Journal of Official Statistics
Vol. 3, No. 1, 1987, pp. 45–67
© Statistics Sweden

Methods and Problems in Coding Natural Language Survey Data

Rodger Knaus[1]

Abstract: This paper discusses a computer algorithm for coding, i.e., classifying, natural language survey data. The algorithm uses "semantic vectors" over the set of codes to be assigned. The database for the algorithm can be automatically constructed from manually coded records. When applied to industry descriptions from the 1970 U.S. Population and Housing Census, the algorithm agreed with expert manual coding in 80 % of the cases. The agreement rate of 80 % for the industry data is comparable to the rate of agreement between a novice and an expert coder.

Key words: Coding; classifying; natural language; survey data; semantic pattern matching.

1. Coding Using Semantic Vectors

1.1. The natural language coding problem

Natural language (NL) is an especially useful response medium in surveys on employment, household expenditures, or health and safety. For these subjects, the variable of interest usually has a large set of possible values and we can expect a large number of different responses. Furthermore, the values for certain variables, like "type of job," are normally described in words and cannot be described by numbers or any other more structured data without first deciding on a coding method.

By our definition, *natural language variables* (NLV) are survey variables that can assume a large number of values. For NLVs, the only adequate notation is in the natural language itself. We also contrast responses in natural language to responses in an *artificial medium*, such as a code number chosen from a table. An artificial medium for a survey response is a finite set of allowable answers defined by the surveyor. Questions whose answers are in artificial mediums are called *artificial questions.* According to this definition, multiple choice questions are an example of artificial questions.

For natural language variables, natural language has the following advantages over an artificial medium:

Economy in the question set: No reasonably sized set of artificial questions can solicit information as complete as simple natural language questions like "Where do you work?" and "What do you do?"

[1] Instant Recall, Box 30134, Bethesda, MD 20814, U.S.A.

Acknowledgement: The author acknowledges the support of the Dept. of Mathematics, Statistics and Computer Science of American University in preparing this paper.

Objectivity of the survey questions: With natural language, the respondent can answer in a way that reflects his or her perception of the subject. In contrast, for artificial media, the survey designer decides the possible answers and the respondent selects one of those answers. The flexibility of natural language can be both an advantage and a disadvantage. On the one hand, the flexibility of natural language permits an uninformed respondent to provide an uninformative answer (e.g. occupation = helper). On the other hand, the same flexibility might provide a more accurate answer. A structured question to computer professionals might ask whether they work primarily with hardware or software, whereas a natural language question would allow them to reply with "firmware" or "communications."

Natural language responses can be reanalyzed using different similarity criteria to answer different questions. For example, systems analysis and data entry might be mapped to the same type of industry (data processing) in an analysis of economic sectors, but to different educational levels in a survey analyzing educational requirements. In this way, natural language data can be reused for different analyses, while artificial media generally do not support such reclassification.

Simplicity for the respondent: Questionnaires using natural language questions are easier for the respondent to complete. Both the question and response are in a medium already familiar to the respondent. This simplifies the survey instructions. Because a single natural language question captures a great deal of information, the use of natural language responses probably also shortens a survey.

The advantages of NL data discussed above must be weighted against the difficulty of processing 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.

Coding is usually the first step in processing census or survey data collected in natural language form. In the coding process, each response is assigned a value from some finite set C of codes. Each value or *code* in the *code set* represents a distinct response. Coding makes the analysis of NL data possible. Conversely, all NL responses mapped to the same member of C are considered identical for subsequent analysis of the survey data. Thus coding is the process of filtering out the variations of linguistic expression from responses that, 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. This method of assigning codes is called *hand coding*. The U.S. Current Population Survey (70 000 households) conducted monthly and the decennial U.S. population and housing census (over 10 million households) are examples of large data sets where the coding of, say, industry and occupation is an extensive operation. For large data sets hand coding is expensive, time-consuming, and error-prone. It is these problems with hand coding that have propelled research in computerized coding at the U.S. Census Bureau, Statistics Sweden, and elsewhere.

1.2. The coding algorithm

This paper deals with a class of computerized coding algorithms called *vector coding (algorithms)*. These algorithms have a number of features in common.

A vector coding algorithm starts by recognizing a set of *linguistic constructs* in the response to be coded. The simplest linguistic constructs are words, and in the phrase "citrus grove" the words "citrus" and "grove" can be recognized. The phrase "citrus grove" is itself a linguistic construct, and might be recognized as such by some vector coding algorithms.

More complicated linguistic constructs have not been used in any of the implemented algorithms, but could be included without changing the basic method. These more complex constructs might include syntax trees, e.g.,

noun phrase (head noun(grove), adjective (citrus)),

and case grammar deep structures (see Section 3.2.1).

Vector coding algorithms use a coding dictionary that contains a vector over codes for each construct in the dictionary. The vector for a linguistic construct L is called its *coding vector*. Each coding vector lies in a vector space over the set C of codes which can be assigned. If c is one of these codes, the cth component of the coding vector of L represents the tendency to assign the code c when L is present.

For example, suppose that we have the following industry codes:

0	agriculture and forestry
1	mining
2	manufacturing
3	utilities
4	transportation and communication
5	wholesale trade
6	retail trade
7	business services
8	personal services
9	government

Then the vector for "citrus" might look like this:

$$(0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0).$$

This indicates that "citrus" appears 95 % of the time in responses coded to agriculture, and 5 % of the time in responses coded to manufacturing. Similarly, vectors for "grove" might be:

$$(0.50, 0, 0, 0, 0, 0, 0.50, 0, 0),$$

indicating that "grove" occurs in descriptions both of agricultural enterprises and also of business services (e.g., grove caretakers). In the same way, the vector for "cannery" might be

$$(0, 0, 1.00, 0, 0, 0, 0, 0, 0, 0),$$

indicating that "cannery" always appears in responses coded to manufacturing. In practice, the empirically built vectors almost always contain some noise (small random components near zero, from word occurrences like "Grove Press") not shown in these examples.

Given the ability to recognize some linguistic constructs and a dictionary of coding vectors, a vector coding algorithm works in the following way.

The algorithm first recognizes the linguistic constructs in the input response. It then looks up the coding vectors for the linguistic constructs which were found, and adds up these coding vectors (possibly with weights, as discussed later in the paper). It finds the largest component and next largest component in the resulting sum vector.

