Methods and Problems in Coding Natural language Survey Data

Rodger Knaus, American University

1. Summary

Experiments with computerized coding of natural language data indicate that agreement of 80% or better can be achieved between computerized coding and the codes assigned by experts; such performance is comparable to that achieved by entry-level coding clerks. This paper presents a class of algorithms for computerized coding based on representing semantic information in natural language constructs as vectors over the set of codes to be assigned.

2. Overview

2.1 The Natural Language Coding Problem

Natural language is useful as a response medium in surveys on subject areas such as employment, household expenditures, or health and safety. In areas such as these, the variable being investigated, (e.g. what kind of job the respondent has) set of possible responses is very large. For such large domains, natural language has the following advantages:

ECONOMY IN THE QUESTION SET: No reasonably sized set of multiple choice or other artificial response medium questions can solicit such complete information as simple natural language questions such as "Where do you work?" and "What do you do?";

OBJECTIVITY OF THE SURVEY QUESTIONS: Artificially structured questions also impose the surveyor's view of the subject upon the respondent to a greater degree than natural language questions, because the questions allow the respondent to answer in a way that reflects the respondent's perception of the subject matter rather than the surveyor's;

GENERALITY OF THE RESPONSE DATA: Should one wish to reanalyze the survey data, using, for example, a different partition of the set of possible responses, natural language responses contain the information for such a reexamination, while artificial response data usually does not;

SIMPLICITY FOR THE RESPONDENT: Natural language reduces the effort for a respondent in completing a questionnaire, because both the question and response are in a medium already familiar to the respondent. This allows instructions on how to complete the survey to be simplified. Because a single natural language question captures a large amount of information, the use of natural language responses probably also shortens a survey.

Against the advantages of NL data listed above must be weighted the difficulty in analyzing natural language data. For responses to multiple choice or other questions with a small fixed set of possible responses, there is a simple correspondence between the form of a response and its meaning. No such simple relation between form and meaning exists for natural language. Natural language (NL) data is often coded as a first step in the statistical analysis of such data. In the coding process, each response is assigned a value from some finite set (call it C) of codes. Each value, or code, in this code set represents a distinct response for the purpose of analyzing the NL data; conversely, all NL responses mapped to the same member of C are considered identical for subsequent analysis of the survey data. Coding is thus the process of filtering out the variations of linguistic expression from responses which, for the purposes of a survey, represent the same response.

In the past and in most current surveys, natural language responses have been coded by persons who assigned codes to the responses. For the large surveys, such as the U.S. monthly Current Population Survey (about 70,000) or the decennial population census (17 million responses), this hand coding is expensive, time-consuming and error-prone. These problems with human coding have motivated research in computerized coding at statistical agencies in the U.S., Sweden and elsewhere.

2.2 The Coding Algorithm

Computerized coding of natural language data can be carried out in the following way: If C is set of codes which are to be assigned to survey responses, each recognizable linguistic construct L is represented for the purposes of coding as a vector over C in which the cth component (for c in C) is the expectation that c should be assigned when L occurs. Linguistic constructs which can be recognized by computer include word roots, words, phrases and kernel sentence deep structures. There is a natural part-whole relation on these linguistic constructs, allowing one to find a set of most inclusive linguistic constructs for a given natural language response. The code assignment algorithm given below first examines the vectors of the most inclusive structures. If these are sufficient for coding, a code is assigned. If not, then the vectors of smaller linguistic structures are used to construct a vector for the given NL response. If this constructed vector implies a code, that code is assigned. Otherwise the process is continued through all vector combination methods in the current implementation of the coding algorithm, and through all linguistic structures down to word roots, until a

code is assigned, or until all possible code assignment strategies are exhausted. In psuedocode, this is expressed as follows:

```
function code(
    nlr:natural language response):code;
var s: set of linguistic constructs;
    v: vector over codes;
    cmax: set of codes
    fns: set of functions
         from a set of C-vectors
         to a C-vector;
    c:code;
begin
s:=set of maximal linguistic constructs;
c:=undefined;
fns:= set of functions
        which combine s-vectors into
        a vector for the response
        as a whole
repeat
  v:=sum of (or other function
             which combines)
     vectors of members of s
     which occur in nlr;
  cmax:=set of codes with a maximal
         v-component;
  if probability
    (cmax has a unique max. v-component)
     > a predetermined desired coding
                reliability
   then c:=cmax
   else if there is another way of
        combining s-vectors,
      let fn:= the next such method
   else  begin
    s:=set of linguistic components of
        the current members of s;
    fns:= set of functions which combine
          s-vectors into a vector for
          the response as a whole;
         end
until c<>undefined or s=nil;
code:=c;
end;
```

