Consiglio Nazionale delle Ricerche

Internal Report

Machines that learn how to code open-ended survey data. Part II: experiments on real respondent data

Andrea Esuli, Tiziano Fagni, Fabrizio Sebastiani

2009-TR-012

ISTI

ISTITUTO DI
SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE
"A. FAEDO"

Pisa

Machines that Learn how to Code Open-Ended Survey Data. Part II: Experiments on Real Respondent Data

Andrea Esuli*, Tiziano Fagni† and Fabrizio Sebastiani‡
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi 1 – 56124 Pisa, Italy

Abstract. In a previous paper we have described a novel approach to coding verbatim responses to open-ended questions that relies on machine learning, and we have introduced VCS$^{\rm TM}$, a working computerized system that we have designed and implemented according to this approach. In the present paper we present the results of a number of experiments we have run on several datasets of respondent data in order to assess the accuracy and the efficiency of VCS$^{\rm TM}$.

Keywords: Survey coding, open-ended questions, open-ended responses, automatic coding, machine learning, accuracy, efficiency.

1. Introduction

In a previous paper (Esuli et al., 2009) (hereafter: "Part I") we have described a novel approach to coding verbatim responses to open-ended questions that relies on machine learning: a general algorithm learns the characteristics that a given verbatim should have in order to be assigned a given code from a set of manually coded verbatims, some of which have been assigned the code while the rest have not. Part I describes the basic philosophy underlying the machine learning approach to verbatim coding and the overall mode of operation of VCS$^{\rm TM}$, a working system that we have designed and implemented according to this approach, and that is now commercially available.

The present paper is a companion to Part I (whose knowledge is assumed on the part of the reader), in which we present the results of a number of experiments we have run on several datasets of respondent data in order to assess the accuracy and the efficiency of VCS$^{\rm TM}$. The paper is organized as follows. Section 2 describes the purpose of these experiments in general, and the criteria we use to assess VCS$^{\rm TM}$. Section 3 looks at accuracy issues, both at the individual level (typically useful

* E-mail: andrea.esuli@isti.cnr.it
† E-mail: tiziano.fagni@isti.cnr.it
‡ E-mail: fabrizio.sebastiani@isti.cnr.it

for applications in customer satisfaction) and at the aggregate level (typically useful for applications in market research); these experiments also allow us to appreciate the characteristics that make a survey more or less amenable to automated coding by means of our system. Section 4 focuses instead on efficiency issues, illustrating how much computer time, in each of the preceding experiments, was needed for generating the binary coders and applying them to uncoded verbatims requires. Section 5 adds further insight into accuracy issues by giving, in question and answer format, a few clarifications on the workings of VCS™. Concluding remarks are made in Section 6.

2. Testing VCS™ on real-world sets of verbatim data

In experimental computer science, in order to assess the quality of a computerized tool it is customary to perform one or more laboratory experiments on predefined benchmarks. In this section we present the results of testing VCS™ on a number of benchmark datasets of real respondent data. From now on, by *dataset* we will mean a set of manually coded verbatims returned by respondents to a given question, plus the codeframe that the human coders have used for coding them.

Different qualities of a system may be tested in an experiment: in our experiments we test three aspects, namely,

- *accuracy*, which measures how frequently the coding decisions of VCS™ agree with the coding decisions of the human coder who has originally coded the data;
- *training efficiency*, which measures how fast is VCS™ in training the binary coders for a given codeframe when using a given training set;
- *coding efficiency*, which measures how fast are the automatically generated binary coders in coding yet uncoded verbatims.

VCS™ accuracy tests are the subject of Section 3, while VCS™ training and coding efficiency tests are the subject of Section 4.

Usually, an experiment involving learning machines is run according to the *train-and-test* methodology, which consists in splitting the dataset in two non-overlapping parts:

- the *training set*, which is used for training the binary coders;
- the *test set*, which is used for testing the binary coders. This is done by feeding the binary coders with the verbatims in the test

set, each stripped from the codes that have been attributed to it by the human coder, asking the binary coders to code them, and comparing the coding decisions of the binary coders with the coding decisions the human coder had taken on the same verbatims.

It is important that there are no verbatims in common between the training set and the test set since, quite obviously, it would be unrealistically easy for a binary coder to code the verbatims it has been trained on.