If the largest component is enough larger than the second largest, then the response is assigned the code corresponding to the largest component. (How much larger the largest component should be is discussed later in the paper.)

Suppose the response to be coded is "citrus grove" and that the dictionary contains the three vectors used as examples above. Then adding the vectors for "citrus" and "grove," and ignoring the problem of weighting the vectors for now, one gets:

vector ("citrus grove") =
(1.45, 0, 0.05, 0, 0, 0, 0, 0.50, 0, 0).

The best code, agriculture, has a component of 1.45 which is considerably better than the runner-up, business services, at 0.5. Therefore the code *agriculture* is assigned to this response.

A set of vector coding algorithms that carry out different levels of linguistic analysis can be stacked one after the other to form a composite algorithm. The earlier algorithms in the list generally perform less linguistic analysis than later algorithms in the list. If an early algorithm can assign a code on the basis of this limited information, it does so. If it cannot assign a code it passes the response to a later algorithm in the list, which performs a more detailed linguistic analysis.

The simplest linguistic construct that can be recognized in a response is the response itself, taken as a phrase. If the phrase is found in the coding dictionary, for example,

coding vector("citrus grove") =
(1, 0, 0, 0, 0, 0, 0, 0, 0, 0),

a code can be assigned without looking up the various words and adding their vectors together. On the other hand, if no vector is found for "avocado grove" in the dictionary, then the phrase-level coding algorithm fails to assign a code, and the response "avocado grove" is passed on to a word-level coding algorithm.

A generalized vector coding algorithm, consisting of a list of vector coding algorithms that work from less detailed to more detailed linguistic analysis, is expressed by the following pseudocode:

```
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 code
        vectors to a code vector;
    c: code;
begin
s:=set of highest level linguistic constructs;
c:=undefined;
fns:=set of functions which combine vectors
      for members of s into a vector for the re-
      sponse as a whole
repeat
  v:=sum of (or other function which com-
      bines) 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 reli-
    ability then c:=cmax
   else if there is another way of combining
    vectors for s components,
    let fn:=the next such method
   else begin
    s:=a more detailed set of linguistic com-
        ponents;
    fns:=set of functions which combine the
          vectors for s into a vector for the
          response as a whole;
          end
until c <> undefined or s=nil;
code:=c;
end;
```

To summarize this algorithm in more concrete terms, the NL input is analyzed for recognizable linguistic components, usually words and phrases. Then a database of pre-constructed vectors over the set of assignable codes is searched for vectors corresponding to the recognizable components. Each vector represents the meaning of the construct for coding purposes and has large components for those codes that are often assigned when the construct appears. Any such retrieved vectors are weighted and summed. The weights reflect the importance, estimated from syntactic features, of the construct in the response being coded. The resulting vector represents the meaning of the whole response. If this vector is sufficiently similar (codirectional) to any of the vectors for individual codes, that code is assigned. Otherwise the process is repeated, if possible, with a set of smaller linguistic features. For example, if no vectors are retrieved at the phrase level, the vector database is searched for individual words. This process of constructing vectors for the response and comparing them to the code vectors is repeated until either a code is assigned or all recognizable linguistic features have been used without success.

1.2.1. Geometric interpretation of vector coding

Using a geometric interpretation, each code is a unit vector and 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.

The directional similarity, represented by the cosine of the angle between two vectors **A** and **B**, can be computed from the inner product $\mathbf{A}*\mathbf{B}$ and the lengths, $|\mathbf{A}|$, $|\mathbf{B}|$,

$$\cos(\text{angle between } \mathbf{A} \text{ and } \mathbf{B}) = \mathbf{A}*\mathbf{B}/|\mathbf{A}|*|\mathbf{B}|.$$

Now suppose that one of these vectors is a unit code vector **C**. **C** has a 1 in the position for code C and 0s elsewhere. Then

$$\cos(\text{angle between } \mathbf{A} \text{ and code } C) = \mathbf{A}_C/|\mathbf{A}|,$$

where $\mathbf{A}_C$ is the cth component of **A**.

The general principle in coding using vectors over codes is to assign a code when the angle between the response vector and a code vector is sufficiently small, or equivalently, when the cosine of the angle is sufficiently close to 1. In practice, however, a simpler test can be used for assigning a code. Assign the code when the ratio,

<2nd. largest component> / <largest component> < = <some constant <1, (e.g. 0.75)>.

This test is computationally simpler than testing the cosine, and is similar in practice for the following reasons. Since there are a fixed finite number of components to all the code vectors, the requirement that the largest code in a vector **A** must exceed all other codes by a fixed ratio places a lower bound on the cosine. In practice, there are usually only a few components of any size to a response vector over the codes. When there are only a few components of any size, the above ratio test with the constant 0.75 almost always guarantees that the angle between the response vector and the code vector is less than 45°.

While the upper bound on the angle between response and code vectors does not seem very strict at first, the response vector for even the typical text description assigned to a particular code will contain components of other codes. Most words and phrases used in a typical description of one code are also used to describe items assigned to other codes. For example, "farm" is used to describe enterprises classified as agriculture, but also in phrases like "farm machinery" that describes enterprises engaged in manufacturing, wholesale, and retail trade. Therefore, the response vectors for a code C do not cluster around the unit vector for the code C, but around some other vector $\mathbf{C}'$. $\mathbf{C}'$ includes components of codes other than C. $\mathbf{C}'$ can share descriptive words and phrases with the code C. The average response vector $\mathbf{C}'$ for the code C is more highly correlated with the typical response vector for a code C than the unit vector for C is.

In addition, the probability that a particular random response vector will be correlated even with 45° of a particular unit code vector is very small for a vector space built over a large set of codes (e.g., 200 or more).

The abstract formulation of the vector-combining coding procedure presented so far is really a procedure template; it represents a variety of actual procedures depending on the linguistic structures that are recognized, and how the vectors are computed and combined. A series of these particular coding procedures can be combined into one larger overall coding program. This is done by trying each particular procedure in sequence on each data record, until some particular procedure in the sequence assigns a code.

1.2.2. Ordering a set of vector coding algorithms

In a coding algorithm built from a sequence of vector coding algorithms that use different levels of linguistic analysis, it makes sense to start with the least possible linguistic analysis. For example, the response can be considered a single phrase, and if found in the coding dictionary, the code with the largest component in this phrase vector is assigned.

