Computing Educated Guesses

E. S. SPIEGELTHAL†

TO DISTINGUISH himself from the poor benighted man-in-the-street, the computer sophisticate is apt to refer to the beasts as "so-called giant brains" or as "lightning-fast idiots." He knows, as do we, the great gulf which separates the human brain from the general-purpose digital computer. Still, the exact dimensions of that gulf are quite unknown, and the desire to show that the hiatus between man and machine is smaller than many suspect impels both the adventuresome and the iconoclastic. The attempt, the successful attempt, to automate one area hitherto considered an exclusively human domain constitutes my topic today.

We are all familiar, if only by hearsay, with the troubles that can beset the best of computer programs if the input to the program is not thoroughly debugged. For a program designed to test the putative behavior of, say, a proposed steam turbine, where the input consists of a scant dozen or so parameters, input debugging is hardly a problem. The situation is quite different for a data-processing operation, particularly when the input is massive, as it usually is. Three alternatives, all unpleasant, present themselves to the supervisor of such a large-scale data-processing operation. He can build a wide variety of error-detecting features into his program, flagging all input errors for subsequent human correction, he can employ a host of human pre-editors to clean up the input, or he can hope that input errors are rare, and let it go at that.

Unhappily, there are many applications where errors are not rare, where the do-nothing solution is obviously frivolous and where, consequently, a sizeable group of humans is necessary, either as pre-editors or as on-line trouble-shooters. Nor is it always the case that the necessary human beings can be clerical types. Certain input debugging calls for sophisticated and knowledgeable practitioners. We are all hopeful—almost all, anyway—that keypunch machines and operators will sooner or later be superseded by character-reading devices and the like. There is no philosophical difficulty in conceiving of typed, printed, or handwritten characters being translated directly into computer language without any human intervention, provided, of course, that those characters were correctly typed, printed, or written to begin with. Suppose, however, that the source characters are incorrect. Consider the ingenuity expended in the Post Office just in recognizing all the variations of "Albuquerque." Our Russian colleagues are supposed to be far advanced in the domains of automatic translation and character-reading, but present their machines with a first edition of "Cybernetics," with all its typographical errors, and horrible difficulties would ensue. Our choice, then, is clear. Either we admit that many important data-processing applications are impossible to automate completely, or we find a way to mechanize the human capacity for making educated guesses. We believe that, for some applications at least, we have found a way.

While the techniques we have developed were conceived with one particular application in mind, I shall describe them without reference to that application, successful as it was. The principal reason for taking this tack is to be able to present the basic, quite general, features of our method without being tripped up by the special form-fitting required by the actual problem. So, let us be general, and consider any language with which humans attempt to communicate with one another. These may be natural languages, like English or German, or artificial languages like Esperanto or certain telegraphic codes. There are all sorts of personal reasons for communication being difficult—ignorance, dogmatism, poor sentence structure, etc.; however, even if these factors did not exist, all sorts of nonhuman noise would beset would-be communicators. Information theory makes much of "redundancy" as an aid in error-detecting and error-correcting when a noisy channel is being used. Indeed, even humans who have never heard of information theory make continual, and skillful, use of redundancy in unscrambling all sorts of garbled communications, whether the trouble be crosstalk in a telephone conversation or missing letters in a crossword puzzle. Without attempting to build a model of the brain, replete with neural nets and such, let us see if we can single out the functions performed by human redundancy-exploiters. If these functions turn out to be performable without recourse to extrasensory perception or to the psychokinetic effect, our automation problem is essentially solved. There remain only the minor problems of collecting all the necessary data, carrying out a rather gruesome programming task and finding a computer fast enough and capacious enough to make our solution practicable. I shall return later to this question of practicability. At the moment, allow me to sketch the functions which, when suitably programmed, allow a general-purpose computer to simulate a redundancy-exploiting, error-detecting, and error-correcting human being.

Rather than jump into a completely general and abstract formulation, let me use a concrete illustration. Fig. 1 shows two familiar sights, a correctly prepared mailing envelope and, below it, a somewhat sloppier version of the same thing. We shall assume at first that a

† General Electric Co., Bethesda, Md.

