Using Heterogeneous Learning Techniques For Identification in a Real World Problem Domain

by

Antony Van der Mude

Coopers & Lybrand
1251 Avenue of the Americas
New York NY 10020

ABSTRACT

A program was developed that converts arbitrary files into data tables. To do this, it determines file type and data field formats, then prompts the user to indicate which source field should be matched to a target field. Analysis of the input file uses a collection of heterogeneous pattern recognition and machine learning techniques. These techniques are tied together by an expert system that keeps track of qualitative descriptions of the file features. As the file is characterized in more detail, specialized feature extraction functions are called to yield more information.

AI Topic: Feature Extraction, Pattern Recognition, Machine Learning

Domain Area: Financial data analysis, Computer productivity tools

Language: Written in C for use on a standard IBM PC XT or compatible with 640K memory and a hard disk.

Status: The program has been completed and the user test is in process.

Effort: 1.5 Man-years

Impact: This system will increase the use of proprietary software on most of the firm's audit engagements.

Using Heterogeneous Learning Techniques For Identification in a Real World Problem Domain

Antony Van der Mude

Coopers & Lybrand
1251 Avenue of the Americas
New York NY 10020

ABSTRACT

A program was developed that converts arbitrary files into data tables. To do this, it determines file type and data field formats, then prompts the user to indicate which source field should be matched to a target field. Analysis of the input file uses a collection of heterogeneous pattern recognition and machine learning techniques. These techniques are tied together by an expert system that keeps track of qualitative descriptions of the file features. As the file is characterized in more detail, specialized feature extraction functions are called to yield more information.

INTRODUCTION

A program was designed to convert print images of account information from a general ledger accounting package into a data table readable by proprietary software used in the auditing process. It is possible to import the data as a Lotus 123 file or as an ASCII table in a predefined, fixed format. Unfortunately, the data format sometimes makes it inconvenient to use those methods. It is often necessary to obtain a printout, then retype the data into the computer. There have also been cases where people have used scanners to read printouts, correct the scanned-in data and then reformat.

A learning system was chosen because it would, in most reasonable cases, handle much of the mundane conversion work. This takes the burden off the user, who can then identify which field is the account number, account description, current or previous balances or extra information (such as financial line number) which should be transferred to the target file.

Pattern Recognition and Machine Learning have similar objectives: given a set of data, find a hypothesis from a space of hypotheses that describes the data acceptably. Problems arise if a technique does not work for all cases of interest, or when each technique analyzes only a particular facet of the problem, not the whole description. Also in some cases, only a partial determination may be desired, such as the gross characteristics of the unknown, with the finer detail to be determined later. Our approach addresses these problems.

The idea of using coordinating techniques has been applied to recognition problems before. This is the intent of the blackboard methodology in HEARSAY [7]. Current research in machine learning has been concerned with comparisons of different techniques [9]. Although this analysis determines which techniques are suitable on specific problems, it leaves unanswered the question of how to combine them in a coordinated manner. A recent implementation of heterogeneous machine learning methods [8] led to a decrease in learning efficiency since the methods were run in parallel. Instead of an a priori

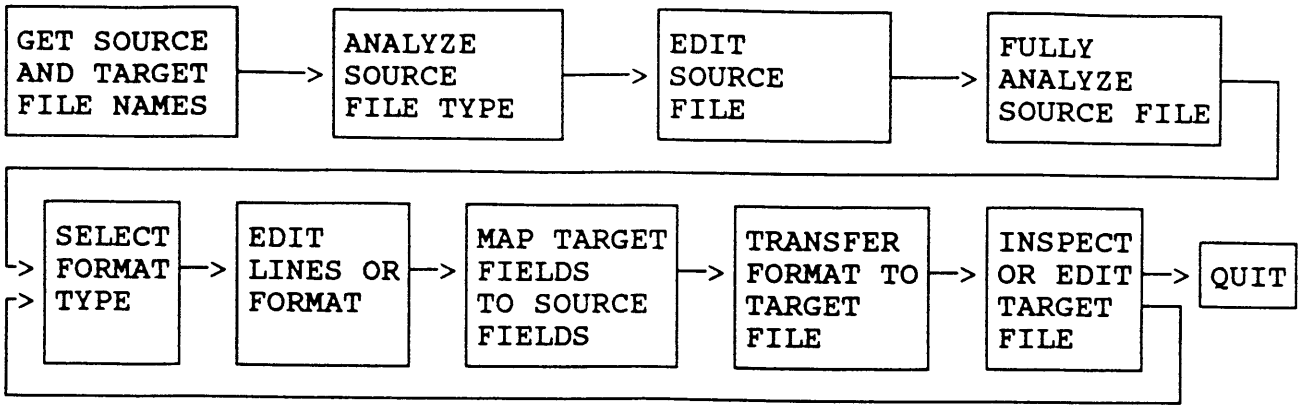

Figure 1. Architecture of the data conversion program.

determination of what learning method to apply, it is better to provide some sort of partial analysis to guide the choice of methods and enable backtracking if one fails.