Geometrically, we can think of each of the codes as a unit vector and that all of these code vectors are mutually perpendicular. When given a response, we construct vectors representing the response in the space spanned by these code vectors. When the resulting response vector has a direction sufficiently similar to one of the code vectors, the code for that vector is assigned. Since the code vectors are an orthonormal basis for the response vectors, the most similar code vector under the cosine measure of similarity is that code vector having the largest component in the response vector. The vector-combining procedure template presented so far represents a variety of actual procedures depending on what linguistic structures are recognized, and how the vectors are computed and combined. A series of such particular coding procedures can be combined into one larger overall coding program by trying the particular procedures in sequence on each data record, until some particular procedure in the sequence assigns a code.

One plausible criterion for choosing and ordering procedures in the sequence of particular procedures is to start with procedures that use a great deal of linguistic knowledge. The records which remain unprocessed after applying this linguistic knowledge may reasonably be considered to be composed of statistically independent words, or other small linguistic units. Therefore the particular vector-processing procedures which occur late in the sequence can use statistical techniques which rely on the assumption of statistical independence of the small linguistic units.

2.3 Experiments with Industry Data

Work on automated coding was performed at the U.S. Bureau of the Census Programming Research Staff by Eli Hellerman and the author during the period 1976-79, as part of the Census Bureau's ongoing effort to automate the coding of industry and occupation data. As an example of automatic coding on actual survey data, results of an experiment from this work is presented.

strategy	% coded	% agreement
exact match on IA	32.0	100
exact match on IB	9.4	93.7
almost exact match on IA & IB	16.9	84.6
sum heuristic weights on IB	40.6	82.7
product conditional probabilities on IB	48.7	62.1
sum heuristic weights on IA & IB	35.9	50.0
product conditional probabilities on IA & IB	32.0	50.0

overall results: 96.4% coded, 82.2% agreement with expert-assigned code on the set where a code was assigned; the s.d. of this agreement percentage is about 2.6%.

In the experiment, each strategy was tried on all the records which remained uncoded by previously tried strategies. Therefore while this table presents an ordering of strategies which was experimentally found to produce good results on the data, it should not be used to compare different strategies, because later strategies get the generally harder records which earlier strategies fail to code. The terms used in the above table are defined as follows:

strategy: the method used to combine

vectors;

% coded: the part of the sample for which a strategy assigned a code;

% agreement: on the sample on which a strategy assigns a code, the fraction where the automatic code agrees with that assigned by an expert coder;

2.4 Experimental Coding Strategies

The strategies used in this experiment are described briefly below; a more detailed discussion of coding strategies appears later in the paper.

exact match: a code is assigned iff the response exactly matches a phrase in the coding handbook. Then the code from the handbook is assigned; Although not implemented on the computer in this manner, we may think of this as a vector method in the spirit of the overall algorithm in which the linguistic units are those phrases which have only one non-zero component. When only one of these vectors appears, that code is assigned. However the combination of two of these vectors is the null vector unless the codes are the same.

almost exact matching: A code is assigned if there is a unique phrase in the coding handbook which has the most words in common with the response. This is a matching strategy which is found useful in information retrieval, and can be thought of as a vector computation in which the vectors represent word occurrences in phrases in the coding handbook; the set of vectors to be combined are all such occurrence vectors for words in the response. These vectors have a 1 for the code of the phrase and 0 elsewhere. These vectors are combined by adding vectors for the same coding manual phrase. If there is a vector with a unique highest code for some score, that code is assigned.

sum of heuristic weights: The linguistic units are word roots. The vector for each word occurrence in the response is of the form H*V where H is a scalar called the heuristic weight of the word, defined in the section on weighting, and V is the vector computed from the conditional probabilities of codes given the word. If (p(ci/w) is the conditional probability of code ci given word w, the cth component of V is p(ci/w)**k, where k is around 0.05. The effect of raising p to this power is to make H*V into a filter which adds weights close to H to all codes with p(c/w) bounded away from 0, while adding weights close to 0 for codes with very small conditional probabilities. H*V, in other words, defines a fuzzy set of codes which are acceptable when given the word w. The fuzziness filters out codes for which the non-zero probabilities are probably the result of human coding errors in the sample used to construct the conditional probabilities.

