A MACHINE-LEARNING APPRENTICE FOR THE COMPLETION OF REPETITIVE FORMS

Leonard A. Hermens
lhermens@eecs.wsu.edu

Jeffrey C. Schlimmer
schlimme@eecs.wsu.edu

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164-2752

ABSTRACT

Forms of all types are used in businesses and government agencies, and most of them are filled in by hand. Yet much time and effort has been expended to automate form-filling by programming specific systems on computers. The high cost of programmers and other resources prohibits many organizations from benefitting from efficient office automation. A learning apprentice can be used for such repetitious form-filling tasks. In this paper, we establish the need for learning apprentices, describe a framework for such a system, explain the difficulties of form-filling, and present empirical results of a form-filling system used in our department from September 1991 to April 1992. The form-filling apprentice saves up to 87% in keystroke effort and correctly predicts nearly 90% of the values on the form.

INTRODUCTION

Forms are a pervasive part of the operation of modern government and business. As operations become more complex, the forms become increasingly complex too, making it difficult for personnel to complete forms accurately and efficiently. Errors committed by personnel during form-filling can be attributed to general misunderstandings about a particular form or the system in which a form is used. Through the use of machine-learning tools it is possible to assist personnel with repetitious form-filling tasks by providing useful default values for sections of a form, thereby reducing the number of keystrokes necessary to complete a form and reducing the risk of errors. One attractive scenario for automated form processing begins with an office worker who is knowledgeable about a particular task and needs to add information to a form.

Using a personal computer or workstation, a paper form is scanned and transformed into an electronic version. The form appears on a computer screen, with each field on the paper form having a corresponding editable field on-screen. Information may be added to the form while a prediction system assists the user by suggesting default values for blank fields and offers friendly advice about possible inconsistencies in the way the form is filled out (form validation). When the worker has finished with the form, it is sent electronically to others. Again, the computer may offer suggestions to help the user route the form to the appropriate people and track its progress enroute. If desired, the finished form may be printed on a suitable printer. This scenario is within reach of current technology. Scanners of sufficient resolution, computers of sufficient memory and speed, and networking components to link personal computers and workstations are all currently available. Software to enable this scenario, on the other hand, requires three significant components: input, output, and intermediate processing stages. On the input side, researchers are making progress on the problem of assimilating scanned documents [1] and have made considerable progress with the tasks of recognizing the form, segmenting the image into fields, and capturing each field's contents. On the output side, NASA researchers have begun looking at the problems associated with automatically routing forms to the next appropriate worker and validating form content [2]. We focus here on the intermediate processing stage, when the form is actually filled in.

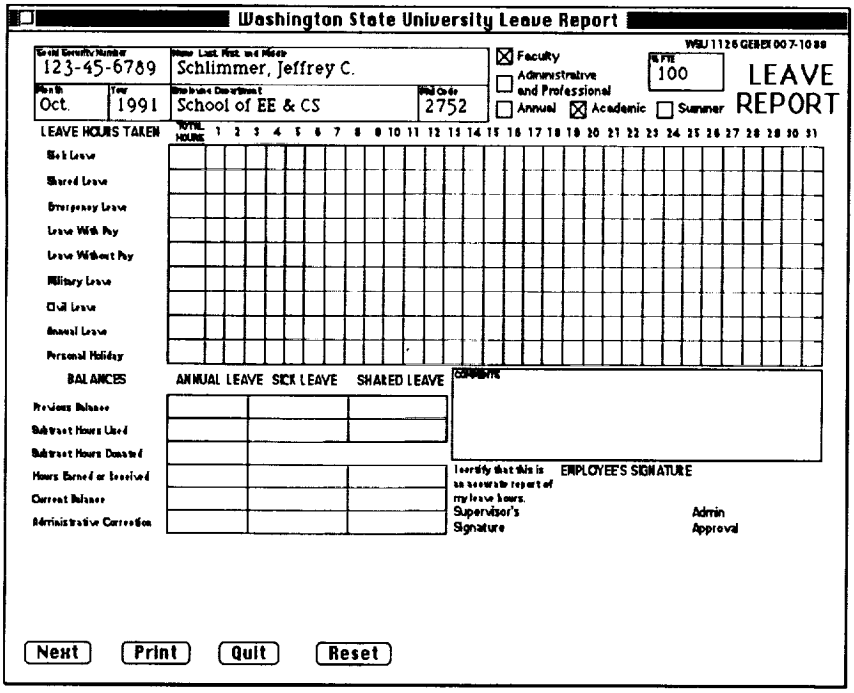

FIGURE 1. The Leave Report form is displayed on-screen in its own window. The user may click the mouse cursor from box-to-box and edit the contents of fields. The control buttons labeled Next, Print, Save, Quit, and Reset are part of the user interface to the form-filling and learning system.