We have carried out all our experiments according to a popular, more refined variant of the train-and-test technique, called *10-fold cross-validation*. This consists in splitting the dataset in 10 equally-sized sets of verbatims, and running 10 train-and-test experiments, each of which consists in using one set as the test set and the union of the other 9 as the training set. The performance of the binary coders is then computed as the average performance that the binary coders have displayed across the 10 different experiments. 10-fold cross-validation provides more reliable results than a single train-and-test experiment, since all the verbatims in the dataset are, sooner or later, being tested upon.

Table I reports the results of our experiments on 13 different datasets (the meaning of the various column headers will be clarified soon). The first 10 datasets (LL-A to LL-L) consist of verbatims from market research surveys and were provided by Language Logic LLC. The LL-B, LL-D, and LL-F to LL-L datasets are from a large consumer packaged-good study, with both open-ended and brand-list questions. The LL-A, LL-C, and LL-E datasets are instead from one wave of a tracking survey that Language Logic LLC codes 12 times a year, which consists of semi-open brand questions. The next 2 datasets (EGG-A and EGG-B) consist of verbatims from customer satisfaction surveys and were provided by Egg plc; for both datasets, which were collected in the context of two different surveys, respondents were answering the question "Have we done anything recently that has especially disappointed you?". The last dataset (ANES L/D) consists of verbatims from a political survey run in 1992 and were obtained from the American National Election Studies (ANES) committee. Two sets of verbatims were used: the first were returned in answer to the question "Is there anything in particular about Mr. Clinton that might make you want to vote for him? If so, what is that?" while the second were returned in answer to the question "Is there anything in particular about Mr. Clinton that might make you want to vote against him? What is that?". Our coding task consisted in guessing whether the verbatim belongs to the former or to the latter set.

Table I. Results of experiments on ten market research datasets (LL-A to LL-L), two customer satisfaction datasets (Egg-A and Egg-B), and one social science dataset (ANES L/D). The columns represent the name of the dataset (DS), the number of verbatims in the dataset ($\#V$), the number of codes in the codeframe ($\#C$), the average number of positive training verbatims per code (AVC), the average number of (non-unique) words in a verbatim (i.e., the "average verbatim length" – AVL), the accuracy at the individual level (F_1^μ), the accuracy at the aggregate level (PD_M and PD_A), training time (Tt) and coding time (Ct).

DS	$\#V$	$\#C$	AVC	AVL	F_1^μ	PD_A	PD_M	Tt	Ct
LL-A	201	18	21.00	1.35	.92	0.8%	4.0%	0.9	0.1
LL-B	501	34	26.65	1.65	.90	0.6%	4.8%	10.0	0.4
LL-C	201	20	10.05	1.61	.89	0.7%	7.4%	0.9	0.1
LL-D	501	27	45.30	3.32	.85	0.8%	5.6%	24.7	1.0
LL-E	201	39	8.41	2.57	.84	0.4%	2.5%	4.7	0.2
LL-F	501	57	37.58	6.99	.82	0.7%	4.8%	61.0	2.5
LL-G	501	104	21.30	6.25	.80	0.5%	5.2%	123.3	5.0
LL-H	501	86	30.08	7.87	.79	0.7%	5.7%	136.4	5.5
LL-I	501	69	33.16	7.70	.78	0.8%	5.2%	102.3	4.1
LL-L	501	65	29.40	5.58	.75	1.0%	9.6%	84.9	3.4
Egg-A	1000	16	123.37	27.37	.62	1.5%	3.1%	173.0	7.0
Egg-B	924	21	67.19	26.47	.55	1.6%	3.6%	175.6	7.1
ANES L/D	2665	1	1396.00	30.83	.86	4.6%	4.6%	47.0	1.9

3. Testing for accuracy

Accuracy may be measured at two different levels:

- at the *individual level*: a hypothetical perfect coding system is the one which, given a code C, assigns C to all and only the verbatims to which the human coder would assign C;
- at the *aggregate level*: here, a hypothetical perfect coding system is the one which, given a code C, assigns C to $x\%$ of the verbatims exactly when the human coder would assign C to $x\%$ of the verbatims.