The H*V vectors are combined by addition. A code is assigned if the best code is better than the next-best by an amount determined by a linear function of the best code score; (a possibly better, more statistically motivated criterion for this 'when to code' decision is presented below).

product of conditional probabilities: The linguistic units are word occurrences and are represented by vectors for word roots. The word root vectors are vectors of conditional probabilities of codes given the word root. These vectors are combined by multiplying their corresponding components. If there is a component in the resulting vector greater by some fixed ratio than all other vectors, then a code is assigned.

The ordering of these algorithms conforms to the principle of increasing statistical independence of the words in the data record. The exact match strategy is one which assumes that the response is an idiom, the most specific sort of linguistic knowledge. The almost-exact matching is a procedure in which lexical and semantic rules and transformations are applied to the data record in an attempt to match the kernel sentence level case grammar deep structure of the response with that of one of the phrases in a coding dictionary. In the sum-of-heuristic weights algorithm, each word may be considered as an independent weighted filter which adds its weight to the total score of the codes which the word-filter accepts. Finally, the product-of-probabilities procedure assigns each code a score proportional to its probability under the assumption that all the words occurred independently of each other, and that only one code can be assigned per data record. A more detailed description of the algorithms used in the experiment cited above appears in the 1981 paper by Knaus. More recent experiments with better results have been conducted at both the U.S. Census Bureau and the Central Statistical Office of Sweden, and are described in the papers of Appel and Lyberg.

3. Semantic Representations for Coding

One of the major problems in building an automated coding system is to find a representation of the meaning, (i.e. a semantic representation) of a linguistic construct adequate for assigning the right code. Ideally such a semantic representation should be largely free from the particular surface structure choices of vocabulary and grammer with which particular respondents encode their

responses; the same meaning, no matter how expressed, should have the same semantic representation. Once such a representation has been selected, the coding of a response can be broken down into the following steps:

map the natural language response into its semantic representation;

assign a code using the semantic representation;

3.1 Construction by Machine a Requirement

Automatic coding proceeds by comparing the information contained in a NL survey response with a database of such knowledge for the subject area of the survey (called the knowledge base of the system in artificial intelligence terminology). While many semantic representations of NL information have been proposed in artificial intelligence and linguistics, for automatic coding a representation of knowledge must be chosen which makes the knowledge base largely constructible by machine. This is because the domain of discourse of many surveys is large (e.g. all economic activity for the census industry data) and because the range of linguistic expression over different respondents (e.g. a large random sample of the general population) is also great. Knowledge representations requiring hand work on each different word, word sense, or meaning are not generally suitable where these extensive knowledge bases are required.

3.2 Vectors over Codes

The semantic representation used in our experiments with automatic coding is a real vector over the coding set C. The meaning of each word, phrase, or other linguistic construct, is, for the purposes of coding, represented by a vector over C; the value of the ci-th component represents the tendency of the construct to represent a response which should be coded to ci. For example, a phrase which always is coded to a particular code c0 might have a c0 component of 1 and all other components 0. The remainder of this paper discusses how to build these vectors for words from hand-coded data, how to combine the word vectors into vectors representing the entire NL response, and how to extract the code from the constructed vector.

3.3 Building the Database

The semantic representation vectors, called semantic vectors, or simply vectors in the following, can be built from a hand-coded sample H of data. Let L be a linguistic construct for which a vector is to be constructed. L must have the property that a computer can be programmed to recognize it reliably, although 100% reliability is not necessary. Examples of such linguistic constructs are words, word roots, phrases and kernel sentence level deep structures.

For each such construct L a count vector is constructed from the sample H of hand-coded responses. This count vector is a vector over C in which the ci component is the number of times an occurrence of of L was observed in responses in H which were hand-coded to ci. These vectors are built by passing H through a program which recognizes the linguistic constructs which occur in each record, and then increments the count for the hand-assigned code in a current count vector for each construct occurring in the record. From this count vector a normalization process can be used to construct the corresponding semantic vector. One useful normalization process is

```
semantic(ci):=count(ci)
   /[sum over ci count(ci)]
```

of assigning code ci when linguistic feature L occurs. This probability, which we will write p(ci/L), will be used extensively in our attempts at automatic code assignment. Besides hand-coded data, another source of information about how to code are the coding handbooks used by human coders. These consist of NL phrases and associated codes; the U.S. Census industry coding handbook, as an example, contains about 15,000 such phrases. This information can be recast into the above form of a vector over the codes in the following way: If the code for phrase P is c, then the cth component of the vector for P is 1 and all other components are 0.