Although a form-filling system can be explicitly programmed for each individual form, there is considerable software engineering overhead for the eventual convenience. Programmers must understand the semantics of the forms in detail, be able to encode specific information into the form-filling program, and then maintain the program as the form itself changes over time. Individuals, companies and government agencies may not have sufficient programmer resources to create and maintain form-filling programs for the hundreds of forms they require to conduct their business. In sharp contrast, programming is not needed with a learning form-filling system because it is able to provide reasonably accurate advice without being explicitly coded to do so. This is one of the hallmarks of such a system.

FORM-FILLING SYSTEM

The focus of this paper is on the intermediate processing stage of form-filling, so we assume the existence of an electronically reproduced, on-screen form—an *electronic form*. The electronic form is created to visually resemble its paper counterpart for two reasons: so users can easily accept the new technology, and so the daily work flow of the user is not adversely affected by the system. Figure 1 shows an electronic version of a Leave Report form used at Washington State University. Although the electronic form appears to be optically scanned image of a paper form, it is actually quite different. Each box, or *field*, on the form is editable; which means a user can type into a text box (e.g., name, address, or social security number), or select a check-box (for selection items) by using the computer's mouse and keyboard. The example form shown in Figure 1 consists of over 300 fields for information input, using both editable text boxes and check-boxes. The user has random access to any of the displayed fields. The control buttons at the bottom of the form labeled Next, Print, Quit, and Reset are part of the user interface to the form-filling and learning system and are not part of a printed Leave Report form.

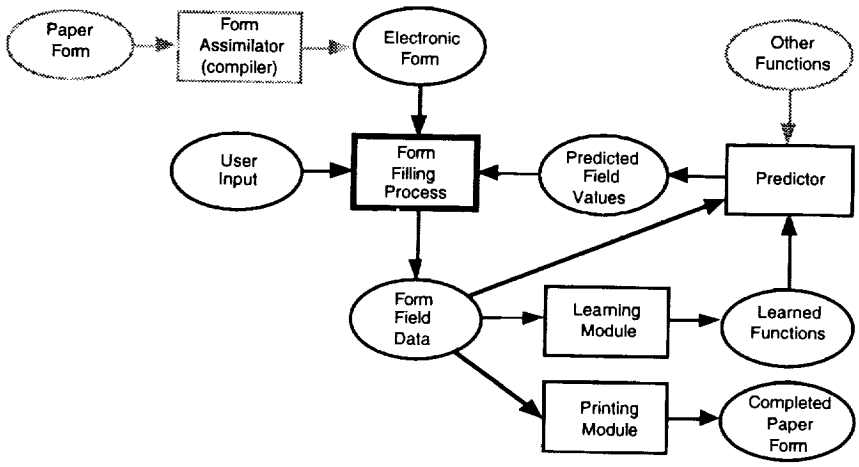

FIGURE 2. A high-level architecture of the form-filling system. Ovals indicate data manipulated by the system, and the boxes are system processes. Arrows indicate the data flow. Although current technology suggests that the automated transformation of a paper form into an electronic one is reasonable and feasible, the shaded oval and box labeled "paper form" and "Form Assimilator (compiler)" were not embodied in the system. The oval titled "Other Functions" indicates expansion possibilities not explored here.

Figure 2 shows a block diagram of the form-filling system used in conjunction with the control buttons. The thick-lined box in the center of the diagram is the core form filling process that combines the electronic form, the user input, and prediction feedback. When a user completes a form field (by typing into an editable box or by clicking a check box), that information is passed as form field data to three modules. Data is presented to the printing module so the user may generate a paper copy of the electronic form. The learning module uses the form data to construct predictive functions; these are used in turn by the predictor module to provide default values for other fields on the form. After each form field is edited by the user, a default value is predicted for each field; after each form is completed, the predictive functions are updated by the learning module.

Although some of the form-filling functions shown in Figure 2 are commonly available in commercial form design packages, our system has an additional component — the learning module. But before we can describe how the learning module plays a role in our system, it is important to first understand how the user interacts with the electronic form.