The former notion of accuracy is especially interesting for measuring the suitability of the system to customer satisfaction applications, where a user is interested in correctly classifying each individual respondent, so as to be able to cater for her specific needs (e.g., ringing her up if she is particularly unsatisfied). The latter notion of accuracy, on the contrary, is especially interesting for opinion surveys and market research, where a user is typically interested in the *percentage* of respondents that fit under a given class, and may be less interested in knowing what each individual respondent thinks.

Note that accuracy at the individual level implies accuracy at the aggregate level, but not vice versa! A system perfectly accurate at the aggregate level may be inaccurate at the individual level; this may happen when the coding errors a system has made are split into two equal-sized subsets of false positives and false negatives, respectively. Therefore, measures of accuracy at the aggregate level are somehow more lenient that measures of accuracy at the individual level (but they are certainly no less important).

3.1. Testing for accuracy at the individual level

3.1.1. *Mathematical measures of individual accuracy*

Accuracy testing requires a mathematical *measure of accuracy* to be defined and agreed upon. The one we adopt for accuracy at the individual level (or *individual accuracy*, for short), called F_1, consists of the *harmonic mean*[1] of two other measures, precision and recall:

- For a given code C, *precision* (denoted π) measures the ability of the system to avoid "overcoding", i.e., attributing C when it should not be attributed. In other words, precision measures the ability of the system to avoid "false positives" (aka "errors of commission", or "Type I errors") for code C.

- For a given code C, *recall* (denoted ρ) measures the ability of the system to avoid "undercoding", i.e, failing to attribute C when it should instead be attributed. In other words, recall measures the ability of the system to avoid "false negatives" (aka "errors of omission", or "Type II errors") for code C.

In a given experiment, precision and recall for a given code C are computed from a four-cell *contingency table* (see Table II), whose cells contain the numbers of true positives, false positives, false negatives, and true negatives, respectively, resulting from the experiment. Precision, recall and F_1 are defined as

$$\pi = \frac{TP}{TP+FP} \qquad \rho = \frac{TP}{TP+FN}$$

[1] http://en.wikipedia.org/wiki/Harmonic_mean

Table II. The four-cell contingency table for code C resulting from an experiment.

Code		coder says	
C		**YES**	**NO**
system	**YES**	TP	FP
says	**NO**	FN	TN

Table III. Example contingency tables for two hypothetical codes C_1 and C_2 (left) and the corresponding computations for precision, recall, and F_1 (right).

Code		coder says	
C_1		**YES**	**NO**
system	**YES**	15	7
says	**NO**	8	70

$$\begin{aligned} \pi &= \frac{15}{15+7} = \frac{15}{22} = .682 \\ \rho &= \frac{15}{15+8} = \frac{15}{23} = .652 \\ F_1 &= \frac{2 \cdot .682 \cdot .652}{.682 + .652} = \mathbf{.667} \end{aligned}$$

Code		coder says	
C_2		**YES**	**NO**
system	**YES**	22	13
says	**NO**	5	60

$$\begin{aligned} \pi_j &= \frac{22}{22+13} = \frac{22}{35} = .629 \\ \rho_j &= \frac{22}{22+5} = \frac{22}{27} = .815 \\ F_1 &= \frac{2 \cdot .629 \cdot .815}{.629 + .815} = \mathbf{.710} \end{aligned}$$

$$F_1 = \frac{2 \cdot \pi \cdot \rho}{\pi + \rho} = \frac{2 \cdot TP}{(2 \cdot TP) + FP + FN}$$

For instance, assume that our test set contains 100 verbatims and that our codeframe consists of two codes C_1 and C_2; Table III illustrates two possible contingency tables for C_1 and C_2, and the relative computations of precision, recall, and F_1.

Precision, recall and F_1 can also be computed relative to an entire codeframe (in this case they are noted π^{μ}, ρ^{μ}, and F_1^{μ}) by using a "combined" contingency table. Table IV illustrates such a contingency table as resulting from the data of Table III.

There are several reasons why F_1 is a good measure of individual accuracy. First of all, F_1 always equals 0 for the "pervert binary coder" (the one for which $TP = TN = 0$, i.e., no correct coding decision) and F_1 always equals 1 for the "perfect system" (the one for which $FN = FP = 0$, i.e., no wrong coding decision). Second, it partially rewards

Table IV. Combined contingency table for codes C_1 and C_2 as resulting from the data of Table III (left) and the corresponding computations for precision, recall, and F_1 (right).