3.4 Weighting of Vectors

The count and conditional probability vectors defined in the previous section have a direction which expresses the tendency of the construct they represent to cause a particular code to be assigned. In this section we discuss a technique for varying the length of these vectors according to their usefulness in coding. One of the ways in which vectors of components can be combined into a vector for the response as a whole is to add the component vectors. By giving a greater weight to those components which have been found to be most useful in coding, one reduces coding errors caused by the random variation among the components of vectors for components, such as the word 'company', which have little usefulness in coding.

3.4.1 The Heuristic Weight.

One way of weighting vectors, which we call

the heuristic weight, is based on the entropy of the distribution of codes for a given component; (in the Census experiments, the linguistic components for which the heuristic weight was computed were word roots, but the computation works for all vectors over C, regardless of what the vectors represent.) Let CO be the count vector for the feature 'is a survey response', i.e. the ci component is the number of responses in our hand-coded sample which have code ci; let V0 be the corresponding vectors of probabilities of assigning ci. Now if Vc is the probability vector for a construct c, Vc represents a construct useful for coding if Vc is not similar to V0; however, if Vc is nearly codirectional with V0, then Vc is of little use in coding, because the probability of c occurring is independent of what code is assigned. If the cith components of V0, Vc are v0i, vci, then we form probabilities pi' as follows:

Let vci':=vci/v0i;

pi':=vci'/sum(vci');

The pi' represent the probabilities of assigning code ci when given construct c under the conditions that the conditional probabilities of assigning ci when given c remained the same, but the distribution of codes in the sample as a whole is made uniform. As the next step in computing the heuristic weight, we compute the entropy E' using the uniformized probabilities pi':

E'=sum pi'*ln(pi')
   over all pi'

This is the entropy of the distribution of the vector Vc when the non-uniformity of the overall distribution V0 has been normalized out. E' is a minimum when Vc is similar to V0 and is approaches 0 when Vc has a large probability for a code which is is rare in the sample as a whole.

The final step in the heuristic weight computation is to transform E' in such a way that constructs that are useful in coding have a large positive weight, while those which are useless have a weight near 0. This means that we want a transformation which maps the normalized entropies E' near 0 into large heuristic weights; conversely, entropies near the minimum of E', (which is a constant ln(1/n) depending on n, the number of codes) are to be mapped into a small interval around 0. The function

H=(Eu-E')/E"

has this property, where

Eu is the entropy of the uniform distribution over C;

E" is the entropy E', modified slightly if necessary to insure that E" is non-zero. In the case where E' is zero, a small random error is added to the distribution Vc. The added error is an estimate of the probability that c would be encountered in a response hand-coded to a code different than that encountered so far. This addition of a random error is justified by the nature of the data, because words sometimes appear in unusual contexts as proper names, as used by persons with limited English, or as transcription errors; therefore a very large sample would contain few if any distributions with zero entropy.

The heuristic weight computed by this method appears to agree with our intuitive notion of how specific a word is for coding. Where the coding is over a set of about 250 (U.S. Census-defined) industry classes, some heuristic weights defined from a hand-coded sample of around 100,000 are given in the following table:

co.	1.78	citrus	7.07
plant	1.85	shoes	7.25
service	1.98	hospital	9.17
metal	3.80	liquor	12.0
medical	4.10	beer	12.3
iron	4.35	airline	58.6
farm	4.60	turbine	60.0

The heuristic weight can be viewed as a generalization of of the inverse document frequency weight for terms in information retrieval. In the information retrieval case, there are just 2 categories, relevant and irrelevant, into which documents are to be assigned. In the case of just 2 categories, the inverse document frequency and heuristic weight are approximately proportional for the case where one of the categories (e.g. the relevant documents) as very low probability.

3.4.2 Other Methods of Weighting Vectors. Alternatively, one might weight each vector by the correlation over all responses in the hand-coded sample between the hand-assigned weight for a code and the weight assigned by the vector. More particularly, for each response r and code c let h(r,c)=1 iff the hand-assigned code of r is c, and h(r,c)=0 if the hand-assigned code is not c. Let v(r,c)=the cth component of v, i.e. the probability of code c given the construct represented by c. Then if there are #C codes and #R responses, there are #R*#C (h,v) pairs, over which we compute the correlation coefficient: Starting with the usual formula for the correlation coefficient and applying some algebra, we get

r=(#C*sum(vc**2)-1)**(1/2)
      codes
  /(#C-1)**(1/2)

This correlation coefficient is seen to be 0 for the uniform distribution, and one

for a vector with just 1 non-0 component. For vectors between the uniform and single-valued extreme, the correlation coefficient is between 0 and 1.

