Neighbour) showing similar performance to a Naïve Bayes classifier. Provost *[12]* also evaluated Naïve Bayes, comparing it with the RIPPER algorithm, showing 95% accuracy after learning on just 50 emails while RIPPER reached 90% accuracy only after learning on 400 emails.

One fairly strong result is described by Cohen *[3]*. He used the RIPPER learning algorithm to induce rules and reported 87%-94% accuracy. He also explored TF-IDF, and achieved 85%-94%. He observed that the rule based approach provided a more understandable description of the email filter. The iFile naïve Bayes classifier [13] was made available to several users to test on their own mail and this gave 89% accuracy. Grutlag (2000) assessed a Linear Support Vector Machine (SVM), reporting results from 70% to 90% correct and with the Unigram Language Model, 65% to 90%. They compared this against TF-IDF where they achieved 67% to 95%, depending on the store of email used.

The more general categorization task achieves weaker results than the levels achieved for two-category spam filtering employed by various researchers. Agent [2] explored learning in a two-class case, this time 'work' versus 'other'. Boone used a hybrid approach with TF-IDF to learn useful features and then both neural networks and nearest neighbour approaches. This gave 98% accuracy on

a datase where the standard IR approach had 91% accuracy.

This poorer result for general classification is unsurprising: useful email classification involves a user defining the class or classes within which they want to store a piece of email and this is a far less well defined task than distinguishing spam. When users make these classifications, there are many complex issues which define the process. For example, some users classify mail on the basis of the time by which they need to act upon it. Some classify mail according to the broad subject area as it relates to their work. In fact, the results summarized above seem very high for any realistic scenario where the user might have modest numbers of mail items in many of their mail folders.

The work described above does suggests that automated classification should be able to operate usefully in helping users create classification mechanisms for their email. The above work also indicates that some folder classifications will be far easier than others. It seems fruitful to explore approaches that can learn at least some classifications quickly. Even more importantly, it seems likely that a useful learner should be able to tell the user how well it performs so the user can decide whether that level of performance is good enough.

Previous work also suggests the need to build these classification mechanisms into an interface which proposes classifications rather than automatically acting on the mail. For example, the work on MailCat [15], using a TF-IDF approach initially had error rates of 20% to 40%. Since this was considered unacceptable, they took a different approach: MailCat recommended its three best predicted folders so that the user could archive email to one of these with a single click. This improved performance to acceptable levels.

Since several approaches seem to achieve good results in some studies, we can afford to explore the usefulness of a range of approaches that are simplest to explain to the user. Then the user should be able to understand any proposed classification rule and maintain a sense of control. This is the direction taken by Pazzani *[11]* who reported a study where users where asked to assess their preferences for different approaches for representing email filtering rules.

Our work is similar to previous work as PBD approach, and machine learning for defining the rules. The significant part of PBD is allowed user to explore the whole process and adjust the rules defined by the machine learning such as TFIDF, Sender, Keywords, DTree, and Naïve Gayes. After processed, it performs prediction as the presence or absence of a word.

3. Overview System

During our exploration, we found that there were many types of email client around the world, most of which come with email filtering (Microsoft Outlook, Hotmail, etc). Unfortunately, not many users apply this technology to deal with email classification problems. This result has been found in previous research. On the other hand, there are many researchers who believe that rule is a best solution to the issue of junk and email classification. Thus, a method to encourage users to customize and understand filter rules is fast becoming essential.

Cohen *[3]*, found that the problem with the use of keyword-spotting rules is based on first, a belief these rules were relatively easy for end users to understand and modify, and second, a suspicion that learning methods alone are not an adequate solution for categorization problems of this type. It seems likely that instead, some mix of automatically and manually constructed classifiers will be necessary to cater for the fact that both the user's interests and the distribution of messages change (sometimes quite rapidly) over time. For instance, at the time of this writing, the rule set above may be accurate for messages I have received over the last few months; however, at some point it will certainly become appropriate to modify it by replacing "Assignment No: 05" with "Assignment No: 06" and "Assignment No: 5" with "Assignment No: 6".

Our solution to the above problem is to combine two existing techniques: PBD and machine learning for defining the rules. In previous approaches in PBD, users demonstrate algorithms to the computer by operating the computer's interface just as they would if they weren't programming. The computer records the user's actions and can then reexecute them later on different inputs. PBD's most important characteristic is that it is accessible to everyone. PBD is not much different from or more difficult than using the computer normally. This characteristic led us to consider PBD as an alternative approach to infer the rules. This is a portion of work from the iems approach is based upon machine learning for defining the rules. We presume that the user wishs to make use of this approach but then should be easily able to adjust rules. We also presume the user wants to be able to explore whole process so that they feel in control of the filtering rules, or other mechanisms used to predict the classification of their mail.