Codes	C_1 and C_2	coder says YES	coder says NO
system says	**YES**	$15+22$	$7+13$
	NO	$8+5$	$70+60$

$$\pi^{\mu} = \frac{(15+22)}{(15+22)+(7+13)} = \frac{37}{57} = .649$$

$$\rho^{\mu} = \frac{(15+22)}{(15+22)+(8+5)} = \frac{37}{50} = .740$$

$$F_1^{\mu} = \frac{2 \cdot .649 \cdot .740}{.649 + .740} = \mathbf{.692}$$

partial success: i.e., if the true codes of a verbatim are C_1, C_2, C_3, C_4, attributing it C_1, C_2, C_3 is rewarded more than attributing it C_1 only. Third, it is not "easy to game", since it has very low values for "trivial" coding systems: e.g., the "trivial rejector" (the binary coder that never assigns the code) has $F_1 = 0$, while the "trivial acceptor" (the binary coder that always assigns the code) has $F_1 = \frac{TP+FN}{TP+FP+FN+TN}$, which is usually low). Fourth, it is symmetric, i.e., its values do not change if one switches the roles of the human coder and the automatic coder; this means that it can also be used as a measure of agreement between any two coders (human or machine) since it does not require one to specify who among the two is the "gold standard" against which the other needs to be checked. For all these reasons, F_1 is an "industry standard" in the field of text classification (Sebastiani, 2002; Sebastiani, 2006).

3.1.2. *The results*

In order to analyse the results of Table I, let us first concentrate on individual accuracy and F_1. The F_1 values obtained across the 15 experiments vary from $F_1 = 0.92$ (best result, on LL-A) to $F_1 = 0.55$ (worst result, on Egg-B), with an average of $F_1 = 0.77$.

3.1.2.1. *How accurate is VCS^{TM}?* How good are these results? Let us remember that F_1 is a measure of how closely VCS™ can mimic the coding behaviour of the human coder (let's call her K_1) who has originally coded the test data. Since the alternative to computer coding is manual coding, the question "How good are the results of VCS™?" is probably best posed as "How accurate are the results of VCS™ with respect to the results I could obtain from a human coder?". In order to make this comparison, what we can do is checking how closely this latter coder (let's call her K_2) can mimic the coding behaviour of K_1, and compare $F_1(VCS^{TM}, K_1)$ with $F_1(K_2, K_1)$. In other words, an answer to this question can be given only following a careful *intercoder* *agreement* study, in which two human coders K_1 and K_2 are asked to independently code a test set of verbatims after one coder has "learnt how to code" from a training set of verbatims coded by the other coder. Unfortunately, this question is left unanswered for most of the 13 datasets listed in Table I, since for coding most of them only one human coder was involved, and no intercoder agreement studies were performed. The only exceptions are the Egg-A and Egg-B datasets: in (Macer et al., 2007) the accuracy of VCS™ was tested it the context of a thorough intercoder agreement study, which showed that VCS™ obtained F_1 values, with respect to either K_1 or K_2, only marginally inferior to $F_1(K_2, K_1)$[2].

3.1.2.2. *Why does the accuracy of VCS™ vary so widely?* A second question we might ask is: why does the accuracy that VCS™ obtains vary so widely across different datasets? Can I somehow predict where in this range will VCS™ perform on my data? The answer to this latter question is: "Somehow, yes." We have experimentally observed (and machine learning theory confirms) that the F_1 of VCS™ tends (i) to increase with the average number of (positive) training verbatims per code (AVC) provided to the system, and (ii) to decrease with the average length of the training verbatims (AVL). In short, a dataset in which there are many (resp., few) training verbatims per code and whose verbatims tend to be short (resp., long) tends to bring about high (resp., low) F_1.

Note that parameter (i) is within the control of the user, while parameter (ii) is not. That is, if there are too few training examples per code, the user can manually code new verbatims, thus increasing AVC at will; however, if verbatims are long, the user certainly cannot shorten them. So, is VCS™ doomed to inaccuracy when the dataset consists of long verbatims (i.e., AVL is high), as is the case in customer satisfaction datasets? No, it is simply the case that, if AVL has a high value, AVC needs to have a high value too in order to "compensate" for the high AVL value.