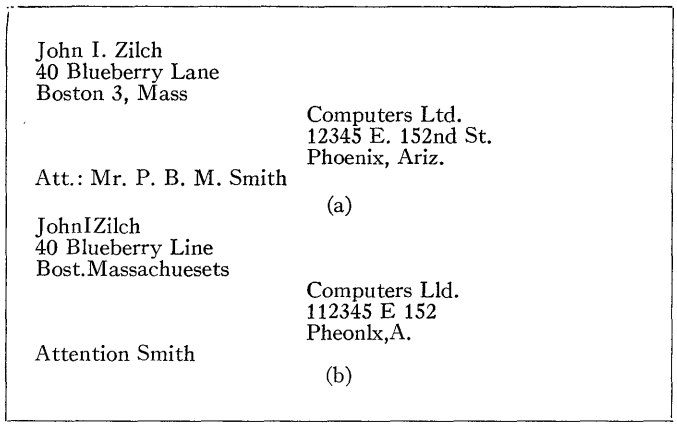

Fig. 1—Envelopes, (a) good, (b) bad.

"perfect" character-reading device has "read" the perfect envelope, and consider the functions which must be performed by a machine to "understand" the envelope, *i.e.*, to route it to the correct addressee by the desired means, *e.g.*, air mail or first class. We shall then consider a fallible character-reading device reading the lower, garbled, envelope, and see what can be done there. It should be emphasized that, in this application, we are not concerned with the essentially straightforward task of actually routing the envelope to its destination. Our job here is just to ascertain the information needed by the routing program.

Our machine must perform two separate functions on each "word" read from the envelope. A word here is any group of contiguous characters on one horizontal line, not containing any embedded blanks or commas. Given any such word, the machine must first ascertain the class of words represented by this word. In our example, the machine must determine that "Phoenix" is the addressee's city, and that "I." is an initial of the sender. This first function is called the "identification" of the word. The second function is that of "recognition." Having established the class to which a word belongs, it is next necessary to determine which one of the class members the given word represents. In our first example, the recognition process is simple, almost trivial. "Phoenix" is matched against every element in a master list of cities and, lo and behold, it is found that "Phoenix" is "Phoenix." A glance at the lower envelope on Fig. 1 will reassure you that the recognition problem is not always a trivial one.

The identification process is fairly easy for a Gestalt-perceiving, pattern-recognizing human who is himself accustomed to writing envelopes according to the standard format. The machine needs a little help in this direction. Fortunately, we can provide this help. On the one hand, we can program our machine to elicit the same data that our pattern-recognizing facility allows us to obtain. Clearly, our character-reader will be able to note, for each word, its relative position with respect to all other words on the envelope, and its position with respect to the envelope itself. For each word, then, we start off with the knowledge of the line it is on, its position on the line (left end, right end, interior) and the words which flank it on either side. With a little extra programming effort we can determine the length, *i.e.*, the number of characters of each word, its character pattern (is it all alphabetic, all numeric, some sort of hybrid?) and, perhaps, the presence in the word of some salient feature, *e.g.*, the colon following "ATT.:". Indeed, we can usually determine quite easily much more information than we need for the identification of our words. Much more, that is to say, when we are dealing with a noiseless channel, and/or a communication format as simple and relatively invariable as the front of an envelope.

Of course, whether this information is adequate, overly complete, or inadequate depends on how we use it. At this point in the identification process, the machine must turn to its accumulated store of factual knowledge, a store which is compiled by a subsidiary program in advance of production running. This store consists of lists and tables of probabilities, and provides the data which, in conjunction with the specific information for each envelope, allow each word to be identified with a high probability of correctness. Our basic technique here is the use of Bayes Factors as instruments for weighing evidence. Fig. 2 gives the essentials of this technique.

For each class of words that can occur in the specific type of communication in question—mail envelopes, in our example—an *a priori* probability is given for the occurrence of a representative (or two, or n) of that class. This probability, like all the others we use in this process, is derived from frequency counts on sufficiently large samples of the data to be processed. Also for each class, we provide the probabilities that, for example, a specific representative of that class will have length 3, or 4, or 5, or that the class representative will be found at the beginning, or the end, of a line. In brief, for every piece of information we scan each envelope for, we have a corresponding set of probability distributions, one set for each class of expected words.