Phrase level coding works well when the response consists of a phrase in the dictionary, which usually means that it is used frequently enough by respondents to be included in the dictionary. Frequent phrases are often idioms, i.e., phrases that are used as a unit. For example, "police station" and "computer center" are much more common in English than "police center" and "computer station", although there is little difference in meaning between "center" and "station" as used here. However, the words "station" and "center" are not freely chosen, but fixed by habit and familiar usage in the English language. Accordingly, the words in many phrases that typically describe certain kinds of establishments are not independent of each other, but are fixed by idiomatic usage.

Even if all idioms were in the phrase dictionary, some responses would contain phrases that were not in the phrase dictionary. These remaining phrases represent concepts for which no idiom exists. For such concepts, the words chosen in a particular response depend on what the response is to describe. Nevertheless, the words in the response vary within a certain semantic constraint. For example, "wooden boats," "wood boats," or "wood rowboats" describe the product of a small-boat builder. The occurrence of each word in this phrase does not determine the other words. In general the words in a non-idiomatic phrase are chosen by the respondent for their contribution to the meaning of the phrase.

For phrases not found in a fairly complete phrase dictionary, the individual words can be treated as if they were statistically independent. Strictly speaking, the words in non-idiomatic phrases are not completely independent. For example, in a phrase "wood ...," "boats" is much more likely to fill the blank than "computer." However, there are many things that are made from wood, so that when the population of responses describes the economy as a whole, the probability of "boat" given "wood" is small, even though it is much more probable than "computer" given "wood." Common descriptions of common kinds of establishments (e.g., "high school") would appear in any reasonably complete phrase dictionary. For uncommon descriptions or uncommon enterprises, the words in the descriptions are non-idiomatic. Because words in non-idiomatic phrases are used in the industry data to describe many diverse enterprises, of which a particular enterprise is a relatively rare occurrence, all these conditional probabilities relating non-idiomatic words are small. Therefore, we can assume that after the responses to be coded have been scanned by a good phrase dictionary, the words of the still uncoded responses are statistically independent. For this reason, the particular vector-processing procedures that occur late in the sequence can use statistical techniques which rely on the assumption of statistical independence of the small linguistic units.

1.3. Experiments with industry data

Experiments with computer algorithms for coding natural language data were performed at the U.S. Bureau of the Census Programming Research Staff by Eli Hellerman and the author during 1976–1979, as part of the Census Bureau's ongoing effort to automate the coding of industry and occupation data. As an example of computerized coding on actual survey data, the results of an experiment from this work is presented.

As mentioned above, the coding algorithm used a succession of strategies. Each strategy assigned a code to a record if the best code assignment from that strategy passed an acceptance criterion such as the <next>/<best> code ratio. Records not coded by a particular strategy were passed on to the next coding strategy in the sequence.

The strategies used in this experiment were the following.

1. In an *exact match*, a code is assigned if the response exactly matches a phrase in the coding handbook. In this case, 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, like the overall algorithm, in which the linguistic units used in constructing a response vector are phrases. Furthermore, only those phrases are included in the coding database which describe items with a common code. When only one of these vectors appears in a response, the single nonzero code for the phrase is assigned. If two phrases with different codes appear in a response, the resulting response vector fails the "next/best" ratio test, and no code is assigned.

   For example, the coding dictionary contains the phrase "orange grove," with the associated code *agriculture.* This associated code may be seen as the single code among the entire set of codes for which the phrase "orange grove," has a nonzero component in its coding vector. If a response "orange grove" was encountered in the data, it would match the entry "orange grove" in the coding dictionary, and the associated code, agriculture, would be assigned.

2. In an *almost exact match,* a code is assigned if there is a phrase in the coding handbook which matches the response (overlooking semantically unimportant variations). One example of an unimportant variation is the presence of words such as "co." at the end of a response.

   Another semantically unimportant variation is the inclusion of a proper name. Many company names, gathered as responses in population census forms, have a phrase in the coding dictionary preceded by a proper name. An example of this would be the response "Adam's Orange Grove." If the phrase is of the form "X Y" (made up of subphrases X and Y with X first), and X consists of words not found in the word dictionary (i.e., probably proper names) while Y is in the phrase dictionary, then by almost exact matching, we can assign the code to "X Y" that would be assigned to Y alone. Note that this strategy fails when X is a name which is also a word in the language, e.g., "Stone's Orange Grove."

Almost exact matching is a restricted form of a matching word count strategy. This strategy assigns a code if there is a unique phrase in the coding handbook that has the most words in common with the response. This is a matching strategy that is useful in information retrieval. It 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 the occurrence vectors for the 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.

3. Using the *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 Section 2.4, and **V** is the vector computed from the conditional probabilities of codes given the word. If $\{p(c_i|w)\}$ is the conditional probability of code c_i given word w, the cth component of **V** is $p(c_i|w)**k$, where k is around 0.05. Raising p to this power turns **H**$*$**V** into a filter which adds weights close to **H** to all codes with $p(c|w)$ bounded away from zero. It also adds weights close to zero for codes with very small conditional probabilities. In other words, **H**$*$**V** defines a fuzzy set of codes that are acceptable when given the word w. The fuzziness filters out codes for which the nonzero probabilities could be the result of hand coding errors. Some hand coding errors are found in the sample that is used to construct the conditional probabilities.

The **H**$*$**V** vectors are added. 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.

For example, using the vectors for "citrus" and "grove" over ten codes (these appeared as example coding vectors above) the heuristic weights of these words are about 10 and 7, using the computation described in the later section on the heuristic weight. Thus the vector for "citrus grove" is

$$\begin{aligned}&\text{vector ("citrus grove")}\\&= 10 * (0.95, 0, 0.05, 0, 0, 0, 0, 0, 0, 0)\\&\quad + 7 * (0.50, 0, 0, 0, 0, 0, 0.50, 0, 0)\\&= (13, 0, 0.5, 0, 0, 0, 0, 3.5, 0, 0).\end{aligned}$$

The code agriculture would be assigned, because its component in the phrase's vector was much larger than the other components.

4. When using the *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 for a given 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 to all other components in the vector, then a code is assigned.

For a given phrase, the components in the product vector are proportional to the probabilities for the various codes, assuming that the words are statistically independent. As discussed above, this is a reasonable assumption if the data has already been filtered through a good phrase dictionary.