When the electronic form system is started for the first time, the form fields are blank. To access a field, the user moves the mouse input device to position the screen cursor over a text box or check box. Clicking on a check box will toggle an unchecked box to checked and vice versa. A click on a text box will illuminate a text-edit cursor which indicates that the user may type information into the field. If a field must be changed, the user can employ typical editing commands to delete or change a field's contents. When user has completed the form, they may click on the Print control button to print a paper copy of the form, click Quit to end the session, or click Next begin working on a new instantiation of the same form. Once a new instantiation of the form is shown on the screen, all of the fields are blank again. When a form is being shown on the screen, it is the *current form.* Once the Next button has been clicked, a new instantiation is displayed, and the form that was most recently displayed becomes the *previous form.*

With this model for the completion of electronic forms, we can now discuss the learning component of the system. Suppose a user begins working on form and types values into several fields. The learning module may incorporate any or all of the information for possible future use by the prediction

module. It may use values from fields on the previous form or structure learned from those examples to complete the remaining fields. For example, the system may use the social security number field on the previous form as a predictor for the social security number field on the current form, if it is applicable to do so. If there are no predictions for a field, the field is left blank. If a prediction is made, all applicable fields are updated on the screen. The system will not change any fields that the user has filled in because they are presumed to be confirmed by the user. The form-filling system and the associated prediction methods are very proactive, yet not intrusive. The user does not have to specifically request default predictions because they are always displayed, yet the system is not intrusive in that any default value presented to the user can be easily overridden with normal editing commands.

Using this user interaction model and the architecture shown in Figure 2, a functioning form-filling apprentice program was designed, implemented and used to process 269 Washington State University (WSU) Leave Report forms from September 1991 to April 1992. Although viewed as a prototype, the operational system was fully-functional with respect to form-filling, learning, prediction, and printing. It ran on a Macintosh computer, and was in actual use by three different office support personnel. The system we implemented was tested using each of two learning methods: COBWEB [3] and ID4 [4], an incremental version of ID3 [5].

To increase performance and improve early learning in the system, the learning and prediction subsystems are allowed to use values from fields on the previous form to predict fields on the current form. This means that the system can effectively learn sequences when the user is filling out form in repetition, either sequential or cyclical. For example, for the form shown in Figure 1, after a month of examples (one processing cycle), the system was able to correctly predict the sequence of employees in our department, and the system filled in the appropriate fields on the next copy of the form.

Using COBWEB, field data values from previous forms are used to predict all values of the fields on the current form. In those fields for which there is no prediction, the field is left blank. The prediction cycle is initiated whenever the user clicks the Next button to instantiate a new form. ID4 was slightly different in that it used a bias called *field ranking*. A typical form is designed to be completed from left to right and top to bottom. So each field on the electronic form is assigned an internal numeric rank, increasing first from left to right and then from top to bottom. The prediction mechanism associated with ID4 is prohibited from referencing any field that has a rank higher than the one being predicted on the current form, but is allowed to use example values from the previous forms. There is a learned function for each field and predictions are made independently. The prediction mechanism fires on fields only if the user has not changed the contents of the field, and only if the field's rank is greater than the field being edited. This *form bias* has proven effective, and system responses have been consistent with users' expectations.

EVALUATION

Our system was tested with COBWEB, ID4, and three reference (benchmark) methods including: no-learning (NL), most-commonly-used values (MC), and most-recently-used values (MR). The NL method provided no default values for fields in each new instantiation of the form. This was equivalent to having the user fill out an entire form manually without the aid of a machine learning system. We defined this as the worst-case behavior so that it could be used for comparison with other methods. The MR method predicted the most recent value for a given field, and the MC method predicted the most common. (In case of a tie, the most recent value was predicted.) Form data was collected and saved for each processed form so that a variety of experimental learning methods could be run subsequently to evaluate and compare their performance.

To evaluate the effectiveness of the form-filling apprentice, we used two comparable metrics for the Leave Report form. The first measure was the total number of keystrokes, and the second was the total

number of field prediction errors. A *keystroke error* was recorded for each key that the user typed to override a prediction made by the system or to insert values into an otherwise blank field on the form. *Prediction error*s are measured by counting the number of fields that the user changed, either to delete or insert information. Typing errors introduced by the user were not counted in either of these totals. Results indicate that ID4 reduced the number of required keystrokes by 87% on 269 forms processed, as compared to the no- learning (NL) method. In addition, the prediction-error rate for ID4 was one-tenth that of NL. However, ID4 was dependent on the ordering of the fields on a particular form and was reliant on the order in which the forms were filled out. Performance for COBWEB was not quite as good as ID4 on this task, but it still reduced the number of keystrokes by approximately 64%. When form processing is very cyclical or sequential as in the Leave Report form-filling task, the system was very good at predicting most of the fields for each new form in the sequence using either COBWEB or ID4. We characterize ID4 as accurate, but inflexible, and COBWEB as somewhat flexible with reasonable accuracy.