Just as for the heuristic weights of the previous section, the correlation coefficient should be computed with a vector of probabilities which have been uniformized with respect to the distribution of codes in the sample as a whole, so that a vector with probabilities similar to the sample as a whole has a computed correlation coefficient of 0. This normalization is important because some codes may occur much more frequently than others.

4. Constructing Vector Representations

In the overview section of the paper, the vector representations and the combination methods actually used in experiments were described. In this section the relation between the methods are discussed and some refinements are suggested.

4.1 Relative Coding Effectiveness

4.1.1 Phrase Matching. There is a saying in the advertising industry that a smart dime never beat a stupid dollar. The same applies here: nothing beats phrase matching, where exactly the entire response, at least up to trivial variations, is matched against an entry in a coding lexicon. This method is fast and reliable. It was not used to maximum effectiveness in the Census experiments, because of an unwillingness to add to the coding manual online. In a production system, one should constantly add new phrases to the online coding handbook because new phrases are constantly coming into use; the coding handbook used in the experiment reported above, for example contained 'tourist cottage' but not 'computer store'. One method of identifying new phrases to add to the coding handbook is to write out to a special file the complete text of responses which are not phrase-coded. One may sort these to identify common phrases not in the coding handbook and present the list to experts, who can identify phrases and associated codes that should be added to the lexicon.

Phrase matching as an initial coding method also increases the statistical independence of words in the residue of records not coded by phrase matching. This independence is an assumption behind the product scoring and the error estimation for linear scoring.

4.1.2 Addition versus Multiplication. In the experiment, both addition and multiplication were used to combine vectors for words into vectors for phrases. Weighted sums generally were good at identifying possible codes but were more error prone, swayed by a single word which was strongly associated with a particular code. The product of conditional probabilities method, on the other hand, avoided these errors, but sometimes failed to identify codes that should be assigned, because the ration between best and next codes failed the coding criterion. Although not tried in the above experiment, it would be reasonable to try these two methods together in a generate-and-test algorithm similar to that used in many artificial intelligence programs. The weighted sum would be used to suggest one or a small set of possible codes. If one code were so suggested, it would be assigned only if the product score was sufficiently good relative to the product scores of other codes. This would eliminate weighted sum assignments which were based on only part of the response which was highly related to a particular code. If several codes were suggested, some function of sum and product codes would be required to pass a criterion function before coding occurs.

4.2 Linguistic Refinements

In the experiment, only phrases and word stems were used. However, other linguistic features can be reliably recognized by computer and might improve the performance of the code assignment strategies.

4.2.1 Case Grammar. In a semantic theory that has become widely used in computerized language processing, Fillmore noted that simple sentences consisted of a verb and a set of arguments, expressed as noun phrases, which stand in fixed semantic relations to the verb. Viewed in this way, a simple sentence has the following semantic parts

action: the action that takes place, described by the sentence verb;

object: the thing which is affected or changed by the action;

source: the environment or state, particularly of the object, before the action;

location: the environment or state, particularly of the object, during the action;

destination: the environment or state, particularly of the object, after the action;

agent: the thing which causes the action;

instrument: the thing which is used in

carrying out the action;

We note that this list of case constituents is typical and suitable for automatic coding, but that linguists differ on exactly what the cases are and how they are defined; Natural languages use word order, word endings, function words and standardized patterns of case cooccurrence to mark the case of noun phrases in a sentence. For example, in English the object in a simple sentence appears after the verb without a preceding preposition.

4.2.2 Word Uses. Case grammar can be applied to sentence fragments as well as sentences. The grammar of such fragments, however, is a function of the question which the fragments answer. As is true most linguistic data, respondents choose a grammatical form which eliminates redundant information and which places information known to the questioner and present in the response before information in the response new to the questioner; (this is the "given-new" principle in linguistics.) For example, in response to the question 'Where do you work?', the answer is often a free-standing noun phrase which is a location in the sentence which would describe the activity of the worksite. Another common response to this question is a nominalized verb plus object. Inspection of the data confirms the general linguistic observation that the form of the question very tightly constrains the grammatical form of the response; in the case of the industry census questions, a few grammatical forms cover all but a few records.

Survey responses are typically very short sentence fragments; 4 words or less were typical in the industry data of the experiment. Computer programs which make use of endings, word order and other syntactic features can identify the case-grammar function of most words in the response. Furthermore these programs can decide if a word in a noun phrase is a head noun or a modifying word. We will define a word use as a triple (w,c,hb), where w is a word, c a case grammar function and hb either head-noun, modifying word, or not applicable (for verbs). After processing with the appropriate linguistic analysis programs, a response may be considered to be a set of word uses.