As an example, the vectors for "citrus" and "grove" are made up of conditional codes given the word corresponding to the vector. If we multiply these probabilities component by component, we get the phrase vector,

$$(0.475, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

As shown by this example, this is a good coding method for finding a code. However, for the phrase "Stone's citrus grove," the product of probabilities would give a zero vector if "stone" had zero probability of occurring in an agriculture response. A word that is almost never used for a particular code can give that code a low score, and thereby cause coding to fail.

By taking logs of the probabilities in the word vectors, we can reformulate this strategy as one of adding word vectors, in conformity with the general algorithm sketched above.

The census experiment described here used data from the 1970 Census of Population and Housing. In this experiment, responses concerned the industry where the respondent or another household member worked. Each record contained two text fields, the *IA* and the *IB* field, and two multiple choice fields.

The *IA* field was the name of the establishment where the individual worked. In production coding of census data, this name sometimes compared with a catalog of establishments with known industry codes. In this experiment, however, no catalog of proper names of establishments was used. The *IA* field was sometimes used as a source of generic words, such as "high school" in "Bethesda High School." Many respondents used a generic name rather than a proper name (e.g., "high school" rather than "Bethesda High School") in this field, as noted by the success of the exact match strategy.

The *IB* field contained a response to the question, "What type of business or industry is this? Describe activity at location where employed." This field provided the main text data for industry coding.

One of the two multiple choice fields, the *IC* field, asked if the establishment where the individual worked was manufacturing, wholesale trade, retail trade, or other, defined on the census schedule to include agriculture, construction, service, government, etc. The other multiple choice field, the *CW* field, asked if the individual worked in federal, state or local government or private industry, and whether the person was self-employed. The responses to these multiple choice fields were treated in the coding algorithms in the same way as text words. A code vector for each multiple choice response to these fields, for example, for an answer like "manufacturing" in the *IC* field, was built in the same manner as for text words (described below) and added into the response vector for the record as a whole just as the vectors were for words and phrases in the *IA* and *IB* fields.

A table below summarizes the results obtained with the strategies described below. Each entry in the table is named by the strategy used (exact match, almost exact match, sum of heuristic weights or product of conditional probabilities), and the text fields (*IA, IB* or both) on which the strategy was used.

For the coding algorithm as a whole, the strategies were tried in the same order as listed in the table. Each strategy was applied to those records not coded by earlier strategies. The middle column of the table, "% coded," lists the percentage of those records that a strategy was tried and where the strategy actually assigned a code. The right column, "% agreement" lists the percentage of records where a strategy assigned the same code as was assigned by coding experts. 250 records were processed in the experiment described by the table below.

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

5. The *overall results* were 96.4 % coded, where 82.2 % agreed with expert-assigned codes (the standard deviation of this agreement percentage is about 2.6 %).

The ordering of these algorithms conforms to the principle of increasing statistical independence of the words in the data record. The first strategy, exact match, assumes that the response is essentially an idiom, a linguistic construct in which particular words appear in a fixed sequence. The words in a phrase that assign a code using the exact match strategy are not at all independent of each other.

In the almost-exact matching strategy, various meaning-preserving transformations are applied to the field. For example, one or more words not found in the database but are located at the beginning of a phrase may be assumed to be proper names in the *IA* field, and therefore deleted. This yields a phrase that exactly matches one in the phrase database. Similarly, a low-content word like "co." may be deleted from the end of a phrase, and suffixes may be stripped from the remaining words. For responses to be coded by inexact matching, the words in a phrase must be somewhat dependent.

Finally, in the sum of heuristic weights and the product of conditional probabilities strategies, the words are considered independent. In particular, the product-of-probabilities procedure assigns each code a score proportional to its probability under the assumption that all the words occurred independently, and that only one code can be assigned per data record.

1.4. *Related work*

Computerized coding of NL industry and occupation data has been studied at the U.S. Bureau of the Census since the early 1970s. The operational background and motivation for this work is described in Biggs (1974). The paper by Appel and Hellerman (1983) describes a variety of algorithms tried at the Bureau of the Census from the early 1970s to 1983. Hellerman (1982) presents the most successful results obtained during this period. Parallel work in coding NL data at Statistics Sweden, which includes highly successful strategies for building and refining coding dictionaries, is described in the papers by Lyberg. (See Lyberg (1981) and Lyberg and Andersson (1983)).

Janas (1977) describes part-of-speech recognition without an extensive dictionary, which could be used to improve the linguistic processing in computerized coding algorithms.

Information retrieval techniques related to those presented here are discussed in Salton (1975). Fillmore (1975) presents a discussion of case grammars. Various applications of statistics to linguistic problems are discussed in the papers Knaus (1981), Moskovitch (1978), and Rieger (1978).

General references related to the topics in this paper include statistics texts such as Freund and Walpole (1980) and Mendenhall

(1968), as well as more specialized works on numerical taxonomy, such as Sneath and Sokal (1973). Rustin (1973) and Winograd (1983) provide surveys of computational linguistics.

2. Semantic Representations for Coding

One of the major problems in building a computerized coding system is finding a representation of the meaning (i.e., a *semantic representation*) of a linguistic construct adequate for assigning the right code. Ideally a semantic representation should be free from particular choices of vocabulary and grammar that a particular respondents uses to express his/her 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 into the following steps.

i. Map the natural language response into its semantic representation.
ii. Assign a code using the semantic representation.

2.1. Construction by computer is a requirement

Automatic coding compares the information contained in an NL response with a database containing information on the subject matter of the survey. This database is called the *knowledge base* of the system in artificial intelligence terminology. Many semantic representations of NL information have been proposed in both artificial intelligence and linguistics. For computerized coding, a representation of knowledge must be chosen that makes the knowledge base constructible by a computer. This is necessary because the subject area covered by many surveys is large (e.g., all economic activities for the industry data used in the census experiment). In addition, the range of linguistic expressions over different respondents (e.g., in 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 suitable when using these extensive knowledge bases.

2.2. Vectors over codes

In our experiments with computerized coding, the semantic representation 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 c_ith component represents the tendency of the construct to represent a response which should be coded to c_i. For example, a phrase which always is coded to a particular code c_0 might have a c_0 component of 1 and all other components zero. 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.