Figure 3 is a composite graph that shows the percentage of keystrokes saved over no-learning for each learning method. The horizontal, independent axis represents the form processing order from 1 to 269 (labeled chronologically from September to April), and includes two indicators for when new employees were added to the processing cycle. The vertical, dependent axis for each learning method ranges from 0 to 100%. Zero percent corresponds to no correct predictions, 100% indicates all correct predictions, and 50% corresponds to correctly predicting half of the keystrokes. Tick marks to the left of each strip chart show the minimum and maximum keystroke savings in percent, along with the median percentage for the first month and last month of processing. The histogram on the right hand side of each graph shows the percentile density for each method.

Periodic downward spikes in the graph indicate poorer performance in the respective learning methods. This is correlated with either the start of a new month in the cycle or the addition of a new employee to the processing order. All of the methods tested have this characteristic to some degree. ID4 has comparable performance to the other methods during the first month, but then improves dramatically and plateaus for the remaining months. The relatively small number of prediction errors after the month of September can be attributed to two factors: the difficulty of predicting a field value for Previous-Balance-Sick-Leave, and the addition of two new employees to the system in January and March.

Predicting Previous-Balance-Sick-Leave is difficult because the system needs to sum a field value on the current form and a field value on the employee's form from the prior processed month. The dependency between a prior month's form and a current form is very much like connected spreadsheets; a field value in one spreadsheet affects an update on a field on separate but connected spreadsheet. Improved results might be realized when an effective method for learning these spreadsheet-like calculations is developed.

DIFFICULTIES OF FORM-FILLING

As might be expected, some fields are quite easy to predict accurately while others are considerably more difficult. For example, the form in Figure 1 shows five check boxes labeled Faculty, Annual, Administrative/Professional, Academic, and Summer. Although the office support personnel believed that they understood the meaning of these boxes, the semantics were often confusing and resulted in user errors. The machine learning methods we used also had difficulty some of them. For example, Equations 1 and 2 show the desired rules for two of the check boxes on the Leave Report form:

$$\text{Check ACADEMIC when NAME is a faculty member and Month} \in \{Aug, Sep, Oct, Nov, Dec, Jan, Feb, Mar, Apr, May\} \quad \text{(EQ 1)}$$