Using such marking of word occurrences with case function and head-noun or modifying word for noun phrase words, we can build and use conditional probability vectors for word uses in the same way in which such vectors were built and used for word stems. However, for many words, informal hand inspection of the data suggests that such vectors for the same word but different uses would vary considerably in direction from one another. For example 'farm' as a head noun in a free-standing noun phrase answering 'Where do you work' is for industry coding very heavily associated with the agriculture code. On the other hand, 'farm' as a modifying noun is much more commonly used in responses which code to machinery and supplies used by farms, i.e. responses with codes other than agriculture. It would appear, from this and other similar examples, that word uses are better predictors of codes than word roots.

4.2.3 Features Based on Word Uses. The basic principle of the coding algorithm presented in the 'overview' section is that when a large constituent determines a code then it is assigned before proceeding to smaller constituents. This principle can be employed for word uses by building a coding dictionary in which the entries are sets of cooccurring word uses and an associated code. Such a dictionary would be consulted before attempting to code using sums or products of word use vectors. This dictionary would be similar to the ordinary coding dictionary but would allow for more variation in linguistic surface structure in responses which can be successfully matched against the dictionary. Another refinement based on word uses is to subdivide the head-noun versus modifying word distinction. One might classify modifying words in noun phrases according to the sum of the weights of the words which come after them in the noun phrase. Alternatively, one might define an inclusion relation on vectors such that $V1 \leq V2$ if every code that is plausible given V1 is plausible given V2; then one might distinguish between modifying words which precede a more inclusive word and those which do not. The motivation for this distinction is found in phrases like 'grocery store', in which 'grocery' functions as if it were a head noun; generally, those words followed by more inclusive words have a behavior which is more like that of head words than is the behavior of words which precede words that do not include them.

5. Assigning Codes to Survey Responses

If V is a vector which represents the response r using the set of linguistic constructs, and c is a code, then the cth component of V is a real number which we will call the score of c in V for r using s. The vector V will sometimes be called a scoring vector. A code with the highest score among the c in C will be called the best code. Our basic code assignment algorithm assigns the best code to a survey response represented by V if this best code has a score sufficiently better than the other scores. In this section we consider some methods for making the decision about whether the best code is sufficiently better-scoring than the rest.

In deciding whether an automated coder has assigned the right code, our

only available criterion is agreement with a hand-assigned code. This is in fact a correct criterion only when the hand-assigned code is correct. In research on automatic coding, it is important to have a sample of hand-coded responses containing as few errors as possible. By having the sample hand-coded by experts, or even a panel of experts, the number of wrong hand codes (perhaps definable as codes later rejected by the same or other experts), can be reduced but not eliminated for an area as complex as industry and occupation coding; indeed, for some responses, there is more than one acceptable code. In assessing the actual accuracy of an automated coder, one must decide in cases of disagreement with the hand code, which code is correct or better, or if both are acceptable. Preferably this evaluation should be done by experts who are blind to which code is automatically assigned, to eliminate any prejudice against the automatic codes.

While keeping the above limitations of hand coding in mind, we will use right code as a convenient shorthand for 'the hand-assigned code' and wrong code as a shorthand for 'a code not equal to the hand-assigned code'.

5.1 When to Assign a Code

5.1.1 Kinds of Scoring Errors.

One problem in automatic coding deciding when the score of the best code is sufficiently better than that of the others to justify assigning a code. In general it is observed that as the spread between the scores for the best code and next-best code increases, the probability of the best code agreeing with the hand code increases, but that there are a few wrong codes with high scores. These disagreements with the hand codes are of several types. In some cases the hand codes are in error. Another source of code disagreement is statistical scoring error, the probability that in our particular hand-coded sample, the true best code appears as the next-best. Stated another way, the statistical scoring error is the probability that the various sample probabilities are such that the sample score of the true best code (i.e. the one that would score best were the probabilities computed over the entire population of responses) is not the highest score.

In the early experiments performed while the author was at the Census Bureau, a fixed linear function involving the best and next-best codes was used as a coding criterion for all records in a given sample. There was no clear relation between these when-to-code functions and the resulting fraction of coding errors. However, some statistical and computational techniques allow one to get a record-dependent estimate for that part of the chance of miscoding that related to statistical errors in the best and next-best scores. We can use these estimates in the code assignment process to control the level of errors due to statistical scoring errors; this is done by assigning a code when and only when the probability of an error due to a statistical scoring error is below some preset level of errors.