2.3. Building the database

The semantic representation vectors, called *semantic vectors,* or simply *vectors* in the following, can be built from a hand-coded sample *hand* of data, i.e., a sample in which each record has been assigned a code by (human) expert coders (the code so assigned to a record will be called its *hand code*). Let L be a linguistic construct that a vector will be constructed for. L must have the property that a computer can reliably recognize it, although 100 % reliability is not necessary. Examples of such linguistic constructs are words, word roots, phrases and syntax-independent meaning representations for short sentences.

For each construct L, a *count vector* is constructed from the sample *hand* of hand-coded responses. This count vector is a vector over C in which the c_i component is the number of times an occurrence of L was observed in responses in *hand*. These were hand-coded to c_i. These vectors are built by processing *hand* with a computer program that recognizes the linguistic constructs which occur in each record. The computer program then increments the count for the hand code of the record 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(c_i): =count(c_i)
/[sum over c_i count(c_i)],

for assigning code c_i when linguistic construct L occurs. This probability, which we write $p(c_i|L)$, is used extensively in our attempts at automatic code assignment.

Besides the hand-coded data, other sources of information on coding are the coding handbooks used by human coders. These handbooks consist of NL phrases and their associated codes. For example, the coding handbook used by the Census Bureau to assign industry codes contains about 15 000 NL phrases. This information can be recast into 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 zero.

2.4. *Weighting of vectors*

The count and conditional probability vectors defined in the previous section have a direction that shows the propensity of the underlying construct to result in a particular code. In this section we discuss a technique for varying the length of these vectors according to their usefulness in coding.

One of the ways 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 that 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 that have little usefulness in coding, such as the word "company."

2.4.1. Entropy

One method of weighting vectors, used in the implemented vector coding algorithms, was the heuristic weight. The heuristic weight of a word, in turn, is computed from the *entropy* of the coding vector of the word. Given the probabilities p_i for code i, the entropy is:

E=sum p_i*ln(p_i) over all p_i.

Entropy is a measure of the uniformity of a probability distribution. It is zero when only one code has a nonzero probability. For distributions where more than one code has a nonzero probability, the entropy is negative, and has it greatest magnitude when the distribution is uniform.

2.4.2. Uniformized probabilities

The first step in weighting coding vectors for words is to construct coding vectors that show not only the relative frequency of different codes when the word is present, but also show how frequent the word is when a response with a given code occurs. Coding vectors composed of *uniformized probabilities,* defined in this section, achieve this goal.

For most surveys, the responses do not occur in equal proportions. Some codes are more frequent than others. For this reason, the distribution of codes for all responses is non-uniform. Now consider a word W that occurs in a certain constant fraction f of the responses that are assigned a code. W provides no information for coding, and occurs uniformly with respect to the codes, although the distribution of W, which is the same as that of the codes over the whole survey, is not uniform. On the other hand, a word with a uniform distribution would occur in different proportions of records assigned to codes that occurred in different proportions in the survey as a whole. Such a word would be an indicator of the relatively rare codes in the survey.

The entropy computation described above is modified to use *uniformized* probabilities $p'(c_i)$ of a code c_i instead of the probability of the code, $p(c_i)$. The uniformized probability indicates the probability of a word indicating a particular code under the assumption that all codes are equally likely. The uniformized probability measures the relative frequency of the word in data assigned to various codes. For a word W, the uniformized probability $p'(c_i)$ of code c_i is defined to be:

i. Proportional to the conditional probability of W given c_i, $p(W|c_i)$.
ii. Scaled such that the sum of the p' over all c_i is 1.

The uniformized probabilities can be computed in the following way:

$S=\text{sum}(p(W|c_i))$, for all c_i, and for all c_i, $p'(c_i):=p(c_i)/S$.

To illustrate the effect of uniformized probabilities, the word "company" (usually abbreviated "co.") is common in descriptions of establishments assigned to almost all codes. Suppose, that "co." occurred in half the descriptions, regardless of the industry code assigned to the description. The probability distribution of "co." is the same as the probability distribution of codes in the entire dataset, and is highly non-uniform. Schools, for example, employ many more people than farm machinery wholesalers. Nevertheless, the uniformized probability distribution is uniform.

"Office," on the other hand occurs with moderate probability in descriptions for a number of industries (e.g., farm cooperative office, agricultural services; telephone co. office, telecommunications). For a few industries, such as office machines manufacturing, however, "office" is highly likely to occur in descriptions of the industry. These industries employ fewer people than other industries where the word occurs with lower probability in descriptions. Thus the probability distribution of a word like "office" appears relatively uniform. On the other hand, the uniformized probability distribution has a relatively large value at office machine mfg. The distribution of uniformized probabilities have large values where "office" is used to describe the industry. This distribution is based on the entropy of the distribution of codes for a given component. In turn, the entropy measures the non-uniformity in the coding experiments described here. The heuristic weight was computed for the linguistic component "word roots," but the computation works for all vectors over C, regardless of what the vectors represent.

2.4.3. The heuristic weight

The final step in the heuristic weight computation is to transform E' so that constructs that are useful in coding have a large positive weight. And reciprocally, those which are useless have a weight near zero. This means that we want a transformation which maps the uniformized entropies E' near zero 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 zero. 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' that is modified slightly to ensure that E" is nonzero. In the case where E' is zero, a small random error is added to the distribution of uniformized probabilities. The added error is an estimate of the probability that c would be encountered in a response hand-coded in a code different from C. This addition of a random error is justified by the nature of the data. Sometimes words appear to be proper names if used in unusual contexts, by persons with limited English, or through transcription errors. For these reasons, a very large sample will contain few, if any, distributions with zero entropy.

The heuristic weight computed by this method appears to agree with our intuitive idea of how well a word lends itself to coding. Where the coding is over a set of about 250 industry classes (defined by the Census Bureau), some heuristic weights derived 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	12.00
medical	4.10	beer	12.30
iron	4.35	airline	58.60
farm	4.60	turbine	60.00

The heuristic weight can be viewed as a generalization of the inverse document frequency weight used in information retrieval. In the information retrieval case, documents can be assigned into just two categories, relevant and irrelevant. In the case of just two categories, the inverse document frequency and heuristic weight are approximately proportional for the case where one of the categories (e.g., the relevant documents) has a very low probability. Both weights are approximately inversely proportional to the probability. This follows for the heuristic weight, as follows. For small entropy, the heuristic weight is approximately inversely proportional to the entropy (assuming the noise in the denominator is kept *very* small), and the entropy can be approximated for small probabilities by a straight line secant.