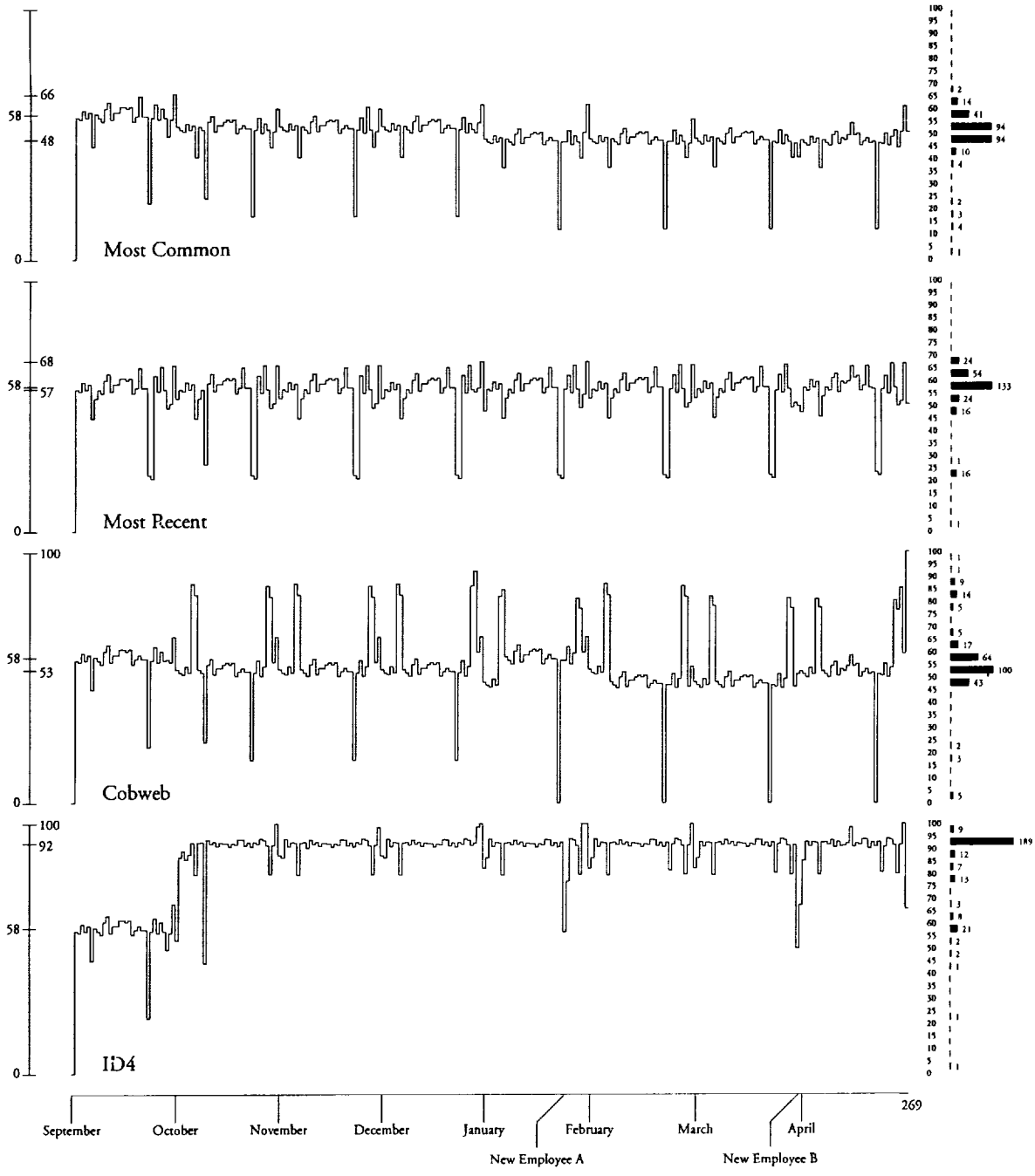

FIGURE 3. A composite graph shows the percentage of keystrokes saved over NL (no-learning) for each learning method tested on the Leave Report form. The horizontal axis is labeled chronologically with indicators for when two new employees were added to the processing cycle. The vertical axis for each learning method indicates the minimum and maximum keystroke savings in percent, as well as the median percentage for the first four months and last four months of processing. A histogram on the right hand side of each graph shows the percentile density for each method.

Check SUMMER when NAME is a faculty member, has summer support, and
Month $\in$ {*May, June, July, August*} (EQ 2)

To be accurate, the system must properly relate a set of months and check boxes to predict the boxes labeled Summer and Academic. One should note that both boxes may be checked in the months of May and August because each of these months are half-summer and half-academic. This subtlety can easily be overlooked by the user, so it is very desirable for the system to accurately predict these fields. Other roadblocks to learning the form shown in Figure 1, for example, include the complex formula for calculating the earned sick leave, which is based on the two above check boxes, plus the %FTE box and a constant value of 8.0. Desired rules for this box are indicated in Equations 3–6:

If Month $\in$ {*Sep, Oct, Nov, Dec, Jan, Feb, Mar, Apr*} and ACADEMIC is checked
then SICK-LEAVE-HOURS-EARNED-OR-RECEIVED is 8.0. (EQ 3)

If Month $\in$ {*May, Aug*} and ACADEMIC is checked, and SUMMER is not checked,
then SICK-LEAVE-HOURS-EARNED-OR-RECEIVED is 4.0. (EQ 4)

If Month $\in$ {*May, Aug*} and ACADEMIC is checked, and SUMMER is checked,
then SICK-LEAVE-HOURS-EARNED-OR-RECEIVED is (%FTE $\cdot$ 8.0). (EQ 5)

If Month $\in$ {*June, July*} and SUMMER is checked,
then SICK-LEAVE-HOURS-EARNED-OR-RECEIVED is (%FTE $\cdot$ 8.0). (EQ 6)

There are other rules that can be generated from the two check boxes and the %FTE cells; however, these rules listed above are the only semantically valid conditions for this form. One could imagine that a simple spreadsheet program can handle conditional formulas such as these, but it would require explicit programming by the user. Our desire is to avoid programming systems for complex rules like those shown above, so the goal remains to provide an agent capable of learning rules like these.