A simple explanation to this phenomenon may be offered through an example drawn from college mathematics. Long verbatims usually mean complex, articulated sentences, with many different linguistic expressions (e.g., words, phrases) and phenomena (e.g., syntactic patterns, idioms) showing up in the dataset. If the number of phenomena to "explain" is large, there needs to be a large amount of information

[2] These F_1 results are not reported here since in that paper experiments were run with a standard training-and-test methodology, with a 70/30 split of the data between training and test, and are thus not comparable with the F_1 results of the present paper, obtained with 10-fold cross-validation and with a 90/10 split.

in order to explain them. This information is nothing else than the training examples; this explains why when the data consists of long verbatims, many training examples are needed to make sense of them, i.e., to obtain good accuracy. This is akin to what happens in systems of linear equations: if there are many variables, many equations are needed for finding a solution (if there are many variables and few equations, the system is underconstrained). In our case, the linguistic expressions and phenomena are the variables, and the training verbatims are the equations constraining these variables: long verbatims means many linguistic phenomena, i.e., many variables, and accuracy thus requires many training verbatims, i.e., many equation.

3.2. Testing for accuracy at the aggregate level

We now move to discussing accuracy at the aggregate level (or *aggregate accuracy*, for short). For measuring it we use a mathematical function that we call *percentile discrepancy* (PD, for short), defined as the absolute value of the difference between the true percentage and the predicted percentage of verbatims with code C. In other words, suppose that 42% of the verbatims in the test set have code C, and suppose that VCS$^{\text{TM}}$ applies instead code C to 40.5%, or to 43.5%, of the verbatims in the test set; in both cases we say that VCS$^{\text{TM}}$ has obtained 1.5% PD on code C. A hypothetical ideal binary coder always obtains $PD = 0$.

For all tests in Table I we report both the average and the maximum PD value obtained across all codes (noted PD_A and PD_M, respectively). A look at these results shows that VCS$^{\text{TM}}$ is indeed very, very accurate at the aggregate level: for instance, PD_A results tell us that in 10 studies out of 13 VCS$^{\text{TM}}$ errs, on average across all codes in the codeframe, by 1.0% or less; this is a tolerable margin of error in many (if not most) market research applications, and it is even more tolerable if this is the only price to pay for coding very large amounts of verbatim data with no manual effort.

In order to visualize this, Figure 1 displays an example histogram from one of the datasets of Table I (dataset LL-E); for each code in the codeframe, two thin bars are displayed side by side, the rightmost representing the true percentage of verbatims that have the code, and the other the percentage of verbatims that have the code as predicted by VCS$^{\text{TM}}$; the figure gives a compelling display of the aggregate accuracy of VCS$^{\text{TM}}$.

Why is VCS$^{\text{TM}}$ so good at the aggregate level? The explanation is slightly technical, and comes down to the fact that, in order to obtain results accurate at the individual level, the internal algorithms

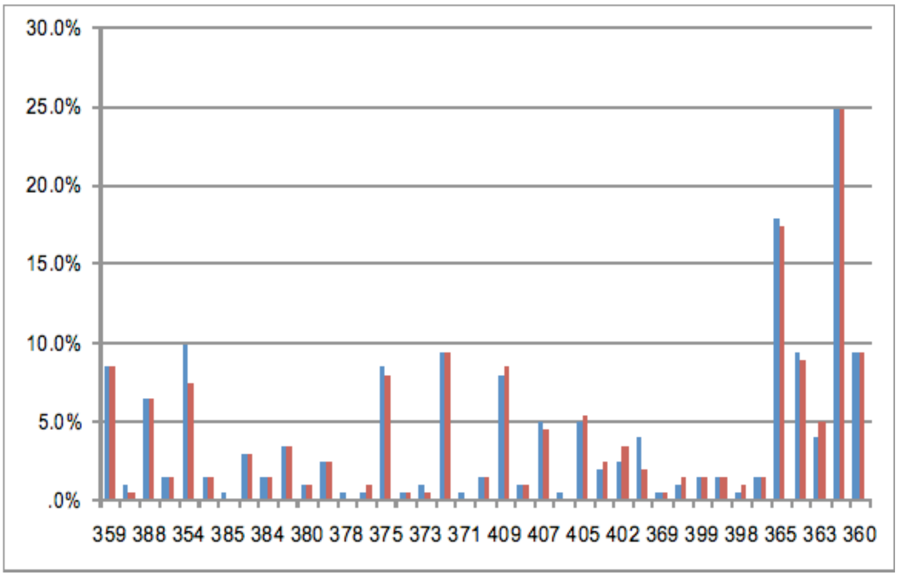

Figure 1. True percentages (first bar) against predicted percentages (second bar) for the LL-E dataset. The highest value of PD (2.5%) is obtained for code 354, where 10% is the correct percentage and 7.5% is the percentage predicted by VCS^{TM}.