In the identification phase of our program, we consider one actual word at a time, testing that word against the hypotheses that it is a representative of expected class A, B, etc. Eq. (1) in Fig. 2 gives the skeleton of such a test. Here we are testing the hypothesis that the word "Smith" is a representative of the "zone-number" class. Our frequency counting is supposed to have informed us that the *a priori* probability that any word on our envelope is in the zone-number class is 0.017. We first test our hypothesis by using the empirically-determined fact that "Smith" has length 5. This gives us our second term on the right side of (1), *i.e.*, the Bayes Factor for the "length event." The product of the Bayes Factor and the *a priori* probability is the *a posteriori* probability that "Smith" is a zone-number. Not very surprisingly, this is a small number. We now compare this number with two thresholds. If the *a*

$$P(H/E_1) = P(H)\cdot\frac{P(E_1/H)}{P(E_1)} = P(H)\cdot\frac{P(E_1/H)}{P(E_1/H)P(H) + P(E_1/\bar{H})P(\bar{H})} \tag{1}$$

$$= (0.017)(0.0001) = 0.0000017\begin{cases} > 0.000001 = \text{Rejection Threshold} \\ < 0.9 \quad\quad = \text{Acceptance Threshold} \end{cases}$$

$$P(H/E_1\cdot E_2) = P(H)\cdot\frac{P(E_1/H)P(E_2/H\cdot E_1)}{P(E_1\cdot E_2)} \sim P(H)\cdot\frac{P(E_1/H)}{P(E_1)}\cdot\frac{P(E_2/H)}{P(E_2)} \tag{2}$$

$$= (0.0000017)(0.00001) < 0.000001 = \text{Rejection Threshold}$$

where

H = hypothesis that "Smith" is a "zone-number"
E_1 = the event that the length of "Smith" is "5"
E_2 = the event that the pattern of "Smith" is "all alphabetic"

Fig. 2—Hypothesis testing.

posteriori probability exceeds the acceptance threshold, we accept the hypothesized identification and turn our attention to the next actual word; if the probability falls below the rejection threshold, we reject the hypothesis, and test the actual word against the next expected word class. Finally, if our probability falls between the two thresholds, we test the same hypothesis against the next event, using our *a posteriori* probability as the new *a priori* probability. In our example in Fig. 2 we have been generously low with our rejection threshold, so that it is necessary to go to (2) where we test the hypothesis, "Smith = zone-number," against the character pattern, and allow the low probability of a zone-number consisting exclusively of letters, to push our hypothesis into limbo. If we were scanning French addresses, with zone-numbers given in Roman numerals, the Bayes Factor in (2) would be very different.

After scrutinizing all the actual words on the envelope in this manner, we may find that certain words are still unidentified. In this case, we iterate through our process once again. However, certain features of the process will have changed. Suppose that we have identified two different zone-numbers in the first pass. Since we expect to find no further zone-numbers, we no longer test any of our undecided actual words against the hypothesis that they are zone-numbers. This not only reduces our processing time—it also changes the *a priori* probabilities of the remaining word classes, and affects the numbers entering into all the Bayes Factors. Another change in the second pass is that new evidence can be used to give rise to Bayes Factors. A word identified as a zone-number in the first pass provides strong evidence that the word to its left is a city name. Clearly, the topological relationships subsisting between words cannot be utilized until some words have been identified.

If successive identification passes still leave a residuum of unidentified actual words, as might happen if, for example, two or more words were run together, thus appearing to the machine as one word, there are subsidiary tricks that can be played. Due to time limitations, I shall have to leave these tricks to your imagination, and move on to the recognition phase.

In the simplest case, all actual words will have been correctly identified and, if the words are all correctly spelled and correctly ingested by our character-reader, recognition will consist of little more than finding the exact match in the proper list, a list determined by the identification of the word. It is possible to make even this simple process simpler or, at least, faster. To search a list of all the cities in the United States can be time-consuming, particularly if the list must be transferred from tape to core memory. However, if the corresponding state has previously been recognized, then a much reduced list of cities can be inputted and searched. Suppose further that the corresponding zone-number has been recognized as "25." Then we need consider only those cities in the given state which have at least 25 zones.

If we are bound to get a direct match whether we scan a big list or a little list, this process of list reduction is of secondary value only. It is when a direct match is not forthcoming that this technique assumes greater importance. In the absence of a direct match, we are constrained to use brute force techniques of a more or less sophisticated nature. If we are fortunate enough to reduce a list down to one entry, then we can avoid brute force completely. Failing this, we can expect two advantages to accrue to the use of a reduced list, in general. First, for the same elapsed time, we can employ more brute force techniques per list entry; second, we can at least hope that, by reducing our initial list, we will expunge spurious candidates to which our brute force techniques might give scores equal to, or even greater than, the score of the correct candidate. For example, Fig. 3 gives one horrible example, often quoted in this connection. Recognizing (a) as being either "New York" or "Newark" is an awful job. A non-brute-force technique, such as list reduction, which removes the false entry from consideration is a welcome way of cutting this Gordian knot.