RELATED WORK

Our system is somewhat similar to other apprenticeship systems like CAP [6], which was developed to help maintain an appointment calendar. CAP was designed to advise an appointment calendar user in the same way that a knowledgable secretary might. For example, a certain type of meeting may require a certain room at a particular time of day—information that a secretary would know from experience. CAP uses learning from examples to predict three features of newly scheduled appointments: meeting time, duration of meeting, and meeting location. The system has been used to manage a faculty member's appointment calendar.

CAP's user interface is based on the Emacs editor, and the prediction information and queries are presented sequentially to the user. Questions asked of the user are presented using a command-line type dialog, and default prediction values are displayed one-at-a-time. In contrast, our system allows the user to view all of the pertinent information on the active form, on-screen, all at once. This gives the user the advantage of global random access to the form fields and their contents. The user is always in control of the order in which the fields are completed.

CAP is designed to utilize a knowledge base that contains calendar information, a database of personnel information, and other system information like currently active rules, neural network computation data, and a history of user input and commands. Alternatively, our system does not utilize information databases (except for the history of completed form examples), yet it attains reasonable predictions in a relatively short amount of time. A departmental database would aid in the prediction of some fields on the Leave Report form, but empirical results have shown that these fields can be predicted quite well after the first month of training.

CONCLUSION AND FUTURE WORK

We have shown that an apprentice can reduce user effort on repetitive form-filling tasks, and we have described a framework in which these tasks can be accomplished. Our form-filling system yielded reasonable predictions for the fields on our test form. Although the results are promising, the system should be tested over a variety of different forms and typical users. This may reveal broad issues that may not have been uncovered in the confines of a single example form.

Although many issues still abound, we foresee at least two new avenues of research in the future. The first issue is that most paper forms are designed for ease of use within the current paper-oriented workplace—paper forms are not designed for electronic processing. Multiple fragments of information may have been clustered into a single field; a simple example of this is a telephone number field. In the U.S., an area code is a predictor for state of residence. The fact that most forms only have one field for telephone number limits the predictive capacity of the area code fragment. It is important, therefore, to find a mechanism by which the syntax of a particular form can be learned so that over-generalized fields can be appropriately partitioned and so that viable predictions can be made.

A second interesting direction for this research is the idea of allowing users to design and create their own forms. In an interactive drawing system, the user may create editable text boxes by drawing their shape with an input device. Then they may complete their personalized form with the aid of an apprentice system. The system may also suggest alternative representations of the user's form, when appropriate, by adding check boxes or converting fields to selection lists to create a more convenient and useful form. These seemingly independent paths for future research might be easily integrated and provide for additional challenges in learning form-filling.

ACKNOWLEDGMENTS

We would like to thank Susan Wallen, Holly Snyder, and Marie Roberts for making themselves available to use and evaluate our system. We also thank them for their cooperation and patience. We would like to thank the anonymous reviewers who provided several helpful suggestions after reading an earlier draft of this paper. This work was supported in part by the National Science Foundation under grant number 92-1290 and by Digital Equipment Corporation. Computing support was provided by Apple Macintosh Common Lisp, and DeltaPoint's DeltaGraph.

REFERENCES

1. Fisher, J., Hinds, S. and D'Amato, D., "A Rule-based System for Document Image Segmentation," *Proceedings of the Tenth International Conference on Pattern Recognition*, IEEE Press, Atlantic City, NJ, 1990, pp. 567–572.
2. Compton, M., Stewart, H., Tabibzadeh, S., and Hastings, B., "Intelligent Purchase Request System," In *Tech. Rep. No. FIA-92-07*, NASA Ames Research Center, Moffett Field, CA, 1992, pp. 56–57.
3. Fisher, D. H., "Knowledge Acquisition via Incremental Conceptual Clustering," *Machine Learning*, Vol. 2, No. 2, 1987, pp. 139–172.
4. Schlimmer, J.C. and Fisher, D., "A Case Study of Incremental Concept Induction," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, AAAI Press, pp. 496–501.
5. Quinlan, J.R., "Induction of Decision Trees," *Machine Learning*, Vol. 1, No. 1, 1986, pp. 81–106.
6. Dent, L., Boticario, J, McDermott, J., Mitchell, T. and Zabowski, D., "A Personal Learning Apprentice," *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI Press, San Jose, CA, 1992, pp. 96–103.