Learners considered

The system of IEMS *[4]* has implemented in JAVA 1.3.1, which allows it to cross over to different platforms. We have explored the whole system, identified some of the most important classifications which include class of Gui, Store, RunLeanrer, LearnerSender, LearnerKeyword, LearnerTFIDF, MultiSet, and InvertTable. The Gui classification is to perform the graphical user interface, and convert user inputs to the relevant system event; The Store class is used to maintain the store

of all the messages, including training and classification; RunLearner is used to control which training algorithm is to be used, ie. Sender, keyword, or TFIDF algorithm; The class of LearnerTFIDF is going to train the set of folder used for learning and classification using the TFIDF algorithm. The classify function uses the standard first order classifier and places mail into proper folder. The induce function is going to train the set of folders used for learning. The reason function explains in word the reason for the particular classification, such as the particular mail message contains some keywords, that should be classified into a particular folder. The above functions must work with invertTable class; The class of InvertTable works in conjunction with the LearnerTFIDF class. When the "Next" button is enabled, this class evaluates the SIM4 similarity score of each of the folder with the next message and guesses the folder with the highest score for that message. The MultiSet class keeps track of the count of each word in a given message. It uses a HashMap to store "word as integer v.s. count" entries. Used by the various learner classes; The class of LearnerKeyword is going to train the set of folders used for learning and classification. The classify function uses the standard first order classifier with the hypothesis induced by the learner. It should place mail message into the proper folder. The reason function explains in word the reason for the particular classification; The class of LearnerSender is going to train the set of folders used for learning and classification. The main purpose is to look at the sender of email messages. A rule for each sender is created and this rule places anything new from that sender into the most commonly used folder they had been previously placed. The classify function uses the standard first order classifier and places mail into proper folder. The induce function is a simple hypothesis based on the Sender to achieve the goal. Note that the hypothesis is given in first order form. The reason function shows the clause as reason for the particular classification.

We would like to show a very high level description of a new algorithm as below, which would be able to accomplish our goal. See the Figure 1.

Figure 1. High level description of a new algorithm (SPBD)

Once a user has read a message, there are two possible courses of action. If they are happy with the classification, they can simply click on the Archive button. This is at the top left of the screen. This moves the message into that folder. In the case of the current message shown in Figure 2, the Archive button would move it to the PChardware folder.

Figure 2.

The other possibility is that the user is not happy with the classification. In that case, the user simply selects the MoveTo button followed by the name of the folder in the left panel. This moves the message to the correct folder. Meanwhile, it pops up the scrutable interface which tries to explain the rules particularly in highlighting some keywords, and waiting user to adjust the rules if they feel confident. This task is trying to encourage users to understand and adjust rules. In the case of the current message shown in Figure 3.

Figure 3.

This interface should help users to understand and customize filter rules faster, as well as encourage users to get involved with email classification tasks. If the system makes the correct classification, the user simply accepts it with a single click. If the system is wrong, the user does the sort of classification task (particular in adjusting rules) they would have had to do anyway. This should significantly reduce the cost of creating a filter rule while improving the accuracy for email classification.

4. Empirical Results

Below we describe the experiments that have been done so far. First, we conducted a 20 minute experiment and compared experience and non-experience users to see whether they could undertake the same task (creating a rule) or not. Second, we conducted a 4 month experiment and compared the Microsoft Outlook and iems system to see which one is easier to manage in terms of users email account. The last was a 4 month experiment conducted to see whether reconstructing the rules was a difficult task or not should disaster occur.

Experiment 1

We conducted this experiment in 20 minutes, setting up 10 computers with fully installed iems system, at a university computer laboratory. We arranged two groups called Group A, & Group B. Each group had 5 members, all students from university. We asked both groups to create a rule and compared how much time they took to customize a rule.

Group A had some experience in customize rules for email classification and group the emails into folders. Group B has no idea about rules, but they knew how to group the emails into folder.

From our observation, we found that most Group B's member only run the system directly without thinking, and they found that the system help them to predict a folder for each email. They could identify error of prediction from some emails, and the system enabled them to see the result of how system has been done so far, especially in highlighting some of most important keywords in green, based on the machine learning result and what the user input manually. Some users tried to add/delete some of keywords and clicked on the confirm button. They tried to test the system in order to see the different result produced.

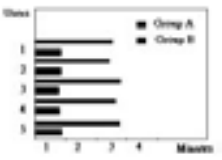

Figure 4.Group A & Group B

From the results, we believe that most Group B's members were able to create a rule in an average of 3 minutes. See Figure 4. Group A had good results as well, they could customize a rule on average in under 1 minute. It is very clear that both groups can perform and feel confident in customizing a rule for email classification in very a short time.

After this test, we gave them a questionnaires. The feedback was very positive. We found the Group A members were very happy with the SPBD technology as they felt it was easy to turn the rules, and believe they could customize a new rule in very a shout time. Group B members enjoyed creating a rule, because they could use their natural ability to identify error prediction easily. Meanwhile, they could change the rules as naturally without any stress.