OVERVIEW

The overall architecture of the program is given in Figure 1. It is composed of a data file analysis module and a data translator connected and controlled by a user interface designed to display large files with many different formats. The analyzer does a syntactic analysis of the file. The translator converts the source into the target fields based on the user's mapping source fields to target fields.

This paper is primarily concerned with the data analyzer. It is a rudimentary forward chaining production rule system controlling the execution of the different learning techniques such as statistical pattern recognition, a pattern language learning system, and classifiers for special file features.

The control store for the production system is a number of character strings. A rule is fired when the strings in the "if" part match the control store. The "then" part of the rule calls a data analysis function whose final result is to alter the string variables. A clean separation between the production system control and detailed data analysis results in a production system that is a throwback to the original pure production rule concept [3]. Since efficiency is secondary to integrating many different types of analysis code, the production system is written as a simple if-else statement in C.

DETAILED DESCRIPTION OF THE DATA ANALYZER

The input file is presumed to be a listing of a data table, composed of characters, organized into lines, grouped into pages. Each line may have one of a number of different formats. There are strings identifying features such as character-type, page-type, line-delimiters, or format-type.

Each analysis function works on a single 32K sample from the start of the data file. The data analysis can result in the generation of quantitative information stored in a large grab-bag of data values accessible to other functions, but not to the production system. The rest of this section describes some of the rules currently implemented, with a summary of the learning or pattern recognition technique controlled by that rule.

IF Char-type Undefined
THEN Char-type = typechar()
The typechar function counts characters in the sample into "types" such as letters, digits, control characters, punctuation, etc. The function computes the Euclidean distance to the characteristic vector for text files ("STANDARD") versus data files ("BINARY"). As more experience is gained this function will be improved by replacing the Euclidean distance by a Bayesean measure [4]. A measure of the goodness of fit to the data will also be done, so that "none of the above" can be returned if a sample is placed between two choices and not close enough to either.

IF Char-type Standard
AND Line-type & delimiter Undefined
THEN Line-delimiter = typeline()
The typeline function determines the line separator and checks whether it is correct for an IBM PC. The existence of the carriage return/line feed pair indicates the Line-type is "STANDARD".

IF Char-data AND Line-data defined
AND lineflag set THEN ready to scan
A partial analysis is a check of the character type and line delimiters but not the format of the lines. This is done by initializing the analyzer with the variable lineflag set.

IF Line-type Standard
AND Page-type Defined
AND Format-type Undefined
THEN Format-type = delmfrmt()
Two types of delimited files are considered. The first is a simple character delimited file as:
1110,Accts Recv-Trade,A,2,7890.12
The other type uses a string delimiter:
1110,"Accts Recv, Trade","A",2,7890.12
This type of recognition is ad hoc. Two problems that can arise are the erroneous identification of decimal points as field delimiters, and, permitting the inclusion of the string delimiter in the string itself. The first problem is handled in an arbitrary manner - decimal points are forbidden as field delimiters. The second case is not addressed currently.

IF Line-type Standard
AND Page-type Defined
AND Format-type Irregular
THEN Format-type = typefrmt()
The typefrmt function is the most sophisticated function in the program. This program uses a hill-climbing technique [2] to determine the best formats. A format is initially considered to be a string of character classifications, which get grouped into fields. This type of language is a simple case of a pattern language, such as the kind inferred by LEX [6]. Languages of this form have a low order polynomial inference complexity [1].

The function operates as follows. Given the list of lines in the sample that are currently unclassified as to format, classify each character on each line into one of these basic types: SPC: Spaces, DIG: Digits, LET: Alphabetic Letters, CTL: Common Control Characters, PNC: Digit Punctuation (.,+-), -CTL: Other Control Characters, -PNC: Other (Non-digit) Punctuation Characters, NON: Non-ASCII characters. Next, the function determines the most common character type for each column, so an initial format string can be derived. If there is one format that predominates over all others, it will be closest to this initial format. If not, one must isolate some lines with a similar format. This is accomplished by scoring each unclassified line by the number of misses with the initial format, graphing number of lines versus number of errors and finding the largest cohesive range of error values. Select only the lines which have errors inside this range then repeat the selection procedure until it stabilizes. This method is not guaranteed to succeed. One problem is that this simple scoring function will not resolve all formats, since two formats may have the same error score.