2.4.4. Other methods of weighting vectors

Alternatively, one might weight each vector by a correlation coefficient computed over a set of data that codes have been assigned to by hand. More particularly, for each response r and code c let $h(r,c)=1$, if and only if the hand-assigned code of r is c. If the hand-assigned code is not c then $h(r,c)=0$. Let $v(r,c)=$the cth component of v, i.e., the probability of code c given the construct represented by c. Thus 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

$$r=\frac{\text{SQRT}(\,\#C * \underset{\{\text{over codes } c\}}{\text{SUM}}(v_c^2)-1\,)}{\text{SQRT}(\,\#C-1)}$$

This correlation coefficient is zero for the uniform distribution, and one for a vector with just one nonzero component. For vectors between the uniform and single-valued extreme, the correlation coefficient is between zero and one.

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

3. Comparison and Refinement of Coding Methods

In a previous section of this paper, the vector representation of coding semantics was defined, and various coding methods (e.g., sum of heuristic weights) were described. In this section the relation between the different methods are discussed and some refinements are suggested.

3.1. *Relative coding effectiveness*

3.1.1. Phrase matching

There is a saying in the advertising industry that a smart dime never beats a stupid dollar. The same applies to coding. Nothing beats phrase matching where the *entire response*, at least up to trivial variations, is matched to an entry in a coding lexicon. This method is fast and reliable. It was not used to maximum effectiveness in the industry coding experiments discussed here. The phrase dictionary was built from old data and a coding clerk's handbook. Both these sources lagged behind current typical English descriptions of places of work in some respects. For example, the coding handbook used in the experiment contained "tourist cottage" but not "computer store." As a result, the phrase dictionary used for industry coding in these experiments was somewhat out of date, and the phrase dictionary was not updated with new phrases found in the data.

No new phrases from the data were added to the coding dictionary. Nevertheless the data contained phrases that were not in the dictionary but these phrases proved to be very reliable indicators of a particular code. Some of these phrases described new economic activities (e.g., "computer store," "genetic engineering") while others represented new idioms in the language (e.g., "day care" for babysitting). Since phrase matching is such a reliable coding method, the best results in a coding system would be obtained by using a dictionary containing these new phrases. Because both the economy and language are constantly changing, the only way to maintain a current phrase dictionary is to add the new phrases to the dictionary from the data, as the data is coded.

One method of identifying new phrases is to keep a special file of the complete text of responses that are not phrase-coded. One sorts this file and compiles a list of common phrases not in the coding handbook. This list is then presented to experts who can identify the phrases and their associated codes that should be added to the lexicon.

If phrase matching fails to assign a code in the first iteration, the statistical independence of the uncoded words increases. Independence is an important assumption in product scoring and error estimation for linear scoring.

3.1.2. Addition versus multiplication

In the experiment described in Section 1.3, both addition and multiplication were used to combine vectors for words into vectors for phrases. Weighted sums generally worked well for identifying possible codes, but were quite error prone. The weighted sums were easily influenced by a single word that was strongly associated with a particular code. On the other hand, the product of conditional probabilities method avoided the errors encountered in the weighted sum method. But sometimes the conditional probabilities failed to identify codes that should be assigned because the ratio 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. A suggested code would be assigned only if the product score was sufficiently high relative to the product scores of other codes. This would eliminate weighted sum assignments that were based on a part of the response that 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.

3.2. *Linguistic refinements*

In the experiment of Section 1.3, only phrases and root words (words with plural and other suffixes removed) were used. However, other linguistic features can be reliably recognized by a computer and might improve the performance of the code assignment strategies.

3.2.1. Case grammar

Case grammar (Fillmore (1975)), a widely accepted approach in both theoretical and computational linguistics, provides a way to describe the semantic content of simple sentences. More importantly for computerized coding, this description of simple sentence semantics also describes the semantics of noun phrases and other sentence fragments, which are more common as answers to certain questions than are complete sentences.

As described by case grammar, a simple sentence consists of a verb and a set of arguments or noun phrases that stand in a fixed semantic relation 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 or person that 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 or person that causes the action;
Instrument – that which is used in carrying out the action.

We note that this list of case constituents is typical and suitable for computerized coding. Linguists do, however, differ on exactly how the cases are defined.

Natural languages use word order, word endings, function words and standardized patterns of case occurrence to mark the noun phrases in a sentence. For example, in English the object in a simple sentence appears after the verb without a preceding preposition.

3.2.2. 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 the fragments answer. As is true of most linguistic data, respondents choose a grammatical form that eliminates redundant information. The respondent structures his/her sentences so that information known to the researcher precedes new information. (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 work site. 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. Four words or less were typical in the industry data of the experiment. Computer programs that 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 is either the head noun, modifying word, or not applicable (for verbs).

Using case grammar functions to mark word occurrences, we can build and use conditional probability vectors for word uses in the same way that vectors were built and used for root words. For many words, however, informal inspection of the data suggests that such vectors for the same word but used in different ways would vary considerably. When coding industry data, "farm" as a head noun in a free-standing noun phrase answering "Where do you work" is very heavily associated with the agriculture code. "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.

3.2.3. Features based on word uses

The basic principle of the coding algorithm (presented in the "overview" section) is that a large constituent determines a code and assigns it 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 word uses and an associated code. This type of 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 structure of responses for responses that can be successfully matched against the dictionary.

Another refinement based on word uses is to separate the head noun from its modifiers. 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 $\mathbf{V1} <= \mathbf{V2}$ if every code that is plausible given $\mathbf{V1}$ is plausible given $\mathbf{V2}$. One might then distinguish between modifying words that precede a more inclusive word and those that do not. The motivation for this distinction is found in phrases like "grocery store," in which "grocery" functions like a head noun.

4. Assigning Codes to Survey Responses

If $\mathbf{V}$ is a vector which represents the response r using a set of linguistic constructs, and c is a code, then the cth component of $\mathbf{V}$ is a real number. We call this real number the *score* of c in $\mathbf{V}$ for r using s. The vector $\mathbf{V}$ 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 $\mathbf{V}$ if this best code has a score sufficiently better than the other scores. In this section we consider methods for deciding whether the best code is sufficiently better than the rest.