In addition, however, there is an additional error of miscoding which is not included in these estimates, i.e. the error that the highest scoring code is truly the highest scoring code but is still wrong. This component of the coding error can be estimated experimentally by comparing the observed coding error after running an automated coder on a large sample of data with the expected level of statistical scoring errors. In estimating the statistical error in coding, we will reduce the problem to that of estimating the error in assigning the best instead of the next-best code. In the case where there are more than 2 close contenders, the pairwise error estimates can be used to get an error estimate for one of a small set of next-best codes, and in the case of a large set of such next-best codes, coding is obviously very risky.

5.1.2 Estimating Errors as a Linear Sum of Random Variables.

In the case where the scoring vector is a weighted sum of conditional probabilities (of codes given constructs) vectors, then a particular code is highest-scoring when and only when some weighted sum of conditional probabilities is > 0. In particular let b,n be the best and next-best codes, V=sum(ai*Vi) and vib, vin the conditional probabilities of b and n in Vi. b is assigned when

sum(ai*vib)-sum(ai*vin)>0.

When the probabilities in the above are such that no one of them is close to 1, the ai coefficients, if they were computed either as heuristic weights or as correlation coefficients, are such that they are stable under changes in the probabilites {vib,vin}, provided that such changes are such that the probabilities stay out of some interval around 1. Therefore we can approximate the statistical coding error in the above inequality by assuming that the a's are constants and looking at the expression on the left as a linear sum of random variables {vib, vin}. The variance of the linear sum is computable in terms of the a's and the variances and covariances of the random variables. In the region of stability we may assume that the covariances are 0. The variance of the left side of the inequality is then

var(b over n)
  =sum(ai**2*(var(vib)+var(vin))

The variances of the (vib, vin) can be computed using formulas for the variances of proportions in a binomial or approximating normal distribution, so that the variance of the inequality expression is computable from available information. This variance allows us to estimate the probability that the score of n would be >= that of b, which is an estimate of the statistical coding error in assigning the code b rather than n. While the above variance and associated probability estimate is often best left to a computer, the method is illustrated in this simple example: Suppose the response is 'auto repair', both words have a heuristic weight of 4 computed from a sample of 1000 occurrences, and the probabilities of various codes are given by

Table of p(code/word)

code	'auto'	'repair'
auto mfg.	.3	0
auto service	.3	.3
electrical repair	0	.15

Then the best code is 'auto service' when

```
wt(auto)*p(auto service/auto)
 +wt(repair)*p(auto service/repair)
 -wt(repair)*p(electrical repair/repair)
 >0
```

The variance of the value of that expression is

```
wt(auto)**2*var(auto service/auto)
 +wt(repair)**2*var(auto service/repair)
 +wt(repair)**2
    *var(electrical repair/repair)
=3*16*(2*2.1*(10**-4) + 1.3*(10**-4))
=2.64*10**-2
```

so the standard deviation of the value of this expression is 0.163. The value 0 is about 6.7 s.d.'s from the observed value of 1.1, so that the statistical error in this code assignment is very small.

However, if one had a similar set of probabilities but the number of observations per word was only 25 (for example with a word pair like 'canvas awnings'), then the s.d. is increased by a factor of sqrt(40), and becomes 1.03. Then the value 0 is about 1.07 s.d.'s from the observed value of the expression, and the statistical coding error is about 14%.

While this example is a made-up one, the numbers are typical of the sample sizes and probabilities which arise when constructing conditional probability vectors from a large sample of actual responses. The example illustrates the extreme variation in statistical coding error based on the sample probabilities used in coding. By illustrating this variation in statistical error between records, the example strongly suggests that coding performance can be improved by using a boolean "when-to-code" function in the computerized coder which estimates the statistical coding error.

5.1.3 Error Estimates in the Unstable Region. When the distribution of probabilities in a vector is such that one probability is near 1 while the others are very small, the coefficient (heuristic weight or correlation coefficient, for example) may be significantly affected by small changes in the large probability. In this case, the above method, which assumed that the coefficients were for practical purposes constant under changes in the probabilities, does not apply. As an alternative method for the unstable case one may

Estimate from the given vector V the probability p that the next occurrence of the construct represented by V will not have the single high-probability code;

If p is sufficiently small, ignore the possibility of a change in the large probability;

Otherwise compute the statistical error as the probability-weighted average of the case where the large probability remains unchanged and the case where the greatest other probability is incremented by the addition of a single occurrence to the sample with that next-highest code in V. In the two subcases, the statistical error is computed by with V assumed constant, but vectors other than V which have not been assumed constant at some previous subdivision of the computation allowed to vary.