of VCS^{TM} attempt to generate binary coders that maximize F_1. Maximizing F_1 means trying to balance false positives and false negatives, since F_1 has the property that, among a set of candidate binary coders for code C that make the same number of mistakes on the same dataset, the one that has the highest F_1 is the one for which the mistakes are equally split into false positives (FP) and false negatives (FN). And when $FP = FN$, then $PD = 0\%$ ...

Of course, similarly to individual accuracy, how good are the PD values obtained by VCS^{TM} would be best assessed with respect to an intercoder agreement study. Interestingly enough, on EGG-A and EGG-B (the only datasets in Table I for which intercoder agreement study were performed, as reported in (Macer et al., 2007)), the PD values that human coders K_1 and K_2 obtained with respect to each other, were similar or sometimes even higher (i.e., worse) than the results that VCS^{TM} obtained with respect to either coder!

The reason is that it is often the case than one coder is *consistently* more liberal than the other in assigning the code (i.e., her mistakes are mostly false positives) while the other is *consistently* more conservative (i.e., her mistakes are mostly false negatives)[3]; both these behaviours bring about high PD, while VCS^{TM} by attempting to maximize F_1, at-

[3] This is an example of so-called *correlated coder error*; see e.g., (Sturgis, 2004).

tempts to strike a balance between liberal and conservative tendencies, thus reducing PD.

4. Testing for efficiency

We now move to discussing efficiency (i.e., computer time requirements) issues. As mentioned at the beginning of Section 2, there are two sides to efficiency in VCS$^{\text{TM}}$:

- *Training-time efficiency*: how fast can the binary coders for a given codeframe be generated from a given set of training verbatims?
- *Coding-time efficiency*: how fast can the binary coders generated for a given codeframe code new, yet uncoded data?

Training and coding running times (in seconds) for all of our 15 experiments are reported in Table I; each result refers to the time required to run one of the 10 experiments involved in the 10-fold cross-validation, involving training from 90% of the verbatims and coding the other 10%.

In order to give an idea of the efficiency of VCS$^{\text{TM}}$ on a more representative example, our tests on the EGG-A and EGG-B datasets translate into the fact that, for a 20-code codeframe, (i) the binary coders can be trained from 1,000 training examples in approximately 2 minutes altogether; and (ii) under the same codeframe, 100,000 verbatims can be coded automatically in approximately 8 minutes. In our tests on the LL-A to LL-L datasets both training and coding were, on average, approximately 7.6 times faster than on EGG-A and EGG-B (due to lower AVL).

In general, the algorithms we employ within VCS$^{\text{TM}}$ are such that training time (resp., coding time) increases proportionally with the number of training verbatims (resp., with the number of verbatims to code), with the number of codes in the codeframe, and with the average length of the training verbatims.

Overall, the results above indicate that VCS$^{\text{TM}}$ is completely up to being employed even on studies of gigantic size (e.g., as when coding huge backlogs of data for retrospective analysis), on which training and coding would be performed literally in a matter of a few hours altogether.

5. Frequently asked questions

QUESTION 1. *Will accuracy keep increasing if I keep adding more and more training data, no matter how much I have already added?*

No, it is typically the case that accuracy will, at a certain point, plateau. In general, the increase in accuracy due to the increase in the number of training examples tends to be brisker at the earlier stages, and slower at the later stages: adding 10 training documents to a training set consisting of 100 documents usually brings about higher benefit than adding 10 training documents to a training set consisting of 1000 documents.

QUESTION 2. *If so, do I then run the risk of making accuracy* decrease *as a result of adding* too many *training examples?*

No, it is practically never the case that there is such a risk. While it is always possible, although rare, that accuracy decreases as a result of adding more training examples (e.g., this might certainly happen in case these newly added examples have been miscoded by the human coder), these are usually local phenomena of small magnitude; in these cases, adding further training examples should bring the situation back to normal.