In deciding whether a coding algorithm has assigned the right code, our only criterion is agreement with a hand-assigned code. This is a correct criterion only when the hand-assigned code is correct. In research on computerized coding, it is important to have a sample of hand-coded responses that are correctly coded. 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. For some responses, there is indeed more than one acceptable code. To assess the accuracy of computerized coding, one must look at cases where the automatic code disagrees with the hand code.

One must determine which code is correct or better, or whether both are acceptable. Preferably, this evaluation is done by experts who are blind to which code is automatically assigned. This eliminates 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 shorthand for "a code not equal to the hand-assigned code."

4.1. *When to assign a code*

4.1.1. Different kinds of scoring errors

One problem in computerized coding is deciding when the score of the best code is sufficiently high to justify assigning a code at all. In general it is observed that as the spread between the scores for the best code and next-best code. As this spread increases, the probability of the best code agreeing with the hand code increases. There are, however, a few wrong codes with high scores. There are several types of disagreements with the hand codes. In some cases the hand codes are in error. Another source of code disagreement is *statistical scoring error,* i.e., 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 sample score of the true best code is not the highest score, given the sample probabilities.

In the early experiments performed while the author was at the U.S. Bureau of the Census, 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. Nevertheless, some statistical and computational techniques allow one to get a record-dependent estimate of coding errors that are related to statistical errors in the best and next-best scores. We can use these estimates to control the level of errors due to statistical scoring errors. This is done by assigning a code only when the probability of an error due to a statistical scoring error is below some preset level.

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 a computerized 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 two close contenders, the pairwise error estimates can be used to get an error estimate for one of a small set of next-best codes. In the case of a large set of such next-best codes, coding is obviously very risky.

4.1.2. Estimating errors as a linear sum of random variables

In the case where the scoring vector $\mathbf{V}$ is a weighted sum $\mathbf{V}=\mathrm{sum}(a_i*\mathbf{V}_i)$ of the vectors of conditional probabilities of codes, given words, phrases or other linguistic structures, a particular code is highest-scoring only when the weighted sum of its conditional probabilities is greater than zero. In particular, let b,n be the best and next-best codes, and v_{ib}, v_{in} the conditional probabilities of b and n in $\mathbf{V}_i$.

When $\mathrm{sum}(a_i*v_{ib})-\mathrm{sum}(a_i*v_{in})$ is positive and bounded away from zero, b is assigned. This expression will be called the *score difference* between the best and next best codes. It measures the amount by which the best code is better than the next best, based on a coding algorithm using the weighted sum of vectors, $\mathbf{V}=\mathrm{sum}(a_i*\mathbf{V}_i)$.

When the probabilities in the above formula are not close to 1, the a_i coefficients are stable despite changes in the probabilities $\{v_{ib}, v_{in}\}$. This is true if the changes are such that the probabilities are bounded away from 1. Therefore we can approximate the statistical coding error in the score difference by assuming that the *a*'s are constants and looking at the expression on the left as a linear sum of random variables $\{v_{ib}, v_{in}\}$. 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 zero. The variance of the left side of the inequality is then

$$\text{SUM}(\,a_i^2 * (\text{var}(v_{ib}) + \text{var}(v_{in})\,),\ \{\text{over } i\}$$

where "var" is an abbreviation for variance. The variances of the $\{v_{ib}, v_{in}\}$ 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. In the response "auto repair," both words have a heuristic weight of 4 computed from a sample of 1 000 occurrences, and the probabilities of various codes are given by the following.

Table of *P*(code|word)

code	auto	repair
auto mfg.	0.3	0
auto service	0.3	0.3
electrical repair	0	0.15

Then the best code is "auto service" when the following is true:

$$\begin{aligned}&wt(\text{auto})*p(\text{auto service}|\text{auto})\\&\quad+wt(\text{repair})*p(\text{auto service}|\text{repair})\\&\quad-wt(\text{repair})*p(\text{electrical repair}|\text{repair})\\&\quad>0\end{aligned}$$

The variance of the value of that expression is:

$$\begin{aligned}&wt(\text{auto})**2*\text{var}(\text{auto service}|\text{auto})\\&\quad+wt(\text{repair})**2*\text{var}(\text{auto service}|\text{repair})\\&\quad+wt(\text{repair})**2\\&\qquad*\text{var}(\text{electrical repair}|\text{repair})\\&\quad=3*16*(2*2.1*(10**-4) + 1.3*(10**-4))\\&\quad=2.64*10**-2.\end{aligned}$$

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

On the other hand 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 standard deviation is increased by a factor of sqrt(40), and is 1.03. The value zero is about 1.07 standard deviations from the observed value of the expression, and the statistical coding error is about 14 %.

While this example is hypothetical, the numbers are typical of the sample sizes and probabilities that arise when constructing conditional probability vectors from a large sample of actual responses. This 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 that is stored in the computer and estimates the statistical coding error.

4.1.3. Error estimates in the unstable region

When the distribution of probabilities in one of the summand vectors 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.

The following is an alternative method for estimating the statistical error in the score difference when one of the probabilities in a summand vector is near 1.

i. 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.
ii. If p is sufficiently small, ignore the possibility of a change in the large probability. Otherwise compute the statistical error for two different cases. The first concerns the probability-weighted average in the case where the large probability remains unchanged. The second concerns 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**. This has the effect of reducing the highest probability. In the two subcases, the statistical error is computed with **V** assumed constant, but vectors other than **V** that have not been assumed constant at some previous subdivision of the computation are allowed to vary.

5. Computer Implementation

At the time these experiments were conducted, hardware was considerably more expensive than it is now and the hardware options were confined to large computers. Processing time was expensive, and the processing time per record was about 1 second for the algorithm in the experiment. These factors prevented more elaborate coding experiments. In addition, development and experiment with the program was hindered by the long waits and temporary shut downs associated with using a heavily loaded time-shared computer.

5.1. Microcomputer-based implementations

5.1.1. Hardware configuration and cost

The microcomputer hardware and software available today (1986) would support economical interactive computerized coding. The resources needed include a microcomputer workstation for each coder, and hard disk storage for the coding dictionary.

The basic workstation costs about 1 500 USD. Each workstation must be able to access the coding dictionary, which can be accomplished with a local area network. The cost of the network includes a network access board for each workstation computer and the cost of the hard disk.