Once this sample has been selected it can be generalized. This is done by taking the minimum union of different categories if that will reduce the error rate in the selected lines to an acceptable minimum. This also allows excluded lines to be included again. The allowable generalization types are: [Digit/Space], [Alphabetic/Space], [Digit-Punct/Space], [Control], [Punctuation], [Number] (Digit/Digit-Punct/Space), [Alphanumeric], [Printing]. A second generalization method uses the set of generalization types in a context sensitive manner. For example, if there is a line that is the same as the current format except that the format has a sequence LET SPC SPC where the line has LET SPC LET, then generalize by the rule LET SPC SPC -> LET LET LET. That allows a preliminary format for lines with a variable length text or numeric field to be generated by concentrating on the typical length fields, then including lines with larger numbers or more text. The final step is to group DIG and DIG/SPC into Digit fields (Example: 12345), DIG/SPC/PNC in Number fields (Example: -12,345,678.90) and LET/DIG/SPC/PNC/-PNC into Text fields. Generalizations of these types have been used in grammatical inference [5].

IF Char-data & Line-data
& Page-data & Format-data defined
THEN ready to scan

Once the file characteristics are identified, the full file is read and processed. A format file is created, listing the format types and fields in each format. For each line in the file, the format that matches the line is given, along with the start byte in the file and the length of the line not including line and page separators.

DATA TRANSLATION AND THE USER INTERFACE

The user interface has been designed with three levels of interaction. Experts can use Pull-down menus for access to all functions: file read/write, editing, analysis or translation. Intermediate users use a flowchart-like screen structure using function keys, giving an overall structure to the process as shown in Figure 1. If this is too overwhelming, beginners can replace the keyboard input routine by an Advisor, which indicates the step the user is performing and presents a list of which operations might be done next. The selection is translated into the appropriate keystrokes.

Initially, the analyzer is run only to identify line breaks so the file can be displayed and the user can do a preliminary edit. After analysis and translation, it is possible to do a final edit either on the lines remaining in the source file or on the lines transferred to the target file.

A challenge presented by the user interface was how to perform the mapping from the source to the target fields, since some source fields could end up in more than one target field. An example of this occurs when the account number includes the financial line number as a subsequence. This problem was solved by having the target fields displayed horizontally, in a window above the file with a pull-down menu of source fields. As the user moves down the menu, the data in that field is highlighted.

The Data Translator looks up a table of source field type to target field type information and determines the conversion steps such as the pad character or conversion from alphabetic to numeric. The translator also knows the key field, for merging of data such as current and prior balances.

FUTURE DEVELOPMENT

The program runs on IBM PC's and compatibles and handles print images or comma delimited files in MS-DOS format. Other types of data files will terminate with the file unanalyzed. Therefore, the tree of methods currently comprising the system consists of two branches with stubs for future expansion.

Further development will be an eclectic approach of adding features in response to user's needs. This may include autocorrelation techniques to analyze line separation where no line separators exist and to determine page feeds and header and footer lines. If needed, we shall also look at adding binary data files, including the ability to interpret numbers in binary form. We also expect to include the ability for the user to add new output format types. Instead of entering the format by hand, it would be possible for the analyzer to be run on a test case.

ACKNOWLEDGMENTS

I would like to thank Saman Hong for his management and encouragement of this project. Ray Raud developed the user interface and data translator. Scott Butterfield programmed the data translator and helped in development of the user interface. Thanks also to David Frawley, Mary Van der Mude and Sean Wood for comments on the initial drafts of this paper.

REFERENCES

[1] Dana Angluin, "Finding Patterns Common to a Set of Strings," *Journal of Computer and System Sciences*, Volume 21, pp 46-62, 1980.

[2] Craig M. Cook, Azriel Rosenfeld and Alan R. Aronson, "Grammatical Inference by Hill Climbing," *Informational Sciences*, Volume 10, pp 59-80, 1976.

[3] Randall Davis and J. J. King, "An Overview of Production Systems," in E. Elcock and D. Michie (Eds.), *Machine Intelligence 8*, Chichester, England: Ellis Horwood, pp 300-332, 1977.

[4] Keinosuke Fukunaga, *Introduction to Statistical Pattern Recognition*, New York, NY: Academic Press, 1972.

[5] Bruce Knobe and Kathleen Knobe, "A Method for Inferring Context-Free Grammars," *Information and Control*, Volume 31, pp 129-146, 1976.

[6] Tom M. Mitchell, Paul E. Utgoff and Ranan Banerji, "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in R. S. Michalski, J. G. Carbonell and T. M. Mitchel (Eds.), *Machine Learning*, Los Altos, CA: Morgan Kaufman, pp 163-190, 1983.

[7] R. Reddy, L. Erman, R. Fennell and R. Neely, "The HEARSAY Speech Understanding System: An Example of the Speech Recognition Process," *IEEE Transactions on Computers*, Volume C-25, pp 427-431, 1976.

[8] David Tcheng, Bruce Lambert, Stephen C-Y Lu, and Larry Rendell, "Building Robust Learning Systems by Combining Induction and Optimization," *Proceedings of the Eleventh International Joint Conference of Artificial Intelligence*, pp 806-812, August 1989.

[9] Sholom M. Weiss and Ioannis Kapouleas, "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods," *Proceedings of the Eleventh International Joint Conference of Artificial Intelligence*, pp 781-787, August 1989.