Again for lack of time, I must give the actual brute-force techniques a very hasty treatment. Let me mention just two techniques of the many available. To match a word which has had two letters transposed [as

in (b) in Fig. 3], against the original word, we look for list entries with the same letter composition as our actual word, *i.e.*, entries with the same number of *A*'s, *B*'s, *C*'s, etc. Scanning these for a single transposition is relatively easy.

A second technique is useful when a letter or two (or more) has been erroneously dropped from, or added to, a word. (c) in Fig. 3 is due to a stuttering typist who repeated the first letter of the word. Two words run together provide further examples of this kind of noise. What we try here is a direct match of our actual word with a proper subset of our list entries, and vice versa.

(a) Newyark
(b) Pheonix
(c) Bboston

Fig. 3—Typical typographical errors.

If no amount of brute force seems to work, and certain words just cannot be recognized, we can either give up gracefully at this juncture or we can admit, even more gracefully, that one of our educated gusses might have been wrong. If we choose the latter alternative, we have the messy job of deciding whether we went haywire in the recognition phase, or all the way back in the identification phase. In either case, it is still necessary to find a likely spot for picking up the dropped stitch without causing the entire garment to unravel. Sometimes, indeed, we are left with the original ball of wool. These, however, are almost always the cases which stump human editors.

This ability to iterate back, and back, and back, can of course lead to excessive use of computer time. It does have its advantages though. It means that a bad guess is not an irrevocable misstep. It also means that various parameters, the identification acceptance and rejection thresholds, for example, are not nearly as critical as they would be in a once-through process. Since these are among the hardest parameters to estimate accurately, any diminution of their sensitivity is a positive gain.

At this point, I should like to restate our major techniques in somewhat folksier terms than "Bayes Factors" and "list reduction." In our identification phase, we attempt to use the constraints imposed by the format, mailing envelopes in our example, plus the constraints of the language itself, the length and character patterns of the expected word classes, to provide a rudimentary form of pattern recognition. We then use our *a priori* knowledge of word statistics and interword relationships to find the most probable matching of the actual words to the expected word classes. Human beings presumably use rank orderings of hypotheses, modified by intuition, to perform such matchings. Since machines lack intuition and since we have not yet developed a calculus of rank orderings, we use the paraphernalia of Bayes Factors to accomplish the same task.

Without undue stretching of the terms, we might say that, in the identification phase, we exploit the syntactic constraints on the language we are processing, whereas in the recognition phase, by our use of the list reduction technique, we exploit the semantic constraints. We might subsume both types of constraint under that much-abused word, redundancy. As for a folksy term for our brute-force techniques, the most accurate that occurs to me is "knowledgeable cynicism." We expect errors to be made and, usually, we have some information as to the kinds and sources of error, as well as their frequencies of occurrence. If we know that typists frequently hit a key next to the one they should hit, we store the keyboard pattern in our program; if we know that our character-reader frequently confuses "o" with "c," that, too, goes into our dossier.

A final word now as to the applicability and practicability of our techniques. What with tape searches and Bayes Factor computations, processing time may, but need not always, be excessive. The preparation of all the lists required in the recognition phase is a painful task. With a relatively stagnant language, this list-making can be a one-shot ordeal; with a volatile language requiring frequent updating of the lists, the pain might be unbearable. What has been said about lists also applies to the preparation of the probability tables for the identification phase. A final pause-giving consideration is the amount of redundancy in the language to be processed, particularly when the processor cannot establish the language, which he sometimes can do. Our private feelings are that a language sufficiently low in redundancy to be unintelligible to a machine will also be unintelligible to a man. I won't press a point which trods so heavily on anthropocentric toes.

In summary, then, we feel our techniques can be useful in some massive data-processing applications, in automating post offices, in translating natural languages, where every second word in the source language has several correlates in the target language, and we know our techniques have worked at least once. I won't ask that you take this on faith, though I'd appreciate it if you would.