The hard disk requirements can be estimated in the following way. Vectors for words take about 1K bytes apiece, given the 250 industry codes. Therefore about 50 Mb. (megabytes) of storage would be needed for this part of the dictionary. The phrase dictionary would take only about 5 Mb., because phrase vectors have few nonzero components. For a system that can code both industry and occupation, 100 Mb. of hard disk should be more than sufficient. The cost for this storage is about 2 000 USD.

At least ten workstations can be supported by a local area network. The local area networks at American University support about 25 users from a single hard disk with no noticable deterioration in service when used at full capacity. With ten workstations on the network, the per user cost of the hard disk is 200 USD, and the cost of a network access card is about 300 USD. In addition, the per user cost of software and printers adds another 200–300 dollars to the cost per workstation. The total direct cost per workstation for a microcomputer coding system (excluding development of the coding software and staffing and maintenance for the local area network) is about 2 200 USD per workstation.

Experience with local area networks at American University has shown, however, that although they are very reliable (e.g., one network break down per year), a network supervisor is needed to maintain the hardware, assist users, and tend to the software. For a network of ten users, perhaps two hours a day should be allocated for these tasks (this is generous). At a yearly salary of 25 000 USD for an individual with the proper training, 600 USD per workstation should be added to cover the costs of network staffing and maintenance.

A very large part of the cost of a computerized coding system is the development of the coding software. This is difficult to estimate, but a coding algorithm similar to that used in the experiment discussed earlier in the paper could probably be developed in six months to one year.

Even more difficult to estimate is the cost of building the coding dictionary. This cost depends on whether hand-coded data is available in machine readable form and on the amount of programming needed to extract the data from the pertinant records. Specialized programs to build the coding data from these extracted files of hand coded data are also needed. Much of the time and effort devoted to establishing computerized coding at the Bureau of the Census was devoted to constructing coding dictionaries. A good discussion of dictionary construction at Statistics Sweden appears in Lyberg (1981).

5.1.2. Advantages of interactive coding

There are several advantages to interactive coding. In interactive coding, the computer codes those records that it can unambiguously assign a code. For more ambiguous cases, the computer assigns a code through a dialog with the user of the system. If a coding system is used during data entry, the clerk can use his/her own judgement to determine the order that the different parts of the response are entered in. Highly descriptive parts should be entered before meaningless parts, like proper names. If coding occurs before the whole record is entered, the natural language response never has to be keyed in. Discretionary entry also prevents the computer from being distracted by proper names which it, in its limited ability to understand language, does not recognize as proper names. Additionally, the computer can catch many probable spelling errors at a time when the entry clerk can correct them. In the experimental data, some records were not coded correctly by the computer because of such data input errors.

While some responses could be assigned codes without any intervention by the user in an interactive system (e.g., responses that matched an entry in the phrase dictionary), others could not be coded by computer alone. Two or more codes could have nearly identical scores. In such cases, the high-scoring codes and a short description of each could be displayed, and the coding clerk could select one of them. Because the computer-generated list of likely codes reduces the chance that the coding clerk will fail to consider some code, computer-assisted coding is likely to contain fewer errors than unassisted hand-coding. On the other hand, the clerk can make a better judgement among codes with close scores than the computer can. The clerk has a vastly superior understanding of natural language and superior reasoning abilities. Therefore, an interactive system can be used to assign codes to responses that are too ambiguous for unassisted coding.

Another possibility in a microcomputer environment is the inclusion of expertsystem rules in the coding algorithm. This is possible because a micro-based algorithm can use more processing time per record. The hardware itself is inexpensive and as long as the microcomputer keeps up with the user, the coding algorithm is fast enough. Several microcomputer implementations of the artificial intelligence language Prolog exist for microcomputers. These have the additional feature of interfacing well with languages (e.g., C) that are efficient for numerical computations. A Prolog rule would allow phrase templates that would in turn expand the power of the phrase dictionary. For example, the fact that manufacturing anything out of plastic is considered part of the plastic fabrication industry could be expressed by the following Prolog rule.

```
code( [plastic | Item], 'plastic fabrication') :-
    code( Item, 'general manufacturing').
```

6. Conclusion

For experiments using the US census's natural language question on industry, the agreement between codes assigned by experts and codes assigned by the computer was 80 %. These results can probably be improved by using the techniques discussed above for deciding when to assign a code, doing more sophisticated linguistic analysis, providing clerk-computer interaction, and incorporating expertsystem rules.

7. References

Appel, M. and Hellerman, E. (1983): Census Bureau Experiments with Automated Industry and Occupation Coding. American Statistical Association, Proceedings of the Section on Survey Research Methods, pp. 32–40.

Biggs, J. (1974): 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.

Fillmore, C.J. (1975): Principles of Case Grammar: The Structure of Language and Meaning. Sanseido Publishing Co., Tokyo.

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

Hellerman, E. (1982): Overview of the Hellerman I&O Coding System. Draft memo, Bureau of the Census, Washington, D.C.

Janas, J.M. (1977): Automatic Recognition of the Parts of Speech for English Texts. Inf. Proc. Man. 13, pp. 205–213.

Knaus, R. (1981): Pattern-Based Semantic Decision Making. In Rieger, B. (ed.): Empirical Semantics, Brockmeyer, Bochum, W. Germany.

Lyberg, L. (1981): Control of the Coding Operation in Statistical Investigations. Urval no. 13, Statistics Sweden, Stockholm.

Lyberg, L. and Andersson, R. (1983): Automated Coding at Statistics Sweden. American Statistical Association, Proceedings of the Section on Survey Research Methods, pp. 41–50.

Mendenhall, W. (1968): The Design and Analysis of Experiments. Duxbury Press, (Division of Wadsworth Publishing), Belmont, CA.

Moskovich, W. (1978): Distributive-Statistical Techniques on Computational Linguistics: Problems and Perspectives. 7th International Conference on Computational Linguistics, Bergen.

Rieger, B. (1978): Feasible Fuzzy Semantics. 7th International Conference on Computational Linguistics, Bergen.

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

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

Sneath, P. and Sokal, R. (1973): Numerical Taxonomy. W.H. Freeman, San Fransisco.

Winograd, T. (1983): Language as a Cognitive Process. Addison Wesley, Reading, Mass.

Received January 1985
Revised May 1986