QUESTION 3. *If I want to increase accuracy for a given code I know I should add more training examples. Are positive and negative examples of this code equally valuable?*

No. While it is true that both positive and negative examples are necessary, positive examples are more informative than negative examples. Just imagine a child being taught by her father what a tiger is. It is certainly more informative for the father to show the child a picture of a tiger and saying to her "This is a tiger!" than showing her a picture of a penguin and saying to her "This is a not a tiger!". So, one should strive to add *positive* examples of the code. Note that, since adding a training verbatim means telling the system, for *each* code in the codeframe, whether the verbatim has the code or not, and since each verbatim typically has only one or few out of the many codes in the codeframe, negative examples abound anyway.

QUESTION 4. *Can I tune the internals of VCS^{TM}, so as to attempt to increase its accuracy?*

No. Had we wanted to provide a coding system that could be tampered with, we would have set up a system based on manually built, manually inspectable, and manually tunable coding rules. The only way a user can improve the coding behaviour of VCS™ is by providing more training data (i.e., adding more training data, or validating more automatically coded data). This is not a limitation of VCS™, it is its beauty!, since no skills are required to operate this software other than standard coding skills. The internals of VCS™ are already optimized based on sophisticated mathematics, and allowing the user to turn knobs would inevitably mean a damage to the accuracy of coding.

Incidentally, the difference between (manually) specifying a rule for recognizing instances of a concept, and simply pointing to examples of the concept, has long been studied by analytic philosophy and cognitive science (see e.g., (Putnam, 1975)), and is known as the difference between the *intensional definitions* of a concept[4] (provided by listing necessary and sufficient conditions that an entity must satisfy for being an instance of the concept) and the *ostensive definition* of a concept[5] (provided by pointing to one or more examples of the concept). Cognitive scientists have long recognized how humans find (unless operating in specialised domains, such as mathematics) ostensive definitions much simpler than intensional definitions: is it easier to provide an intensional definition of “red” (i.e., a rule for recognizing red objects) or to provide an ostensive definition of “red” (i.e., a list of representative red objects)?

6. Concluding remarks

The experimental results obtained by running VCS™ on a variety of real respondent datasets confirm that the approach embodied by it is a viable one, since all these experiments were characterized by good accuracy at the individual level, very good accuracy at the aggregate level, and excellent training and coding speed.

The main challenge for the future will consist in obtaining even higher levels of accuracy, and on even more difficult types of data, such as highly problematic textual data (such as data obtained from OCR, or data from surveys administered via cell-phone text messaging) or audio data (as collected in CATI).

References

Esuli, A., T. Fagni, and F. Sebastiani: 2009, ‘Machines that Learn how to Code Open-Ended Survey Data. Part I: The Basic Approach and a Working System’. *International Journal of Market Research.* Submitted for publication.

Macer, T., M. Pearson, and F. Sebastiani: 2007, ‘Cracking the Code: What customers say, in their own words’. In: *Proceedings of the 50th Annual Conference of the Market Research Society (MRS’07).* Brighton, UK.

[4] See e.g., `http://en.wikipedia.org/wiki/Intensional_definition`

[5] See e.g., `http://en.wikipedia.org/wiki/Ostensive_definition`

Putnam, H. W.: 1975, 'The Meaning of "Meaning"'. In: K. Gunderson (ed.): *Language, Mind, and Knowledge*, Vol. 7. Minneapolis, US: University of Minnesota Press, pp. 131–193.

Sebastiani, F.: 2002, 'Machine learning in automated text categorization'. *ACM Computing Surveys* **34**(1), 1–47.

Sebastiani, F.: 2006, 'Classification of text, automatic'. In: K. Brown (ed.): *The Encyclopedia of Language and Linguistics*, Vol. 2. Amsterdam, NL: Elsevier Science Publishers, second edition, pp. 457–463.

Sturgis, P.: 2004, 'The Effect of Coding Error on Time Use Surveys Estimates'. *Journal of Official Statistics* **20**(3), 467–480.

Address for Offprints:
Fabrizio Sebastiani
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi 1 – 56124 Pisa, Italy