Experiment 2

We set up another experiment, and conducted it over 4 months. First we looked at 10 users from university lectures. Currently they use Microsoft outlook filter to classify email into folder. The average amount of emails received in each day was 69 messages. We force them to use iems system to classify the emails, and then we compared the results to see which one gave a better performance.

After 3 months, we found very positive result such as the members continuing to use iems system to receive the emails from students and other staff. The reason given was that they could create/tune the rules in a very short space of time and only the system only required simple actions, compared with Microsoft Outlook.

Experiment 3

This experiment was conducted over 4 months. First we asked 10 users from university lectures to join our experiment. These students had been attacked by computer disaster, such as viruses, hard drive fault and operating system failure or other effects. They also had experience using Microsoft Express filters and know how to use rules to file emails into folders automatically. We asked them to use our new iems system instead of their current system.

After 4 months, we found very positive answers. They found iems system very powerful in creating rules in a very short time. They didn't feel that the system required much effort to use and they believed that if a disaster should happen again, they would not worry about re-creating the rules. One thing they worried about was how to back up the email files and it would seem that this is becoming another important task.

5. Conclusion

The SPBD approach is an easy-to-use personal assistant that helps users create a rule in a way more natural to them to solve their email classification problems. SPBD approach makes very few demands on users; they have nothing extra to learn when creating the rules and the only thing required is their natural ability. Users can identify the wrong prediction easily and move to the mail to the correct folder. Meanwhile it pops up with an interface to provide details and to ask them to modify it if

necessary. In the future, if the system detects a similar email coming in, it will predict a folder for this email. Users are required to do a final confirmation such as clicking on Achieve button.

The experiment results are very positive, as previous discussed. Experience and Non-Experience users can do the same task after 3 minutes. Some who have experience using Microsoft Outlook filters to classify their email into folder changed email reader from Microsoft Outlook to IEMS after 4 months. From the other experiment result, we see some are worry about. After 4 months, they believe this is the right technology to assist them in recovering rules in a very short time. This amounts to a significant qualitative enhancement that is likely to encourage users to file their mail using email filter by naturally ability.

While IEMS was developed for electronic mail the same technique can easily be used to organize other types of electronic documents such as disk files, audio, bookmarks, recordings, and other text-based documents that are placed into a hierarchy of folders.

ACKNOWLEDGMENTS

We would like to thank the people who contributed their email archives enabling us to conduct the expirical study. We also thank the reviewers for their valuable recommendations for improving this document.

References and Citations

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K., & Spyropoulos, C. An experimental comparison of naïve Bayesian and keyword-based anit-spam filtering with personal e-mail messages. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval IN, 2000.

[2] Boone, G. Concept features in re:agent, an intelligent email agent. Second International Conference on Autonomous Agents, 1998.

[3] Cohen, W. Learning rules that classify e-mail. Papers from the AAAI Spring Symposium on Machine Learning in Information Access, pp. 18-25, 1996.

[4] Crawford, E., Kay, J., & McCreath, E. Automatic induction of rules for e-mail classification. In Proceeding of the Sixth Australiansian Document Computing Symposium, coffs Harbour, Australia, 2001.

[5] Cypher, A. Watch What I Do: Programming by Demonstration. MIT Press, Cambridge, Mass., 1993

[6] Ducheneaut, N., & Bellotti, V., Email as habitat: an exploration of embedded personal information management. Interactions, 8, 30-38, 2001.

[7] Lau, T., Weld, D. S., Programming by demonstration: An inductive learning Formulation, Intelligent User Interface, 145-152, 1999.

[8] Mackay, W., Triggers and barriers to customizing software. CHI'91 Conference on Human Factors in Computing Systems (pp. 153-160). New in Computing System. New Orleans, Louisiana. 1991.

[9] Myers, B. Creating user interfaces using programming by example, visual programming, and constraints. ACM Transact. Program Lang, Syst. 12, 2, 143-177, 1990.

[10] Pantel, P., & Lin, D., Spamcop: A spam classification & organization program. Proceedings of AAAI-98 Workshop on Learning for Text Categorization, (pp. 95-98), 1998.

[11] Pazzani, M. Representation of electronic mail filtering profiles: A user study. Proc. ACM Conf. Intelligent User Interfaces. ACM Press, 2000.

[12] Provost, J. Naïve-bayes vs. rule-learning in classification of email, 1999.

[13] Rennie, J. Ifile: An application of machine learning to e-mail filtering. KDD-2000 Text Mining Workshop, Boston, 2000.

[14] Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. A Bayesian approach to filtering junk email, AAAI-98 Workshop on learning for Text Categorization IN, 1998.

[15] Segal, R., & Kephart, M. Mailcat: An intelligent assistant for organizing e-mail. Proceedings of the Third International Conference on Autonomous Agents (pp. 276-282). Seattle, WA, 1999.

[16] Smith, D., Cypher, A. and Tesler, L., Novice programming comes of age, Communication of the ACM, 43(3):75-81, 2000.