6. Computer Implementation

At the time these experiments were conducted, hardware was considerably more expensive and the hardware options confined to large computers. Processing time was expensive, and the concern with processing time per record, about 1 sec. for the algorithm in the experiment, prevented more elaborate coding experiments. In addition, development and experiment with the program was hindered by the long waits and down time associated with a heavily loaded time-shared computer.

6.1 Coding on a Local Area Network

Today, all of these limitations can be overcome. 16-bit micros in a local area network could be used to code during the data entry task. For example we might use the following system:

shared hard disk: The storage requirements would be around 40 million bytes for word-use vectors for industry coding, plus another few million bytes for the coding

handbooks. A 70 mb. hard disk for $7000 has recently been announced.

work station micros: These are 16-bit micros; if used only for coding, floppy disk drives would not be needed, so that the cost per micro would be around $2500.

additional ram for the micros: A large electronic disk is added to the workstation micros so that common phrases and words can be stored locally to reduce network traffic and improve response time. Currently 1 mb. of ram is available for $1600, and prices should continue to drop.

communications hardware: For each workstation and for the disk controller, a network interface board, about $1000, is needed.

mainframe link: A link with a mainframe is needed to get data to be coded and to offload coded records. This link might consist of a high speed data line and interface hardware; a very rough cost estimate might be several thousand dollars.

A local area network like the above should be able to support 20 users with a response time that would appear almost instantaneous, assuming that as each word was entered, some processing occurred while the user typed the rest of the record. The per-user cost would be around $6000. The fast response time and sophisticated programming language software available in this environment would also be ideal for developing coding software.

6.2 Advantages of Coding during Data Entry

6.2.1 Elimination of Errors.

There appear to be several advantages to coding during data entry. For one thing, the entry clerk can use some judgement in the order in which parts of the response are entered, entering what appear highly descriptive before usually meaningless parts, such as proper names. If coding occurs before the whole record is entered, the rest never has to be keyboarded. This discretionary entry also prevents the machine from being distracted by proper names which it, in its limited ability to understand language, does not recognize as such. Additionally, the computer can catch many probably spelling errors at a time when the entry clerk can correct them. In the experimental data, some records were not coded correctly by the machine because of such coding errors.

7. References

Appel, Martin, and Eli Hellerman. Census Bureau Experiments with Automated Industry and Occupation Coding. Joint Statistical Meetings, Toronto, August, 1983.

Biggs, J. Coding performance in the 1970 census. in Evaluation and Research program PHC(E)-8, 1970 Census of Population and Housing, U.S. Bureau of the Census, 1974.

Freund. J. and R. Walpole. Mathematical Statistics. Prentice Hall., Englewood Cliffs, N.J., 1980.

Hellerman, Eli. Overview of the Hellerman I&O Coding System. draft memo, Bureau of the Census, Washington, D.C., 1982.

Janas, J.M. Automatic recognition of the parts of speech for English texts. Inf. Proc. Man. 13, 205-213, 1977.

Knaus, Rodger. Pattern-based semantic decision making. in Rieger, B. (ed.): Empirical Semantics, Brockmeyer, Bochum, W. Germany, 1981.

Lyberg, Lars. Control of the Coding Operation in Statistical Investigations. Statistiska Centralbyraan, Stockholm. 1980.

Lyberg, Lars. Automated Coding at Statistics Sweden. Joint Statistical Meetings, Toronto, August, 1983.

Mendenhall, W. The Design and Analysis of Experiments. Duxbury Press, (Div. of Wadsworth Publishing), Belmont, Ca., 1968.

Moskovich, W. Distributive-statistical techniques in computational linguistics: problems and perspectives. 7th International Conference on Computational Linguistics, Bergen, Norway,1978.

Rieger, B. Feasible fuzzy semantics. 7th International Conference on Computational Linguistics, Bergen, Norway,1978.

Rustin, G. (ed.). Natural Language Processing. Algorithmics Press, N.Y., 1973.

Salton, G. Dynamic Information and Library Processing. Prentice Hall., Englewood Cliffs, N.J., 1975.

Sneath, P. and R. Sokal. Numerical Taxonomy. W.H. Freeman, San Francisco, 1973.

Winograd, Terry. Language as a Cognitive Process. Addison Wesley, Reading, Mass. 1983.